The Selection of First Programming Language

Dr H T Esendal
Dept of Computer Science
School of Computing and Mathematical Sciences
De Montfort University
Leicester, England

Abstract

The paper summarises the ideas developed by the author towards the selection of a first programming language for students of computing.

The ideas are based mainly on three sources of data. The first is a national survey of all academic and a number of industrial and commercial organisations in the UK, concentrating on their choice of primary programming language. The second is a local survey, of current students and staff in the School of Computing and Mathematical Sciences at De Montfort University, concentrating on an evaluation of Modula-2 as a first language. The third is personal contact with a number of students, chosen at random, to observe the psychological and technological impact that the first programming language has on their perceptions and subsequent behaviour.

The data from the above sources are combined with a review of, first, the benefits and drawbacks of languages currently in vogue and, second, the potential of 'newer' languages to fit the bill better.

To conclude, lists are presented of essential and desirable features of a first language and a number of recommendations are made on how to make the selection.

1. Introduction

This work was prompted by the results of a survey (Latham, 1993) and the staff feedback from a workshop (Perkins, 1993). The survey was conducted in 1992/93, under the supervision of the author, as part of an MSc project, to determine the main (or, if different, first) teaching language of UK academic institutions. The results are presented in the next section. The workshop was organised by the Open University, over two days in September 1993, to discuss the issues surrounding first language selection.

The main points to emerge from the two events were the continuing diversity of first languages in use and the acceptance of paradigms other than the traditional procedural paradigm for teaching programming. The result was a recognition of the increasing difficulties that first language selection would present in the future.

The paper does not promote or condemn any particular language. Neither does it compare or contrast specific languages. Such exercises are useful only when a choice has already been

made, and put into practice, or is about to be made from shortlist of alternatives. The paper proposes, instead, a methodology by which to ease the task of selecting a first language.

The idea of adopting (and, maybe, adapting) a methodolgy for language selection is not new, whether in a general sense (Feuer & Gehani, 1984), or with software engineering principles in mind (Shaw et al, 1984), or emphasizing the implemetation particulars (Wirth, 1984). The main problem with these methodologies is that they tend to concentrate on the technical aspects of languages, with detailed analyses of their features or functions. The proposal of this paper is a methodology that emphasizes the academic issues.

2. A National Survey

A total of 110 universities and colleges were targetted, in order to determine their primary teaching languages. Unfortunately, the overall return rate was a disappointing 26%. While this made the results statistically insignificant, the exercise was still useful in highlighting general trends. The replies indicated the following distribution in the use of first languages.

These results generally confirm the findings of two similar surveys, one in the UK (Furber, 1992) and one in the USA (Morton & Norgaard, 1993). The main difference is that Modula-2 now appears to have replaced Pascal as the popular choice.

It is advisable to be cautious because the results from the surveys mentioned above contradict the findings of another survey (Watts, 1992), which gives the following approximate profile.

In spite of such contradictions, and no matter which language is at the top, or by what percentage, the recurring picture is one of diversity. Moreover, this trend is expected to continue.

3. A Local Survey

The primary teaching language in the School of Computing and Mathematical Sciences at De Montfort University is the Logitech Modula-2 Version 3. A survey was carried out in 1993, involving the students and staff of year one programming modules, to determine their specific reactions to Modula-2.

The survey, to which 27 students and 4 staff replied, was conducted at the end of the second and the beginning of the third semesters. By the time of the survey, the students had one semester of learning to program and one semester of applying their knowledge.

The students were asked how easy they found Modula-2 to learn and what they thought were the 'good' and 'bad' aspects of it as a first language. The staff were asked to concentrate on the manner of first language selection and what factors they thought should play a part. The replies were, again, statistically insignificant due to the relatively low numbers involved. Therefore, the findings were used as guidelines rather than statistical certainties.

The first point to emerge from the student survey was the relative ease with which the majority had learned to program in Modula-2. On a scale of 1 to 5, ranging from hard to easy, the approximate percentage distribution of the replies were as follows.

Beyond this initial ease of learning, however, the students had little to say in favour of Modula-2. They appreciated its support for structured programming and the discipline it encouraged in team work. On the other hand, file handling, printer controls, case sensitivity, debugging tools, and error messages were all mentioned several times as being bad. Poor graphics capability and slow compilation speed were also mentioned by a few. In one extreme case, "nothing" was seen as being good and "everything" was labelled as bad.

As for the 40% who found Modula-2 hard to learn, there is no data available to indicate whether this was due to the language itself or the manner in which programming is taught. This may be the subject of a future study.

The staff, taking a wider view of the issues involved, were primarily concerned with the programming paradigm to adopt and the cost of teaching it via the 'best' language. Two paradigms were mentioned as being worthy of attention: procedural and object oriented; the procedural paradigm for its relative simplicity and the object paradigm for its wide-spread acceptance in industrial and commercial software or systems development.

As regards Modula-2, it was not their favourite language, although they had no problems learning it or teaching it. They liked its strong typing, implementation of modules, and extensibility via libraries. They disliked the inherent weaknesses of its standard libraries (including files and graphics) and lack of relevance to industry or commerce.

4. Further student feedback

The final step was to seek feedback from a number of final year students, chosen at random, to determine how Modula-2 had influenced their perceptions of and behaviour towards programming. These students, 6 in total, from Software Engineering and Business Information Systems courses, had studied COBOL, C, and C++ by now and were expected to make more informed judgements.

They all appreciated the power of modularised software development and the use of Modula-2 as a tool in preparing them for programming in the large. However, they felt unenthusiastic about studying languages that had no direct relevence outside their studies. Consequently, they did not favour using Modula-2 for their final year projects, saying that Modula-2 did not enhance their employment prospects. On a more technical level, the limited and non-standard file input and output facilities on offer were thought to make life unnecessarily difficult for large, realistically complex systems.

5. Looking to the future

The aim of this section is to illustrate, by looking to the past, the difficulties that can be expected in the future. The starting point is the programming paradigm.

The selection of the programming paradigm ("what to teach") will precede the selection of the first language ("how to teach it"). This defines the framework within which the students are then taught. Each of the four possible programming paradigms has received attention, some more than others.

Currently, the most popular is the procedural paradigm, represented by languages such as Pascal and Modula-2. These languages are small in size and are easy to learn. Moreover, their long standing existence has produced a wealth of teaching and learning materials. On the other hand, they have little or no post-education value and can be viewed as reiterating an (almost) obsolete paradigm, given the increasing popularity of object orientation.

Staying within the procedural paradigm but improving the post-education value could lead to C (Mody, 1991). C is popular with students, especially in their final year, as a job hunting tool. This is a particularly important issue in difficult economic times (Citron, 1983). The problem is that C is not a beginners language, with many, some severe, shortcomings (Pohl & Edelson, 1988).

The logical step to take may be towards languages like Extended Pascal (Hetherington, 1993), or Oberon (Radensky, 1993). These languages, being derivates of Pascal and Modula-2, resemble their ancestors but contain modern features. Extended Pascal, for example, is said to increase the range of features greatly without compromising the Pascal security of processing variables. Similarly, Oberon retains the simplicity and efficiency of Modula-2, but supports object orientation, based on type extension.

Staying with or adopting the procedural paradigm should not be seen as a retrograde step because with the right representative language, students could be introduced to the ideas of modularity, information hiding, and objects relatively easily.

There is growing interest in the object paradigm, which, at the moment, is the only serious alternative to the procedural paradigm. Industrial and commercial organisations have accepted system design and implementation using objects. Any move in this direction would, therefore, provide students with a valuable starting point for post-educational credibility.

The main problems are the complexity of the object paradigm, the long lead time needed to get students started, and the battle between pure and impure languages. For example, Smalltalk and Eiffel are pure object oriented languages but have limited practical value beyond education. C++ and Visual Basic have been widely accepted in industry but are considered to be impure, and therefore, lesser languages.

Using Smalltalk as a first language is not without problems (Skublics & White, 1991). Similarly, while some consider Eiffel to be the best language for teaching, as well as for industrial use (Perkins, 1993), others maintain that Eiffel needs a cheap and accessible processor to take off and that it is not yet ready to challenge C++ (Watlins, 1993). And, this is in spite of the many flaws in C++ (Edelson & Pohl, 1989).

The functional programming paradigm has also received some attention. Students have been introduced to programming via special, experimental languages, like FP-X1 (Pagan, 1986) as well as more mainstream languages, like Scheme (Berman, 1994). However, bearing in mind the characteristics of the paradigm and the shortcomings inherent in the implementations of the various functional languages (Bell, Morrey & Pugh, 1987), it is highly debatable that

functional programming has developed and matured sufficiently over the years to warrant its consideration as a first language for computer science students.

The logic paradigm seems to have been abandoned altogether (Perkins, 1993), possibly as a result of the complexities inherent in knowledge representation and modelling, as required by Prolog (Bottino, 1992). The current situation is that neither the functional nor the logic paradigm is a serious alternative to the procedural or object paradigm as a framework for introduction to programming.

When choosing a first language, regardless of the paradigm, there is the added complexity of deciding on a real or customised language. A real language is one that is taught in its entirety (except, maybe, a few very advanced features), according to its standard, via a commercially available environment. The argument in favour of using a real language is that it provides students with a real-life environment. Any problems that they encounter (and, eventually, solve) are those that they will encounter in real life. Hence, using a real language prepares students for their working lives.

A customised language is one that is either a subset of a real language or one that has been developed especially for the purpose of teaching. The decision to use a customised language is based on the understanding that learning to program is a separate issue from the details of a particular language. Therefore, it is better to use a language that is aimed at teaching, thereby minimising the incidental issues that get in the way. Having learned the fundamentals, the students can then apply their programming knowledge to any language.

One approach is to use a series of subsets of the same language, say, one subset per year of study, each one more comprehensive than the previous one. Also, different modules may use special subsets to emphasize particular aspects of the language. By the time the students complete their studies, they will have covered the entire language in depth. Ada, for example, is well suited to this approach (Radensky, 1990), with the recommendation that language subsets be supported by subset compilers, concentrating on error reporting and efficiency, pitched at the right level for the target students.

However, even this approach is not without its problems. Ada has been used as a first language (Radensky et al, 1988), and its potential for advanced programming is acknowledged (Frank & Smith, 1990), thus providing a complete programming tool. Nevertheless, features like I/O with the USE option, strings, loop index variables, and readability can appear to the beginner as inconsistent and, therefore, be difficult to apply (Srinivasan, 1994). The main problem with special languages is availability. They require the development of an environment, which, minimally speaking, must be a compiler. For a custom-built language, development time and effort are the crucial factors, with no guarantee of success at the end. Also, using a customised language creates an artificial environment, which may not exist in real life and may, therefore, give the students a false impression of software design and programming.

6. Recommendations

The message is that almost any language can be used successfully to teach programming, providing there is staff enthusiasm and commitment. However, this success is made easier if care is taken to get the initial selection right and the language meets the expectations of the staff. This is the area of activity for which the following methodology is proposed.

The methodology recommends that a list be drawn up, in a table as shown below, containing the essential and desirable features that the staff think the first language should provide.

Each feature has associated with it a weighting, expressed as a percentage. Weightings represent the relative importance of individual features. The total of all weightings for all features equals 100.

The next step is to assess potential languages. A score, expressed as a value between 0 and 5, is assigned as a measure of how well the language being assessed satisfies a feature. The rank is the product of weighting and score, indicating how acceptable the language is for that feature.

The recommendation is for the same list to be applied to each language under consideration and the one with the highest total rank value to be proposed as the optimum candidate. The methodology is not claimed to be statistically sound.

The following table is offered as an example. The entries were made on the basis of the feedback received from the students and staff who took part in the two surveys mentioned above, combined with the author's views.

The simplicity is intentional. The aim is to use this table as the first hurdle for the potential languages. Should the exercise deliver two or more languages, then a detailed study should be undertaken, maybe along similar lines.

The above list is not claimed to be complete. Also, the same list is not expected to satisfy all. The proposal is for each programming teaching team to compile its own list and assign weightings to each feature, as appropriate, reflecting their philosophy and emphasis.

7. Conclusions

A simple methodology for assessing languages is proposed. It works by identifying staff requirements and quantifying how well languages satisfy those requirements.

The methodology has not been tried yet. The aim of this paper is to invite comments from interested parties. The intention is to improve and extend the methodology, in order to enable its successful application in the near future.

Acknowledgements

The author would like to thank all colleagues and students who have contributed to this work.

References

Bell D., Morrey I., Pugh J., (1987), *Software Engineering, A Programming Approach*, Prentice-Hall International (UK) Ltd, London

Berman A.M., (1994), *Does Scheme enhance an introductory programming course?*, Some preliminary emprical results, SIGPLAN Notices, V29, N2, p44

Bottino, R.M., (1992), *Comparing different approaches to programming from an educational viewpoint*, Computers & Education, V18, N4, p273

Citron, J., (1983), *Computer education in times of recession: Should Pascal come first?*, Computers & Education, V7, N3, p143

Edelson, D. & Pohl, I., (1989), *C++:Solving C's shortcomings?,* Computer Languages, V14, N3, p137

Feuer A.R. & Gehani N.H., (1984), *A methodology for Comparing Programming Languages:* Comparing and Assessing Programming Languages Ada, C, Pascal, A. Feuer, N. Gehani (Editors), Prentice-Hall Software Series, Englewood Cliffs, p197

Frank, T.S. & Smith J.F., (1990), *Ada as a CS-1 and CS-2 language*, SIGCSE Bulletin, V22, N2, p47

Furber, D., (1992), *A Survey of the Teaching of Programming to Computing Undergraduates in UK Universities and Polytechnics*, The Computer Journal, V35, N5, p530

Hetherington, A., (1993)*, An introduction to the Extended Pascal language*, ACM SIGPLAN Notices, V28, N11, p42

Latham, P., (1993), S*ource Code Metrics,* MSc IT Project Report, De Montfort University

Mody, R.P., (1991), *C in education and software engineering*, SIGCSE Bulletin, V23, N3, p45

Morton L. & Norgaard N., (1993,) *A Survey of Programming Languages in Computer Science Programs*, SIGCSE Bulletin, V25, N2, p9

Pagan F.G., (1986), *On the Feasibility of Teaching Backus-style Functional Programming as a First Language,* SIGCSE Bulletin, V18, N3

Perkins G.R., (1993), *Choice of Programming Languages*, De Montfort University Milton Keynes, Research Note RN93-1

Pohl I, & Edelson D., (1988), *A to Z: C Language Shortcomings*, Computer Languages, V13, N2, p51

Radensky A et al, (1988), *Experience with Ada as a first programming language*, SIGCSE Bulletin, V20, N4, p58

Radensky A., (1990), *Can Ada be used as a primary programming language?* Major problems and their solution by means of subsets, 21st Technical Symposium on Computer Science Education, SIGCSE Bulletin, V22, N1, p201

Radensky A., (1993), *A voyage to Oberon*, SIGCSE Bulletin, V25, N3

Shaw M et al., (1984), *A Comparison of Programming Languages for Software Engineering,* Comparing and Assessing Programming Languages Ada, C, Pascal, A. Feuer, N. Gehani (Editors), Prentice-Hall Software Series, Englewood Cliffs, p209

Skublics S. & White P., (1991), *Teaching Smalltalk as a first programming language*, 22nd Technological Symposium on Computer Science Education, SIGCSE Bulletin, V23, N1

Srinivasan S., (1994), *A critical look at some Ada features*, ACM SIGPLAN Notices, V29, N3, p18

Watkins A., (1993*), Eiffel/S 1.2: Is the waiting over?*, EXE, V7, p18

Watts W., (1992), *La Resistance*, EXE, V6, p28

Wirth N., (1984), *Programming Languages: What to DemandR and How To Assess Them*, Comparing and Assessing Programming Languages Ada, C, Pascal, A. Feuer, N. Gehani (Editors), Prentice-Hall Software Series, Englewood Cliffs, p245