Language Trends in Introductory Programming Courses

Michael de Raadt, Richard Watson and Mark Toleman
University of Southern Queensland, Australia

deraadt@usq.edu.au rwatson@usq.edu.au markt@usq.edu.au

Abstract

Deciding what to teach novice programmers about programming and, in particular, which programming language to teach to novice programmers, and how to teach it, is a common topic for debate within universities. Should an industry relevant programming language be taught, or should a language designed for teaching novices be used? In order to design tools and methodologies for the teaching of novice programmers it is important to uncover what is being taught, and in turn, what will be taught in the future. A census of introductory programming courses administered within all Australian universities has been undertaken. The census aimed to reveal not only what computer programming languages are being taught, but also how they are being taught. From the results of this census two key factors emerged: perceived industry pressure for graduates with certain language skills versus academic training for generic programming skills.

Keywords: novice programming, teaching programming languages

Introduction

Since the inception of introductory programming courses, there has been significant debate about how to teach programming to novices. When constructing an introductory programming course, instructors must consider what language to teach, how their course will benefit their students' programming skills for later study or employment, what paradigm should be used and what tools, if any, could assist in the teaching of their students. The intention of this paper is to examine how these issues are being dealt with in universities.

Pham (Pham, 1996) discussed the pressures on universities that affect the 'purpose' of computing courses. These included advancing technology, demand for industry relevant skills and government pressure through funding of universities to "cater for a mass clientele". Many papers express experiences of conducting a single introductory programming course, (eg. Clark, MacNish & Royal, 1998, Hagan, 1997), but these do not reflect the population of introductory programming courses, and do not allow for analysis of trends in this academic endeavour.

Prior to 1970, languages that could be taught to novices were limited to those available, including FORTRAN and Cobol. In 1971 Niklaus Wirth introduced the language Pascal (Wirth, 1971) specifically for teaching novices programmers. At various times between 1971 and 1997, 92% of universities in Australian taught Pascal. In 1995, Levy (Levy, 1995) reported a movement away from Pascal claiming it was no longer capable of demonstrating all necessary concepts and was not a commercial language, this while Java was just being released. In 1997, Pascal was taught in an Australian university for the last time.

In the wake of the decline of Pascal, the current study was conceived to discover exactly what has filled the vacuum created in introductory programming.

Courses investigated by this study assume no prior programming experience and run for one semester (usually the first semester within a first year undergraduate program). The courses cover the basics of programming, including algorithms and problem solving, sequence, selection, repetition and data types. These are typical courses worldwide so the results of this study are widely applicable.

Surveys have been conducted involving introductory programming courses within Australian universities (McDonald, 1999, Robins, 1998). While fulfilling their goals of revealing language choice decisions, these studies did not cover trends in language choice, types of students taught, paradigms taught, and did not provide a context for examining these choices. These surveys covered only a small part of the population of Australian universities.

Because of the number of Australian universities (thirty-nine) it was possible to carry out a comprehensive *census* rather than a more common, less reliable *survey*. The census sought to address the following questions.

- What programming languages are being taught?
- Are universities teaching industry relevant languages or are they using teaching languages?
- Does language choice depend on the kind of university? In particular, do older, more research-oriented universities differ from their younger counterparts on this choice?
- Is there a distinction in the languages taught to students studying for different disciplines?
- How long do universities use an introductory language before it is updated?
- Do Object Oriented languages dominate introductory programming, and are they taught using an 'Object Early' approach (where objects and classes are presented from the beginning of the course)?
- What tools are being used when teach novices in introductory programming courses?

This paper is organised as follows: the construction of the census and how it was undertaken is described in the next section; results from the census are presented and analysed in the following section; and finally, conclusions and possible future work are suggested.

Census: Trends in Novice Programming

The census covered all thirty-seven of the thirty-nine universities that offered introductory programming courses. The census was undertaken during the first half of 2001.

Construction of the Census

A list of questions was drafted, refined and piloted. In an attempt to answer the research questions posed in the introduction, the questions asked in the census were as follows.

1. What programming language is being used?
2. Why was this language chosen?
3. Are there plans to change the language?
4. Which languages were taught previously in the course and when did use of the current language start?
5. For what type of student is your first programming course designed?
6. How many students are currently undertaking this course?
7. Are environments and/or tools beyond simple editors and command-line compilers used to support teaching of the language in practical sessions?
8. What paradigm is being taught using the language (regardless of what is traditionally thought to apply to this language)?

Method of Data Collection

In order to determine who should be asked to participate, a list of universities that offer degrees accredited by the Australian Computer Society (Australian Computer Society, 2000) was used. An attempt was made to cover all introductory programming courses within each university, not just those taught in computer science schools/departments.

To maximise the participation rate, the census was conducted by telephone. All people who were asked agreed to participate.

Results and Discussion

A summary of the results is shown in Table 1. In terms of student participation, 19,900 implies approximately 4% of all undergraduates were studying an introductory programming course during the first half of 2001. Participants were asked when they had started using the current language in their teaching. The figure for 'average years of using current language' is a measure of this.

Courses	57
Universities	37
Students (Approx)	19,900
Average Students per Course	349
Average Years of Using Current Language	4.13

Table 1 General results.

Languages Currently Taught

Nine different languages are being taught in Australian universities. Participants were asked to indicate what language was taught prior to these current languages. The number of 'dropped' languages was eighteen (double the current number). Language diversity has reduced; analysis of the census data shows that 18 languages were taught in 1996, 17 in 1997, 16 in 1998, 14 in 1999, 11 in 2000 and 9 in 2001. Table 2 shows the number of courses using particular languages currently, and the number of courses that used particular languages prior to their current language. Six of these nine current languages (Java, VB, C++, C, Eiffel, Delphi and Jbase) are widely used in industry. By student numbers 86% of students are being taught an industry relevant language.

Language	Currently Used	Dropped
Java	23	1
Visual Basic	14	2
C++	8	6
C	4	8
Haskell	3	1
Eiffel	2	
Ada	1	4
Delphi	1	1
JBase	1	
Pascal		13
Modula 2		3
Smalltalk		3
Miranda		2
Others (Basic, Blue, Cobol, DBase, Gopher, Turing)		(6)

Table 2 Languages currently and previously taught.

Pascal is no longer taught in any Australian university, nor are its descendants (Modula, Oberon or Component Pascal). One course uses Delphi, however the participant responsible for this course indicated that they would be moving to C++ at the end of the current run of the course.

Use of languages, weighted by the number of students taught these languages, is shown in Figure 1. The most widely taught language is Java, followed by Visual Basic, although, if C and C++ are combined (as they are often taught interchangeably), then this surpasses VB.

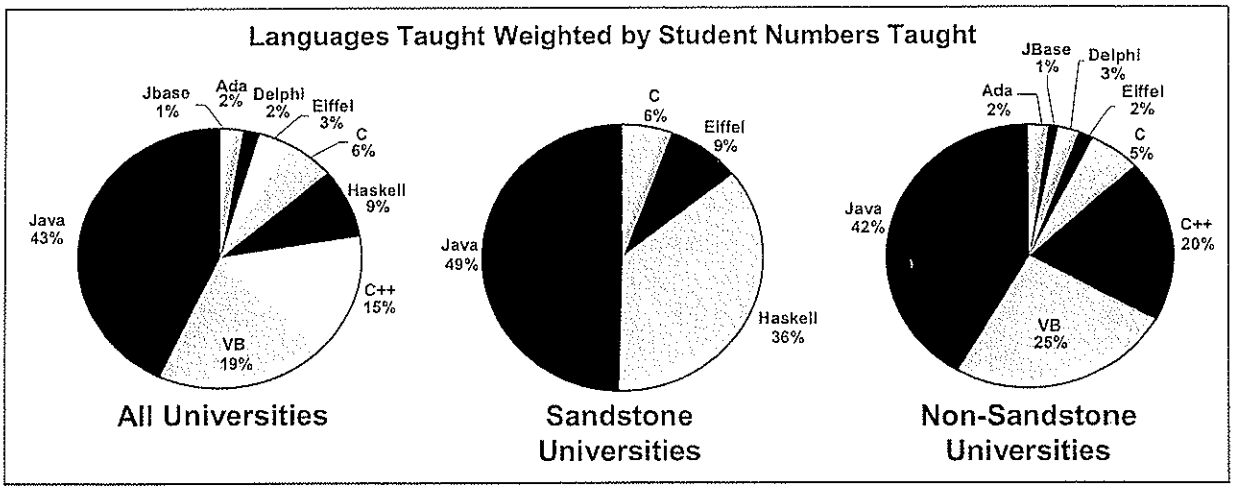

Figure 1 Use of languages weighted by student numbers

Worthy of note is the fact that four of the six courses teaching 'non-commercial' languages (Ada, Eiffel and Haskell) are run within the 'Sandstone' universities (Australian universities established before 1950 (Ashenden & Milligan, 1999)). If these universities are excluded, the use of non-commercial languages, weighted by student numbers, falls from 14% to 4%. There is a clear distinction between the Sandstone universities and other universities in the languages they teach. These Sandstone universities are ranked in (Ashenden & Milligan, 1999) as the highest for 'Prestige' and 'Student Demand'. This suggests that universities not competing for students are more confident in choosing non-commercial languages.

Participants were asked to indicate why they had chosen their particular language. The responses are summarised in Table 3. The reasons given by participants for choosing the language they are currently teaching was dominated by a willingness to satisfy the perceived need to teach a language that will provide their graduates with marketable skills. To most participants this was more important than the pedagogical benefits available in the language they had chosen to teach.

As well as choosing different languages, Sandstone universities made their choices using different criteria, as shown in Table 4. This distinction reinforces the difference between Sandstone and non-sandstone universities.

Industry relevance/Marketable/Student demand	33
Pedagogical benefits of language	19
Structure of degree/Department politics	16
OOP language wanted	15
GUI interface	6
Availability/Cost to students	5
Easy to find appropriate texts	2

Table 3 Count of reasons given for language choice in all universities.

Types of Students Taught

Introductory programming courses target students from a range of disciplines. Census participants nominated student discipline groups for which their course was designed. The disciplines recorded were Computer Science/Information Technology, Engineering, Business, and Other. Many courses were designed for a range of disciplines. The proportion of students undertaking the courses based on kind of course are as follows:

- Computer Science/IT only 26.5%
- Computer Science/IT & Engineering 20.7%
- All disciplines 16.5%
- Business and Other 8.5%
- Engineering only 4.7%
- Other combinations 20.1%

Pedagogical benefits of language	6
Industry relevance/Marketable/Student demand	4
Structure of degree/Department politics	2
Availability/Cost to students	2
OOP language wanted	2
Easy to find appropriate texts	1

Table 4 Count of reasons given for language choice by participants from 'Sandstone' universities.

An examination of the types of languages being taught within courses designed with specific types of students in mind shows why these languages may have been chosen.

- The popularity of Java is uniform across all disciplines.
- Visual Basic is taught widely within courses designed exclusively for business or other (non-engineering/non-computer-science/non-IT) students; Visual Basic is taught in 78% of such courses.
- Courses for Computer Science and Engineering students show a higher use of C++ and Haskell.

Trends in Language Use

Participants were asked if they had definite plans to change the language they were teaching. Only five of the fifty-seven participants indicated that they had definite plans to change, although many participants stated that the language taught was constantly under review. Those who did indicate they had definite plans to change the language taught were not consistent in the language to which they were switching. For example, one participant indicated they were planning to change from VB to Java, while another indicated the opposite. Efforts to predict future trends are therefore limited to a study of the past.

There appears to be very little correlation between language previously taught and language currently taught. Although it might be expected that instructors would chose new languages with the same paradigm or similar language features, this is not the case. Instead, these decisions appear to be more motivated by reasons shown in Table 3 and Table 4.

When courses are grouped by language and measured by the average length that courses have used a particular language then the results are as shown in Figure 2.

In this figure the average length that each language has been taught, measured in years, is indicated by the horizontal width of each bar and the value next to each bar. The number of courses teaching this language is indicated by the vertical height of each bar and the value in parentheses next to each bar.

Of note is that Java is the most widely taught language, and it has been taught, on average, for only a short period of time.

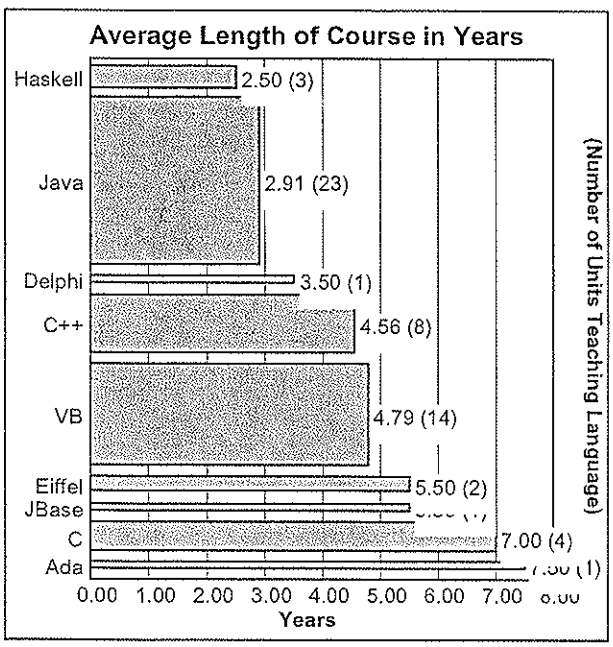

Figure 2 Average length of language use and number of units teaching that language

Paradigm Taught

Participants were also asked what paradigm they were employing in their teaching. In many cases, paradigm was restricted by language taught, but of great interest is the distinction between OOP languages that are taught initially using an Object Early approach or a Procedural approach. Over half of all students are initially taught using a Procedural approach, 40% using an Object Early approach, and 9% using a functional paradigm. However, 81% of students are being taught OOP languages. Courses teaching OOP languages, and the paradigms initially used within these courses, are broken down by language as show in Figure 3.

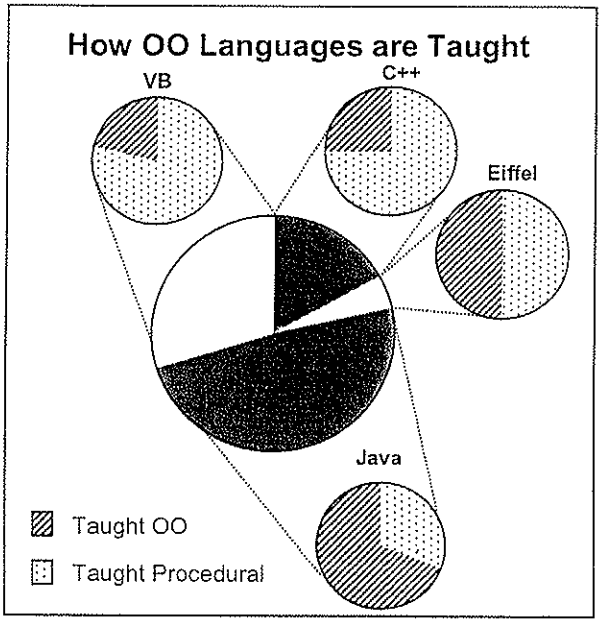

Figure 3 Paradigm used to teach OO languages

Early calls for a transition to teaching the Object Paradigm caused much debate (Decker & Hirshfield, 1992, Wallingford, 1996) which continues today. Some authors acclaim the benefits of teaching OOP languages, but express disappointment at the less than adequate suitability of commercial OOP languages for teaching (Andreae et al, 1999, Kölling, Koch & Rosenberg, 1995). OOP languages have become widespread in industry and these languages are taught widely in Australian universities. Teaching methods, however, have not changed as rapidly. Some 86% of languages taught are OOP languages, but less than half are taught using an Object Early approach. An exception to this is the teaching of Java, which is taught with an early introduction of objects and classes in 70% of courses that teach it. Teaching non-OO Java may seem impossible to some, but means that 30% of instructors are asking their students to ignore class declarations in Java until later.

Environments and/or Tools Used

Participants were asked to indicate what tools they used to assist teaching in practical sessions, other than simple editors and command-line compilers. Figure 4 indicates the environments/tools being used. Some languages are limited to environments in which they can be taught, but the greatest number of participants answered they were avoiding using such technologies, indicating the following reasons:

- cost for students,
- time required to familiarise students with environments, and
- the blurring of distinct steps in the programming process.

Courses not teaching a language that force the use of an integrated environment, continue to use facilities such as text editors and command-line compilers, that have been available since the inception of the introductory programming course. This is despite the existence of many more sophisticated programming tools and environments used by professional programmers. Clearly there is a lack of tools that are designed specifically for novice programmers, are freely available, easy to use, do not obscure the details of the programming process, and in which instructors can be confident in teaching.

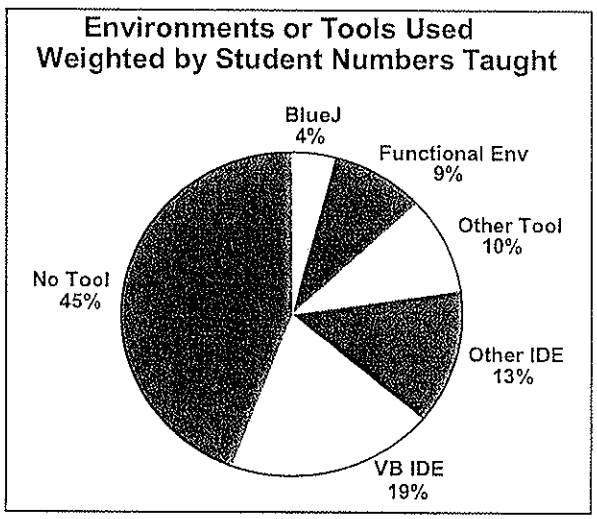

Figure 4 Environments and/or tools used for teaching in practical sessions.

Conclusions and Future Work

In conclusion, the questions raised in the introduction are addressed and future research directions are considered.

What languages are being taught?

The main languages being taught are Java, Visual Basic, C/C++ and Haskell. A trend in the reduction of the number of languages being taught was noted. Perceived industry pressure and pedagogical reasons are directing universities towards a smaller pool of languages. It would be reasonable to concede that this trend will continue, perhaps until only the four languages above remain. A study is planned within two years to follow this trend.

Are universities teaching industry relevant languages?

In most cases, universities are making compromises that they feel will produce the greatest benefits for their graduates. The proportion of students being taught an 'industry relevant' language is 86%. The first reason given by most participants for choosing a language was perceived industry demand, or pressure from students for a commercial language. This does not indicate that participants actively surveyed industry demand for particular languages. A study of how the programming languages perceived as demanded by industry differ from actual demand is a possible future direction.

Is there a distinction between Sandstone universities and other universities?

Different languages are being taught in Sandstone universities when compared to non-sandstone universities. Sandstone universities also have different priorities for choosing a language. Courses within these universities are in great demand from students, which permits a focus on what will benefit them best pedagogically rather than what will attract students. In the future other universities may take the lead of the sandstone universities and teach academic languages, or the sandstone universities may take on more commercial languages. A planned study within two years will reveal if either of these possibilities has occurred.

Is there a distinction between languages taught to students of different disciplines?

Java is popular across all disciplines. Java is a popular language seen as relevant to industry. Visual Basic is popular within courses designed for students of non-technical disciplines. The GUI/event-driven features of VB allow interesting solutions to be created quickly. Some participants involved in courses from non-technical disciplines stated that this was important, as students in these courses are not likely to attempt any further programming courses. C/C++ and Haskell are popular within courses designed for students of technical disciplines. These languages allow a focus on computing principles necessary for later programming.

Introductory programming is no longer taught exclusively within computer science settings. Future studies may examine the growth of introductory programming courses outside computer science, and what distinguishes these courses.

How long do universities use an introductory language?

From examining the use of current languages, at least four years is the average life of a language used in a course. Repetition of this study will provide longitudinal trends of this factor. At this stage however, there seems to be no new languages on the horizon capable of overthrowing the current most popular languages.

Does an Object Paradigm dominate?

Over eighty percent of students are being taught OOP languages. This may be because of perceived popularity of these languages or perhaps simply a need to teach object oriented languages. With this in mind, however, over half of the courses using these languages are not using an Object Early approach. One possible reason for this is the experience of instructors who previously taught using a procedural paradigm and, while accepting an OO language, are reluctant to adopt an Object Early approach. A future study may discover exactly why this is the case.

What tools are being used?

Of participants who were not restricted to an environment by the language they teach, the majority indicated that they chose to use simple text editors and command-line compilers. Participants reasoned that complex environments require additional instruction, which consumes valuable teaching time. Future research into programming tools designed specifically for novices would be wise to consider the reasons given by participants for not choosing a tool, before creating such tools.

A future study, planned within two years, will reveal further trends, and add more longitudinal data to this study. Additional questions may be added to reveal:

- Is there a distinction in the instruction provided to non-technical students?
- What methods of assessment are being used?
- How many hours are dedicated to teaching of algorithms and problem solving within the course?

Industry, it seems, can influence languages taught within tertiary institutions, and the languages taught in tertiary institutions in turn influence the languages used within industry. But which comes first? Is it the chicken or the egg? Who is making the decisions that will guide the future of programming? Some suggest that we are driven toward debate over technological issues of introductory programming when we should be focussing on teaching algorithms and problem solving. Should universities teach generic skills that can be applied to many languages, or should they teach specific languages in a technology-based manner? Is it even necessary to introduce any specific language at all in an introductory course (Lister, 2000)?

References

Ashenden, D. & Milligan, S (1999). *The good universities guide: Universities, TAFE and private colleges in 2000.* Hobsons Australia.

Andreae, P., Biddle, R., Dobbie. G., Gale. A., Miller, L. & Tempero, E (1999). *Surprises in teaching CS1 with Java (Technical report).* Online. Internet. [1999]. Retrieved September 5, 2001 via FTP: ftp://ftp.mcs.vuw.ac.nz/doc/vuw-publications/CS-TR-98/CS-TR-98-9.ps.gz

Australian Computer Society (2000). *Accredited courses.* Retrieved August 29, 2001 on the World Wide Web: http://203.58.197.209/acs/events_admin/course20_6.htm

Clark, D., MacNish, C. & Royle, G.F (1998). Java as a teaching language--opportunities, pitfalls and solutions. *The proceedings of the third Australasian conference on computer science education* (July 1998), ACM Press, 173-179.

Decker, R. & Hirshfield, S (1992). A case for, and an instance of, objects in CS1. *Addendum to the proceedings on Object-oriented programming systems, languages, and applications* (Addendum) (October 1992), ACM Press, 309-312.

Hagan, D. Monitoring and evaluating a redesigned first year programming course. *Proceedings of the conference on Integrating technology into computer science education* (June 1997), ACM Press, 37-39.

Kölling M., Koch, B. & Rosenberg, J. (1995). Requirements for a first year object-oriented teaching language. *Papers of the 26th SIGCSE technical symposium on computer science education* (March 1995), ACM Press, 173-177.

Levy, S.P. (1995). Computer language usage in CS1: Survey results. *SIGCSE Bulletin*, 27, 21-26.

Lister, R. (2000). On blooming first year programming, and its blooming assessment. Proceedings of the Australasian computing education conference. (December 2000), ACM Press, 158-162.

McDonald, C. (1999). *1st year programming languages in Australian and New Zealand universities.* Retrieved June 26, 2001 on the World Wide Web: http://www.cs.uwa.edu.au/~chris/java-in-cs1/anzacs.html

Pham, B. (1996). The changing curriculum of computing and information technology in Australia. *Proceedings of the second Australasian conference on computer science education* (July 1996), ACM Press, 149-154.

Robins, A. (1998). *First language survey.* Retrieved June 15, 2001 on the World Wide Web: http://www.cs.otago.ac.nz/survey/surveyhome.html

Wallingford, E. (1996). Toward a first course based on object-oriented patterns. *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education* (February 1996), ACM Press, 27-31.

Wirth, N. (1971). The Programming Language Pascal. *Acta Informatica*, 1, 35-63.

Biographies

Michael de Raadt is a PhD student and instructor of programming at the University of Southern Queensland. Michael undertook undergraduate study at the University of Western Sydney and achieved his Bachelor of Applied Science Degree with Distinction in 1998, and achieved First Class Honours and was awarded the UWS University Medal in 1999. Michael is also a recipient of the ACS prize for Highest Achievement.

Dr Richard Watson is a lecturer in the Department of Mathematics and Computing at the University of Southern Queensland. He has taught programming to undergraduates at all levels for the past 12 years. He conducts research into functional programming languages.

Dr Mark Toleman is an Associate Professor of Information Systems at the University of Southern Queensland where he has taught computing subjects to engineers, scientists and business students for 15 years. He has a PhD in computer science from the University of Queensland and is an Associated Academic of the Software Verification Research Centre there.