An Analysis of Introductory University Programming Courses in the UK

ABSTRACT

This paper reports the results of a survey of over70 introductory programming courses delivered at UK universities as part of their first year computer science (or similar) degree programmes, conducted in the first half of 2016. Results of this survey are compared with a related survey conducted since 2010 (as well as earlier surveys from 2001 and 2003) on universities in Australia and New Zealand. Trends in student numbers, programming paradigm, programming languages and environment/tools used, as well as the reasons for choice of such are reported. Other aspects of first programming courses such as instructor experience, external delivery of courses and resources given to students are also examined.

The results indicate a trend towards...

Categories and Subject Descriptors

K.3.2 [**Computers & Education**]: Computer and Information Science Education—*Computer Science Education*; K.4.1 [**Computers And Society**]: Public Policy Issues

Keywords

Introductory Programming, Programming Languages, Programming Environments, Computer Science Education, Higher Education, Tertiary Education, UK

1. INTRODUCTION

For many years – and increasingly at all levels of compulsory and post-compulsory education – the choice of programming language to introduce basic programming principles, constructs, syntax and semantics has been regularly revisited. Even in the context of what are perceived to be the most difficult introductory topics in computer science degrees, numerous key themes across programming appear [2].

So what is a good first programming language? The issues surrounding choosing a first language [3, 5] – and a search of the ACM Digital Library identified a number of papers of the form "*X as a first programming language*", going as far back as the 1980s – as well as the potential impact on students' grades and attainment [4].

Decades of research on the teaching of introductory programming has had limited effect on classroom practice [10]; although relevant research exists across several disciplines including education and cognitive science, disciplinary differences have made this material inaccessible to many computing educators. Furthermore, computer science instructors have not had access to comprehensive surveys of research in this area [8, 10].

However, in Australasia there has been substantial longitudinal data collection [11, 7, 6] surveying the teaching of introductory programming courses in higher education. However, such surveys have not been conducted elsewhere on this scale, and this paper reports the findings from running the first such similar survey in the UK.

We remind the reader that the UK consists of four nations historically ruled by one parliament, with an overall population of 64.5 million: England (population: 54.3 million), Scotland (5.3 million), Wales (3.1 million) and Northern Ireland (1.8 million) [9]. In 1997, Scotland and Wales held referendums which determined in both cases the desire for self-government, with the creation of assemblies to which a variety of powers – in particular, education – were devolved from the UK Parliament. In the context of international focus on curriculum and qualification reform to support computer science education and digital skills in schools, the four education systems of the UK have proposed and implemented a variety of changes [12, 1], particular in England, with a new compulsory computing curriculum for ages 5-16 from September 2014.

For universities across the UK offering computer science degrees, this curriculum reform in schools has had uncertain (and emerging) impact on their undergraduate programmes, with the diversity of educational background of applicants likely to be increasing before it narrows: it is certainly possible for prospective students to have anywhere from zero to four or five years experience (and potentially two school qualifications) in computer science with some knowledge of programming.

Over the past three years, there has been increased scrutiny of the quality of teaching in UK universities, partly linked to the current levels – and potential future increases – of tuition fees (generally paid by the student though government-supported loans), as well as the perceived value of professional body accreditation and graduate employability, especially for STEM disciplines. In February 2015, the

UK Department of Business, Innovation & Skills and the Higher Education Funding Council for England (HEFCE) initiated independent reviews of STEM degree accreditation and graduate employability[1], with a specific review – the Shadbolt review [13] – focusing on computer science degree accreditation and graduate employability, reporting back in May 2016. Alongside a number of recommendations, the Shadbolt review split generalist universities in England into three bands, based on their average (across all subjects) entrance tariff (qualifications of entrants); we have followed that banding during our analysis the English results, so our data should allow comparisons.

2. METHODOLOGY

2.1 Recruitment of Participants

To recruit participants for the survey, a general invitation email was sent to the Council of Professors and Heads of Computing (CPHC) mailing list. CPHC have members from over 100 UK universities, and are the representational body for this group in the UK[2].The invitation asked for the survey to be passed on to the most appropriate person to fill it out; Director of Studies, Chair of Teaching Committee or the best fit for the individual institution. A couple of reminder circular e-mails were also issued.

The survey was hosted online, and was open from mid-May until the end of June 2016, at which point it was closed and the results were downloaded and analysed. Due to the recruitment method, there were duplicate responses from some departments, and these were reconciled by direct enquiry.

2.2 Questions

The questions used in the survey were generously provided by the authors of [6], so as to allow direct comparison between the results of this survey and that of the Australian/New Zealand 2014 survey. Where possible, questions were left unchanged, although a small minority were edited to reflect the UK target audience.

As defined in [6], text in the survey made clear that the terminology "course" was used for "the basic unit of study that is completed by students towards a degree, usually studied over a period pf a semester or session, in conjunction with other units of study".

The first section of the survey asked about the programming language(s) in use, the reasons for their choice, and the perceived difficulty and usefulness of the programming language(s). Following this were questions regarding the use of environments or development tools; which ones were used, the reasons for their choice and the perceived difficulty. General questions about paradigm, instructor experience and external delivery were asked, along with questions regarding students receiving unauthorised assistance, and the resources provided to students. Finally, participants were asked to identify their top three main aims when teaching introductory programming, and were also allowed to provide further comments.

In [6], participants were asked to rank the importance of the given reasons for choosing a programming language,
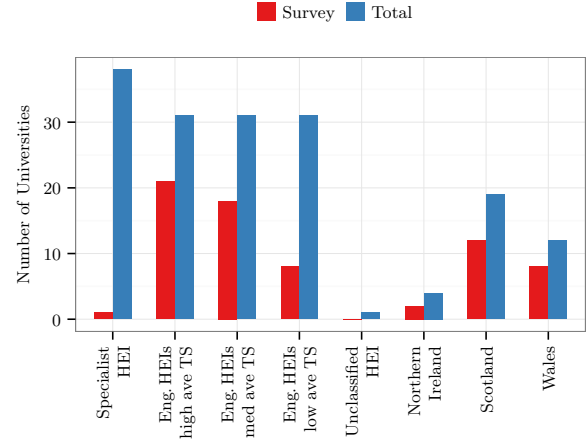
Figure 1: The number of responding universities per Nation/ Tariff Group.

Table 1: The number of programming languages used in first programming courses.

Languages	1	2	3	4
Courses	59	17	3	1

environment or tool. Due to technical limitations in online survey tool used, it was not possible to do so in this survey, so our Figure 3 just reports counts.

Most questions were not mandatory; the exceptions were "what programming language(s) are in use?" and a small number of feeder questions to allow the survey to function correctly.

3. RESULTS AND DISCUSSION

3.1 Universities and Courses

Upon completion of the survey, 155 instructors had, at least, started the survey. Sixty-one of these dropped out before answering the mandatory questions, and a further 14 were duplicates. Therefore, the results presented here are drawn from the responses of 80 instructors from at least 70 universities. Some participants did not answer all questions and due to this the response rate varies by question.

Excluding the Open University's 3200 students, the participants in the survey represented 13462 students, with a mean of 173 (but a standard deviation of 88). Looking at Figure 1 we see good response rates, apart from the specialist HEIs (most of whom do not teach computing) and the "low tariff" English ones. Fewer of these teach computing, this factor alone explains the response rate. In Northern Ireland, we had responses from the two universities, but not the university colleges, which are historically teacher-training colleges.

3.2 Languages

This is the immediate major difference with [6]. Their survey showed a dead heat (27.3% of courses) between Java and Python, with Python winning (33.7% to 26.9%) when weighted by the number of students on the course. Our findings (Figure 2) show that Java is a clear winner by any
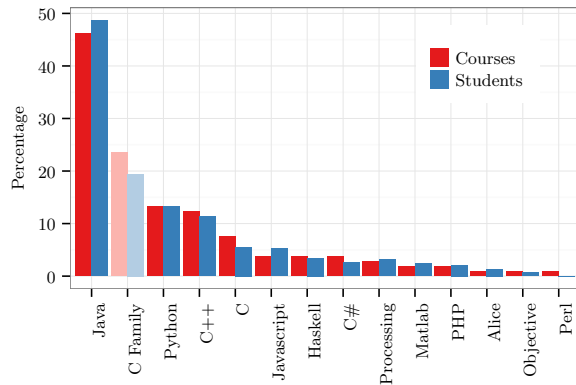
Figure 2: Language popularity by percentage of courses and students (excl. OU).

Table 2: The main paradigm in use in the first programming course.

Paradigm	Object-Oriented	Procedural	Functional
Courses	40	27	7

metric, being used in just under half the courses, while the runner-up, Python, is in use in 13.2%. The C family (C, C++ and C#) together scores 23.6% by courses and 19.5% by students. Figure 2 shows the effect of student-number weighting *but* we have excluded the Open University from this weighting, as its 3200 students learning Python (and Sense, a variant of Scratch) would have distorted the comparison.

Figure 3 shows some of the reasons for this: Java scores higher on "relevance to industry" and, perhaps somewhat surprisingly, much higher on "Object Oriented language".

Figure 5 breaks down the choice of language by nation and tariff group. It is noticeable that the three English tariff groups differ significantly, with Python outnumbering Java in the low tariff universities, and C being almost exclusively in the high tariff universities.

Table 3: The number of years the instructor has been involved in teaching introductory programming.

Years	<2	2 - 5	5 - 10	10 - 20	20 - 30	>30
Instructors	3	9	9	27	19	7

3.3 Instructor Experience

Participants were asked: "How many years have you been involved in teaching of introductory programming?". The results, shown in Table 3, indicate that of the survey participants, the average was between 10 - 20 years.

3.4 IDEs and Tools

Various data about IDEs and tools are collected in Figures 8–11. We note that, while Eclipse is the most popular tool by some way (Figure 8), it is also deemed to be most difficult (Figure 11). This, apparently perverse, practice might be explained by the extent of re-use of Eclipse in other courses (Figure 10).
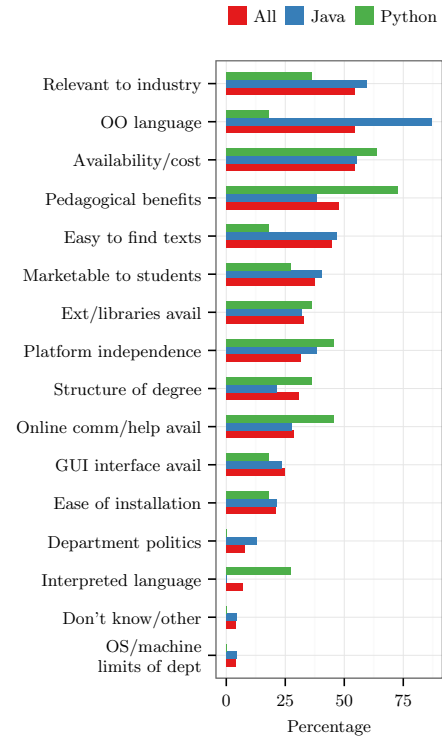

Figure 3: Reasons given for choosing a programming language by percentage for: all languages; Java; and Python.
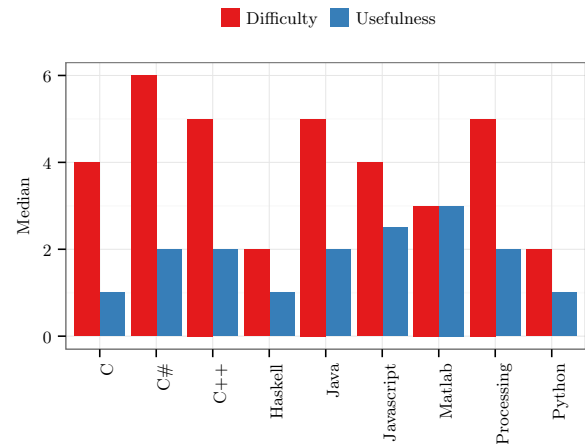

Figure 4: The median of the perceived difficulty and [pedagogic] usefulness of language, where 1 is 'extremely easy/useful' and 7 is 'extremely difficult/useless'. Answers must have been given by at least two instructors.

Table 4: The number of tools/environments used in first programming courses.
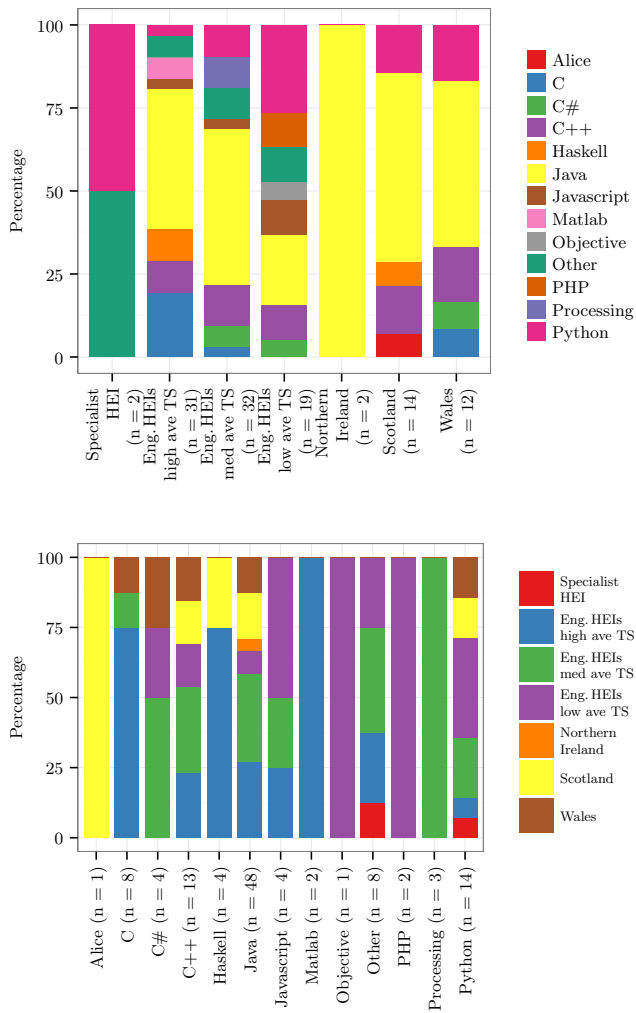
Tools	1	2	3	4	8
Courses	34	15	6	2	1

Figure 7: The breakdown of the main paradigm in use for each programming language.

Figure 5: The breakdown of programming languages by Nation and Tariff Groups.
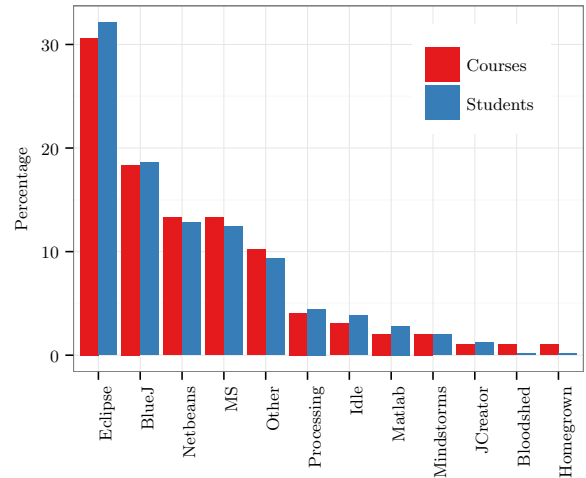
Figure 8: Tool or environment popularity by percentage of courses and students.
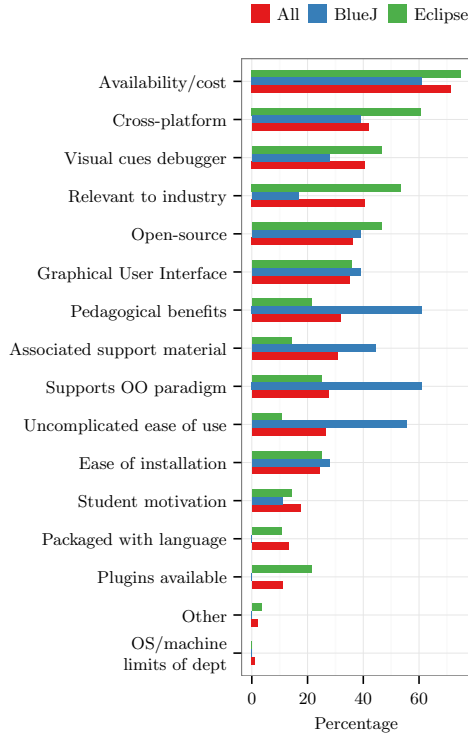

Figure 6: The breakdown of the main paradigm in use for every Tariff Group.

Figure 9: Reasons given for choosing a tool or environment by percentage for: all tools and environments; BlueJ; and Eclipse.
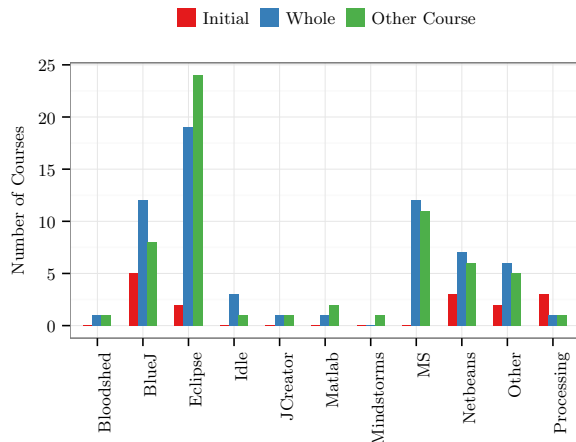

Figure 10: For each tool or environment, whether it is used: for an initial part of the first programming course; throughout the whole of the first programming course; in any other course in the degree.
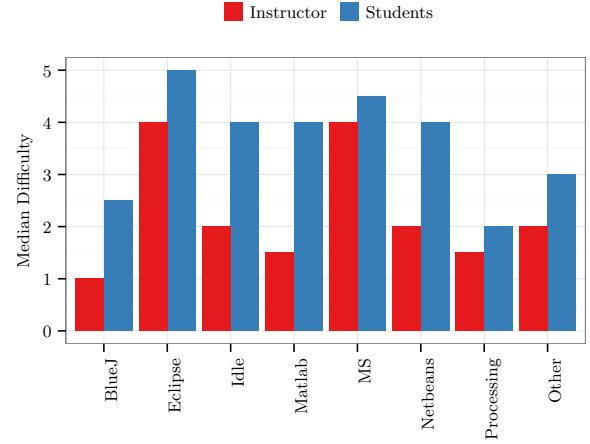

Figure 11: The median difficulty rating of tool/environment for the instructor and students to use, where 1 is 'extremely easy' and 7 is 'extremely difficult'. Answers must have been given by at least two instructors.

3.5 Other Aspects of the Course

The questionnaire asked about the resources in terms of examples, books etc. provided to students. The results are rather similar to [6, Figure 14] so we do not repeat that here: details are in the full paper.

Conversely, [6] asked about **@Ellen: exact question please** but didn't give the results in their paper. We report our results in Figure 12, as we think they are of general interest.

3.6 Aims of an Introductory Programming Course

[6] asked their respondents for the aims of their introductory programming course. They, and we, asked for (up to) three aims. The authors then attempted to classify the free-text answers into the same categorisation as [6] used. While it is trivial to map the written aim "Thinking algorithmically" to [6]'s "Algorithmic thinking" and so on, many were not so clear: for example, we mapped "To learn a specific language" to "syntax/writing basic code". There were also a class of aims, such as "Establish professional software development practices", that seemed very coherent, but didn't map clearly to the [6] aims. These we have categorised as "Software Engineering".

4. GENERAL DISCUSSION

4.1 The U.K. context

4.2 Comparison with Australasia

Here we compare with [6], the latest Australasian survey. We have already commented on the major difference in language choice, which colours many of the other comparisons. n fact, the U.K.'s language choices seem more similar to Australasia's 2010 choices [7] and [6, Table 4] than even Australasia's 2013 choices. It is hard to know which comes
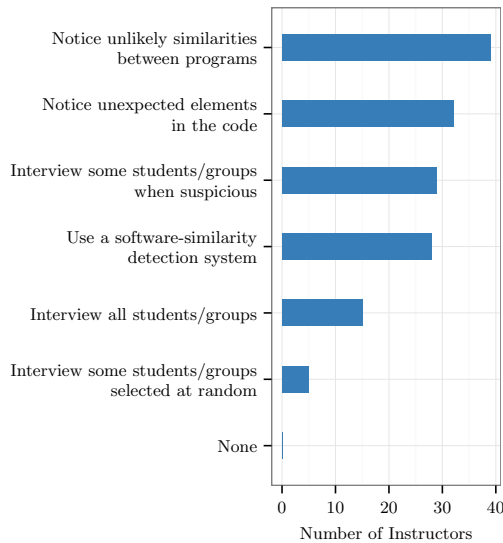
Figure 12: Steps taken to determine whether students have received unauthorised assistance on assignments.
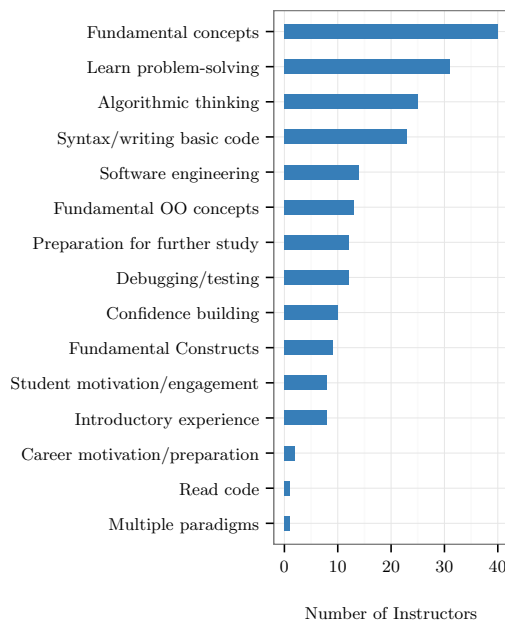

Figure 13: Aims of the introductory course.

first, but we also notice that our difficulty/utility data (Figure 4) is somewhat different from [6, Figures 7,8]

Another difference shows up in the tools/enviroments are: Figure 8 versus [6]'s Figure 11. There, "None" and "Other" were the top two categories, with Idle, at 15%, the most popular named product. In the UK, "Other" is fifth and Idle seventh, with no "None" @Ellen: is that right?.

5. ACKNOWLEDGEMENTS

6. REFERENCES

[1] N. Brown, S. Sentance, T. Crick, and S. Humphreys. Restart: The Resurgence of Computer Science in UK Schools. *ACM TOCE*, 14(2):1–22, 2014.

[2] N. B. Dale. Most difficult topics in CS1: results of an online survey of educators. *ACM SIGCSE Bulletin*, 38(2):49–53, 2006.

[3] D. Gupta. What is a good first programming language? *Crossroads*, 10(4), 2004.

[4] M. Ivanović, Z. Budimac, M. Radovanović, and M. Savić. Does the choice of the first programming language influence students' grades? In *Proceedings of the 16th International Conference on Computer Systems and Technologies*, pages 305–312, 2015.

[5] R. M. Kaplan. Choosing a first programming language. In *Proceedings of the ACM Conference on Information Technology Education*, pages 163–164, 2010.

[6] R. Mason and G. Cooper. Introductory programming courses in Australia and New Zealand in 2013 – trends and reasons. In *Proc. 16th Australasian Computing Education Conference (ACE'14)*, pages 139–147, 2014.

[7] R. Mason, G. Cooper, and M. Raadt. Trends in introductory programming courses in Australian universities: languages, environments and pedagogy. In *Proc. 14th Australasian Computing Education Conference (ACE'12)*, pages 33–42, 2012.

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

[9] Office for National Statistics. UK Population Estimates., ONS, 2016.

[10] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A Survey of Literature on the Teaching of Introductory Programming. *ACM SIGCSE Bulletin*, 39(4):204–223, 2007.

[11] M. Raadt, R. Watson, and M. Toleman. Trends in introductory programming courses in Australian universities: languages, environments and pedagogy. In *Proc. 6th Australasian Computing Education Conference (ACE'04)*, pages 277–282, 2004.

[12] Royal Society. Shutdown or restart? The way forward for computing in UK schools, January 2012.

[13] N. Shadbolt. *Computer science degree accreditation and graduate employability: Shadbolt review.* Department for Business, Innovation & Skills, UK Government, May 2016.