An Analysis of Introductory University Programming Courses in the UK

ABSTRACT

This paper reports the results of a survey of 80 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) in Australia and New Zealand. We report on student numbers, programming paradigm, programming languages and environment/tools used, as well as the reasons for choice of such.

The results in this first UK survey indicate a trend towards...

...especially in the context of substantial computer science curriculum reform in UK schools, as well as increasingly scrutiny of teaching excellence and graduate employability for UK universities.

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 – appear to be legion, along with 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 Australia and New Zealand there have been longitudinal data collections [11, 7, 6] surveying the teaching of introductory programming courses in universities. Surprisingly, 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 (with an overall population of 64.5 million: England: 54.3 million, Scotland: 5.3 million, Wales: 3.1 million and Northern Ireland: 1.8 million) historically ruled by one parliament, but now with devolved assemblies responsible for four separate education systems. In the context of increasing 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 school curriculum reform has had uncertain (and emerging) impact on the delivery of their undergraduate programmes, with the diversity of educational background of applicants likely to increase before it narrows: it is certainly possible now 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 few years, there has been increasing 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 through government-supported loans), as well as the perceived value of professional body accreditation and graduate employability. In February 2015, the UK Department of Business, Innovation & Skills initiated independent reviews of STEM degree ac-

SIGCSE 2017 Seattle, Washington, USA

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

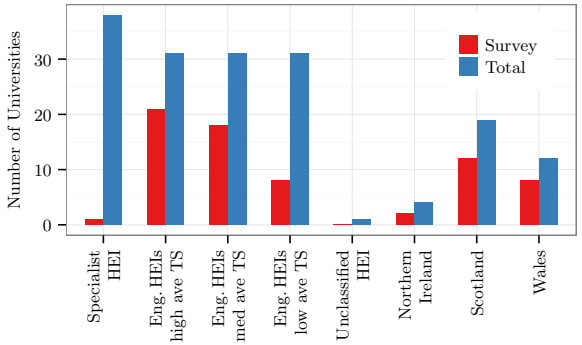

Figure 1: The number of responding universities per Nation/ Tariff Group.

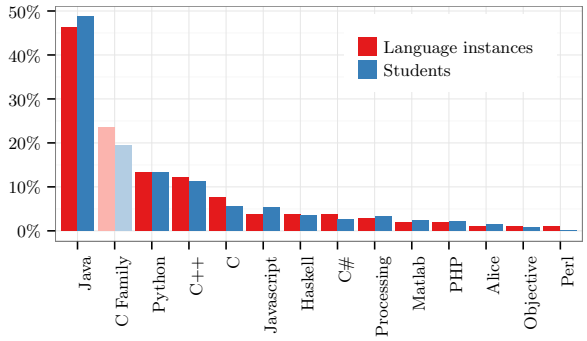

Figure 2: Language popularity by percentage of courses and students (excl. OU).

creditation and graduate employability[1], with a specific focus – the Shadbolt review [13] – on computer science degree accreditation and graduate employability, reporting back in May 2016. Alongside a number of recommendations to address the relatively high unemployment rates of computer sciences graduates, particular on the quality of data, course types, gender and demographics, 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.

Thus, in this emerging environment of policy, curricula, pedagogy and the evolving demands of high-quality learning & teaching for computer science degree programmes, we present the findings from the first national scale survey of introductory programming languages at UK universities. Through this survey, we identify and analyse trends in student numbers, programming paradigm, programming languages and environment/tools used, as well as the reasons for choice of such are reported.

2. METHODOLOGY

2.1 Recruitment of Participants

To recruit for the survey, a general call for participants was sent out to the Council of Professors and Heads of Computing (CPHC) membership; CPHC is the representative body for university computer science departments in the UK, with nearly 800 members at over 100 institutions[2]. The survey was hosted online and was available from mid-May until the end of June 2016; the invitation asked for the survey to be passed on to the most appropriate person for that institution to complete it. Due to the recruitment method, there were a number of duplicate responses from certain 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 2014 Aus/NZ survey. Where possible, questions were left unchanged, although a small minority were edited to reflect the UK target audience. As defined in the 2014 Aus/NZ survey [6], 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 their perceived difficulty and usefulness. Then, 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 the 2014 Aus/NZ survey [6], participants were asked to rank the importance of the given reasons for choosing a programming language, environment or tool. Due to technical limitations in the online survey tool used, it was not possible to do so in this survey, so 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

[1]https://www.gov.uk/government/collections/graduate-employment-and-accreditation-in-stem-independent-reviews

[2]https://cphc.ac.uk/who-we-are/

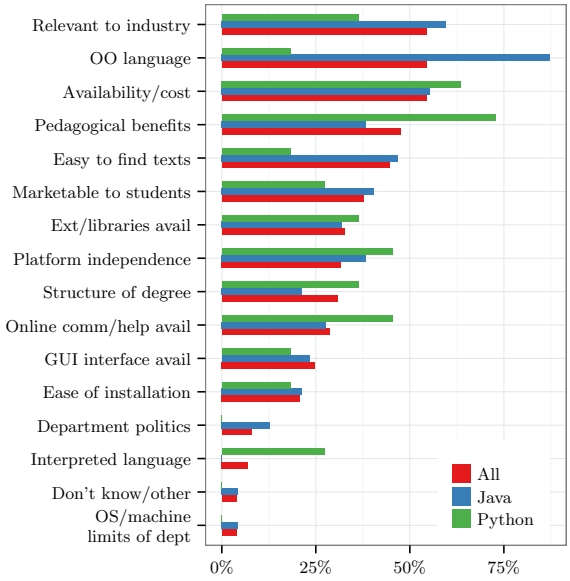

Figure 3: Reasons given for choosing a programming language by percentage for: all languages; Java; and Python.

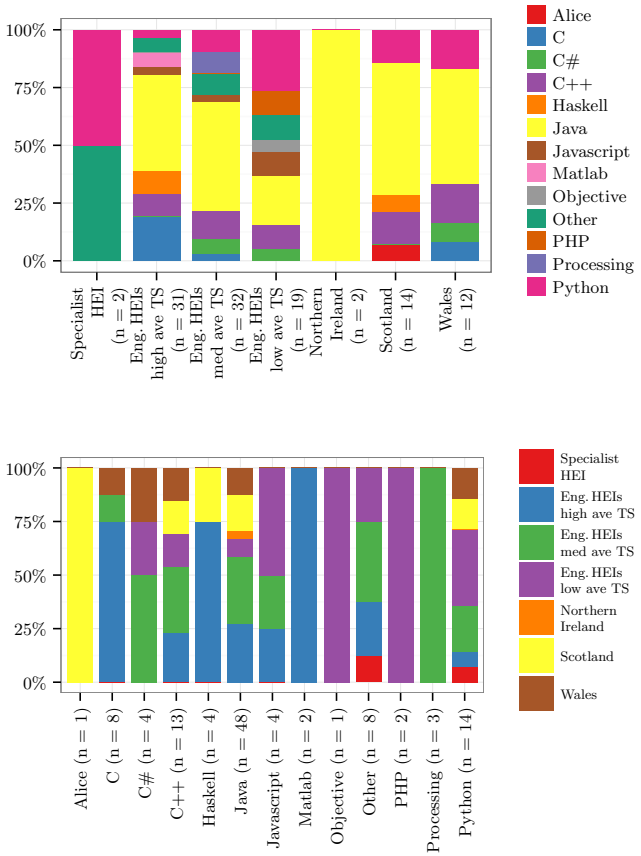

Figure 4: The breakdown of programming languages by Nation and Tariff Groups.

Ireland, we had responses from the two universities, but not the university colleges, which are historically teacher-training colleges.

3.2 Languages

One of the mandatory questions in the survey, and a major point of interest related to the programming languages in use in introductory programming courses. Participants were asked to select languages from a list of 22 programming languages and also had the option to choose "Other" and specify a language not included in the list. The majority of courses surveyed (59 out of 80) use only one programming language, with 17 using two (and only three and one institutions using three and four languages respectively). From the 80 courses, the total number of *language instances* is 106, as some courses use more than one language to teach introductory programming.

Of the 22 languages provided, 13 were selected at least once. The relative popularity of languages is shown in Figure 2, where the prevalence is given by the percentage of a language over all language instances (106 total), and weighted by student numbers (16662 total). With regard to the time a language is used, of 93 language instances, the majority (65%) are used for the whole of the introductory programming course, 14.0% of language instances are used in the first part of a course and 21.5% of language instances are used after another programming language.

The relative popularity of languages is the immediate major difference with [6]. Their survey showed a dead heat (27.3% of language instances) 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 metric, being used in over half the courses (61.3%) and just under half of all language instances (46.2%), while the runner-up, Python, is in use in 17.5% of courses and makes up 13.2% of language instances. The C family (C, C++ and C#) together is in use in 31.3% of introductory programming courses, and scores 23.6% of language instances 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.

For each language selected, participants were asked to give the reasons for choosing that language for the introductory programming course. Figure 3 shows the frequency of these reasons for all languages grouped together and for Java and Python individually. When the reasons given are combined for all languages, three reasons tie for first place: "relevance to industry"; "object-oriented language"; and "availability and cost to students", all chosen by 54.5% of participants who answered this question.

Looked at individually, the most popular reason given for choosing Java is "object-oriented language" at 87.2%, while Python scores highest on "pedagogical benefits", at 72.7%. This may explain the popularity of Java: Java scores higher on "relevance to industry" and, perhaps somewhat surprisingly, much higher on "object-oriented language" than Python, which scores only 18.2%.

Figure 4 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.

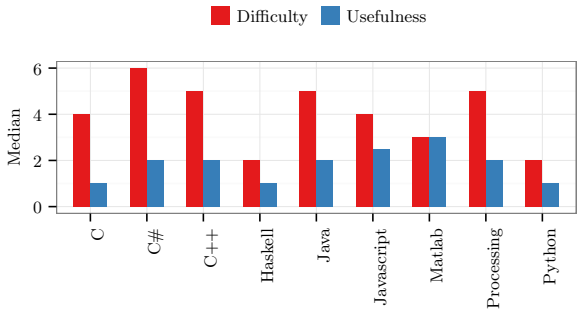

Figure 5: The median of the perceived difficulty and [pedagogic] usefulness of language, where 1 is 'extremely easy/useful' and 7 is 'extremely difficult/useless'.

Table 1: The main paradigm in use in the first programming course.

Paradigm	OO	Procedural	Functional	No Answer
Courses	40	27	7	6

3.3 Paradigm Taught

Instructors were asked what paradigm was being taught in their introductory programming course, regardless of what is traditionally thought to apply to the language(s) in use. This question, understandably, caused some dissatisfaction in the comments section, with many participants noting that more than paradigm is taught in their course. Although this was to be expected, we wanted to be able to directly compare our results to [6], and so did not alter the question.

The results of this question are given in Table 1. We can see that the most popular paradigm is object-oriented, followed by procedural and functional (logical was also offered as a choice but was not selected).

The results of the previous question were used to analyse the prevalence of paradigms across nations and tariff score groups. This analysis is displayed in Figure 6. Caution must be applied when interpreting these results, as participants could only choose one paradigm, even though more may be in use.

[@James: Need to add some commentary here.]

In the same way as above, the languages chosen were analysed with regard to the main paradigm in use. The results are given in Figure 7. Again, caution must be applied, as for a given course, only one paradigm is chosen, even though more than one language and/or paradigm may be in use.

[@James: Need to add some commentary here.]

3.4 Instructor Experience

Participants were asked: "How many years have you been involved in teaching of introductory programming?". The results, shown in Table 2, indicate that of the survey participants, the average was between 10 - 20 years.

Table 2: The number of years the instructor has been teaching introductory programming.

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

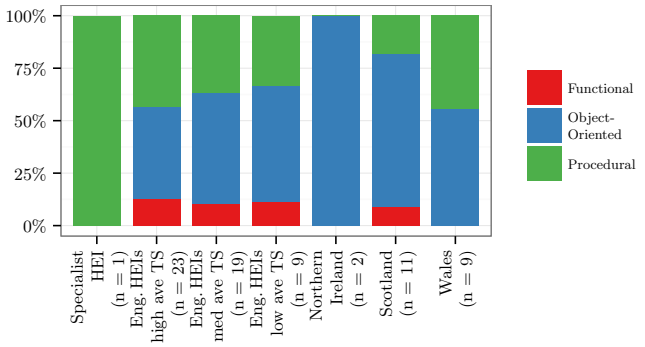

Figure 6: The breakdown of the main paradigm in use for every Tariff Group.

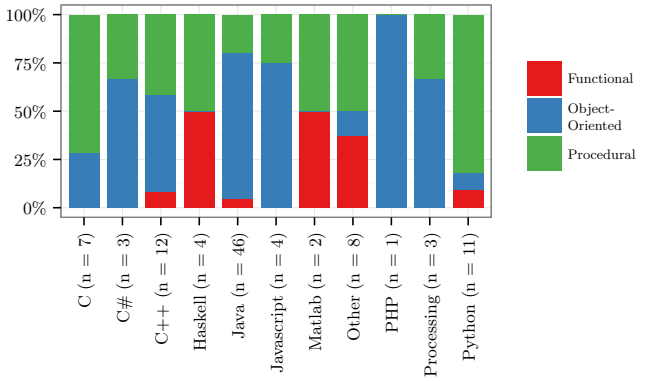

Figure 7: The breakdown of the main paradigm in use for each programming language.

3.5 IDEs and Tools

Participants in the survey were asked if they encouraged students in the first programming course to use environments and/or tools beyond simple text editors and command line compilers. The majority of participants of this question (74.4% of 78 instructors) responded that they did encourage tools. Of the 58 instructors that did select a tool/IDE, the majority (58.6%) use only one, with 25.8% and 10.3% using two and three respectively; very few (5.2%) used four or more, with one respondent using eight.

The survey asked participants to select the tools and IDEs in use in their introductory programming course out of a list of 24, which the option to specify "Other". Of the 24 provided, 12 were chosen at least once. The relative popularity of IDEs and tools is shown in Figure 8. The most popular tool/IDE in the survey was Eclipse, reported in 37.5% of courses and scoring 25.0% of tool/IDE instances, and 26.8% when weighted by students. Following this is "no tool/IDE", which accounts for 27.5% of courses. The second most popular tool/IDE is BlueJ, which was reported in 22.5% of courses and scored 15.0% of tool/IDE instances, and 15.5% when weighted by students. Participants were also asked why each tool/IDE was chosen for their course, and asked to select from a list of reasons. The results of this are give in Figure 9, for all tools and IDEs grouped together, and for the two most popular choices, Eclipse and BlueJ.

3.5.1 Reuse of tool/IDE

Instructors were also asked whether the tool/IDE was used for an initial part of the first programming course or

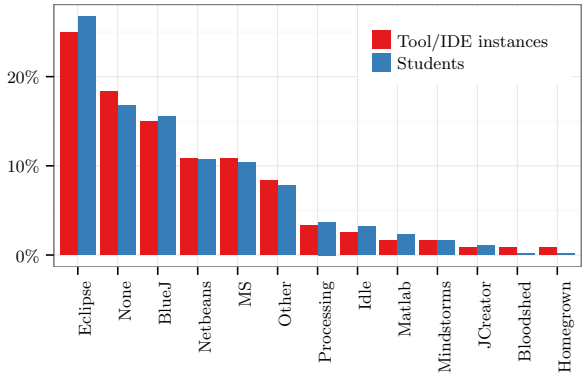

Figure 8: Tool or environment popularity by percentage of courses and students.

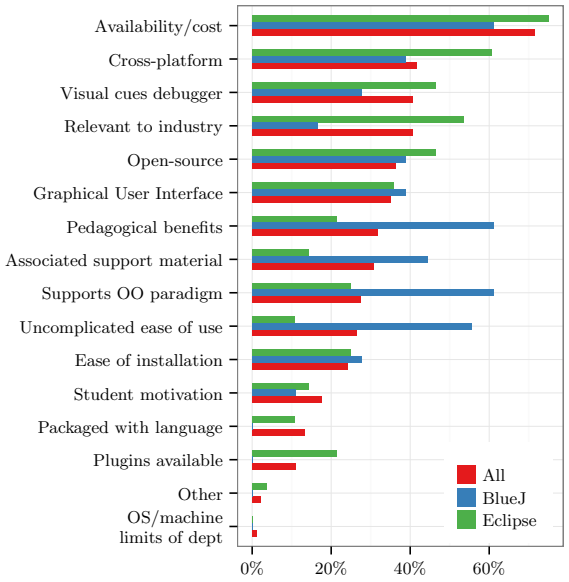

Figure 9: Reasons given for choosing a tool or environment by percentage for: all tools and environments; BlueJ; and Eclipse.

throughout the whole of the course; and whether it was used in any other course in the degree (Figure 10).

3.5.2 *Difficulty of tool/IDE*

In addition to this, instructors were asked to rate how difficult *they* found the tool/IDE on a Likert scale from extremely easy (1) to extremely difficult (7), and also how difficult they believed *the students* found the tool/IDE, shown in Figure 11.

We note that, while Eclipse is the most popular tool by some way, it is also deemed to be most difficult. This, apparently perverse, practice might be explained by the extent of re-use of Eclipse in other courses.

3.6 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.

We asked participants: "what steps do you take to try to determine whether students have received unauthorised assistance on assignments?". The details are in the full paper, but range from "notice unlikely similarities" (59.1% of the 66 instructors who responded to this question) to "interview some students/groups at random", selected by only 5 instructors.

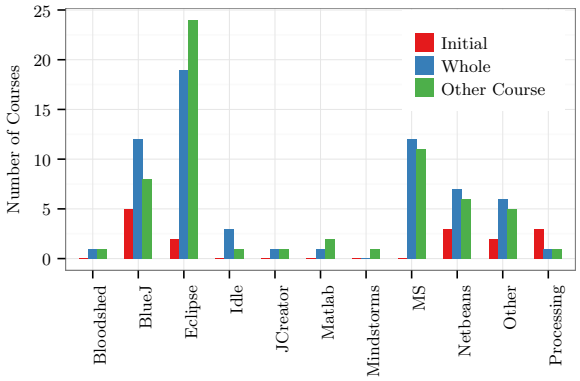

Figure 10: For each tool or environment, whether it is used: for an initial part of the first programming course; throughout the whole of the first programming course; in any other course in the degree.

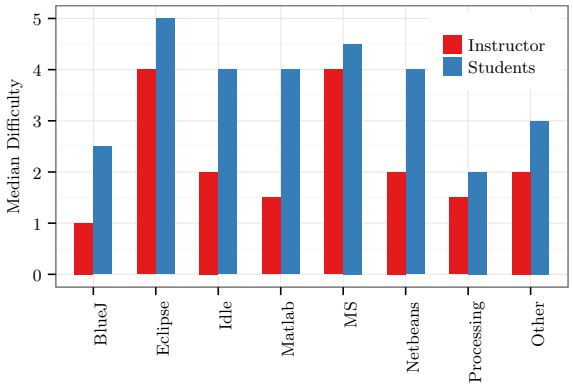

Figure 11: The median difficulty rating of tool for the instructor and students to use, where 1 is 'extremely easy' and 7 is 'extremely difficult'.

3.7 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". Results of this question are shown in Figure 12.

4. GENERAL DISCUSSION

4.1 The U.K. context

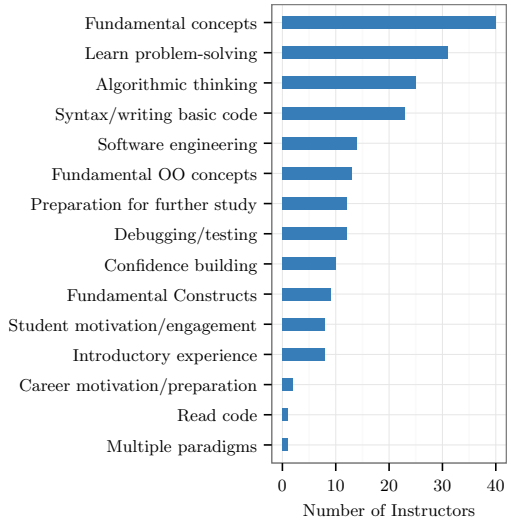

Figure 12: Aims of the introductory course.

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 first, but we also notice that our difficulty/utility data (Figure 5) 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, "None" is second, "Other" is sixth and Idle eighth.

5. ACKNOWLEDGEMENTS

The authors would like to thank the participants for their engagement with the survey, as well as the authors of [8] for providing us with their survey and permission to use it. We are grateful to the GW4 Alliance (Universities of Bath, Bristol, Cardiff and Exeter) for funding the survey.

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. Technical report, 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.