Language Choice in Introductory Programming Courses at Australasian and UK Universities

Simon
University of Newcastle
Australia
simon@newcastle.edu.au

Raina Mason
Southern Cross University
Australia
raina.mason@scu.edu.au

Tom Crick
Cardiff Metropolitan University
United Kingdom
tcrick@cardiffmet.ac.uk

James H. Davenport
University of Bath
United Kingdom
j.h.davenport@bath.ac.uk

Ellen Murphy
University of Bath
United Kingdom
e.murphy@bath.ac.uk

ABSTRACT

Parallel surveys of introductory programming courses were conducted in Australasia and the UK, with a view to examining the programming languages being used, the preferred integrated development environments (if any), and the reasons for these choices, alongside a number of other key aspects of these courses. This paper summarises some of the similarities and differences between the findings of the two surveys.

In the UK, Java is clearly the dominant programming language in introductory programming courses, with Eclipse as the dominant environment. Java was also the dominant language in Australasia six years ago, but now shares the lead with Python; we speculate on the reasons for this. Other differences between the two surveys are equally interesting. Overall, however, there appears to be a reasonable similarity in the way these undergraduate courses are conducted in the UK and in Australasia. While the degree structures differ markedly between and within these regions – a possible explanation for some of the differences – some of the similarities are noteworthy and have the potential to provide insight into approaches in other regions and countries.

CCS CONCEPTS

• **Social and professional topics → Computing education**;

KEYWORDS

Introductory programming, Programming pedagogy, Programming environments, CS1, Computing curricula, Computing education

ACM Reference format:
Simon, Raina Mason, Tom Crick, James H. Davenport, and Ellen Murphy. 2018. Language Choice in Introductory Programming Courses at Australasian and UK Universities. In *Proceedings of The 49th ACM Technical Symposium on Computer Science Education, Baltimore, MD, USA, Feb. 21–24, 2018 (SIGCSE '18),* 6 pages.
https://doi.org/10.1145/3159450.3159547

SIGCSE '18, Feb. 21–24, 2018, Baltimore, MD, USA

ACM ISBN 978-1-4503-5103-4/18/02…$15.00
https://doi.org/10.1145/3159450.3159547

1 INTRODUCTION

The choice of programming language for introductory programming courses, at various levels of school and higher education, has been the subject of discussion for decades and is unlikely to ever be fully resolved. How can we best introduce the 'art' or 'discipline' of computer programming via key programming principles, constructs, syntax, and semantics? This question is often discussed informally, in staff rooms, at conferences, and in online forums such as the mailing list of the ACM Special Interest Group on Computer Science Education, but it is also in evidence in peer-reviewed academic literature [2, 8, 10, 11]. A search of the ACM Digital Library identifies a number of papers of the form "X as a first programming language" going as far back as the 1970s. The issues surrounding the choice of a first language are numerous, related as they are to the question of what educators aim to achieve from teaching programming [9, 20], especially as part of the first year of an undergraduate computing degree program.

Nevertheless, it appears that the wealth of research on the teaching of programming does not easily transfer into improving classroom practice [18]. There is related research in other disciplines, such as education and cognitive science, but disciplinary differences often make it difficult to transfer the findings to computing education. To date, computing instructors have generally not had access to comprehensive surveys of research and practice in this area [16, 18].

To address some of these issues, a long-term study has been conducted in Australasia (Australia and New Zealand) over the past 15 years. In 2001 and 2003, de Raadt and colleagues conducted censuses of Australian and New Zealand universities to examine trends in the programming languages and environments used in introductory programming courses [5–7]. In 2010 and 2013, Mason and Cooper conducted similar surveys online, providing further data on the programming languages being used, along with insight into pedagogies [12–14]. A related national-scale survey conducted in the USA in 2011 provided insight into the state of computing education in that country [4].

The most recent iteration of the Australasian survey was conducted in 2016 [15], along with an almost identical study in the

UK [17]. This paper presents a comparative analysis of these two studies, identifying similarities and differences between the two cohorts. The work is further contextualised in the UK by substantial reform of computing curricula and qualifications in compulsory school education [1], as well as increasing scrutiny of teaching quality and graduate employability in UK universities [3, 19, 21].

1.1 Research questions

While the two surveys were conducted independently, they share a number of research questions:

- What are the aims of the first programming course?
- What programming languages are used in introductory programming courses, and what reasons are given for the choice?
- What programming environments are used with the selected programming languages and what reasons are given for the choice?
- What resources or practices are employed in the courses to enhance the students' learning experience?

1.2 Degree structures in the participating nations

The different nations covered by the two surveys have between them at least three distinct degree structures. In the UK, the degrees in Scotland and Northern Ireland are somewhat like those in northern America: two years of broad education followed by two years in a focused major. Introductory programming in such a degree might well be taught to students who are not yet sure whether they will major in computing. Still in the UK, in England and Wales the three-year single-honours degree is focused from the outset, with introductory programming taught to students who expect to be studying little other than computing for three years. Degrees in Australia and New Zealand also run for three years, but are somewhat less focused: the computing content will typically be half or two thirds of the degree, with the remainder being more or less related subjects such as business, science, communication, or indeed whatever the student chooses. It is quite possible that these differing structures exert different influences on the choice of introductory programming language.

2 METHODS AND DEMOGRAPHICS

2.1 Participants

Both surveys were hosted online, with appropriate institutional ethics approval, during the first part of 2016. Several strategies were used to invite the people in charge of introductory programming courses to participate in the surveys.

In Australia and New Zealand, email invitations were sent to past participants, a relevant mailing list, and academics identified from their University's website. In the UK, contact was made through the members of the Council of Professors and Heads of Computing (CPHC), the representative body for computing departments in UK universities.

These strategies resulted in valid responses for 48 introductory programming courses in Australasia and 80 courses in the UK. Responses came from 35 institutions (57%) in Australasia and 70 institutions (47%) in the UK. Both regions include universities that do not offer computing, so the effective response rates are a little higher than presented here.

A number of responses in each survey were excluded from the analysis because they were duplicate responses for their course or because they were substantially short of being complete.

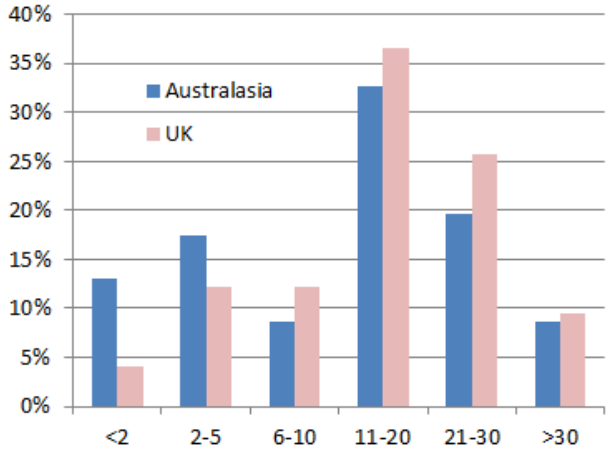

Figure 1: Years of experience of respondents

2.2 Survey questions

A core set of questions was asked on all prior Australasian surveys, with additional questions changing in each iteration. The core questions, which were included in both surveys reported here, include those about the programming language/s used in the course, the programming environment/s used in the course, and why those languages and environments were chosen. Participants were also asked about the aims of their course, instructor experience, and the learning resources that they provide for their students.

2.3 Demographics

How long have the respondents been teaching? There are at least two distinct strategies for the teaching of introductory programming: to entrust the course to senior academics who are known to be excellent teachers, in the hope of giving the students the smoothest possible introduction to the topic; or to consider the course as a burden and assign it to junior academics. Figure 1 shows the teaching experience of the Australasian and UK respondents.

The spans are not of equal lengths, so the shape of the curve is not in itself meaningful; however, it is interesting to compare the numbers in the five-year span 6-10 years with those in the shorter spans <2 years and 3-5 years. At least in Australasia, the low number in that span might reflect a period of low hiring between 2005-2010, when student numbers were low.

The UK proportions are lower than the Australasian ones in the first two spans and higher in the last three. This might suggest that UK departments are more inclined to assign the introductory programming course to senior academics, and Australasian departments to those with less experience.

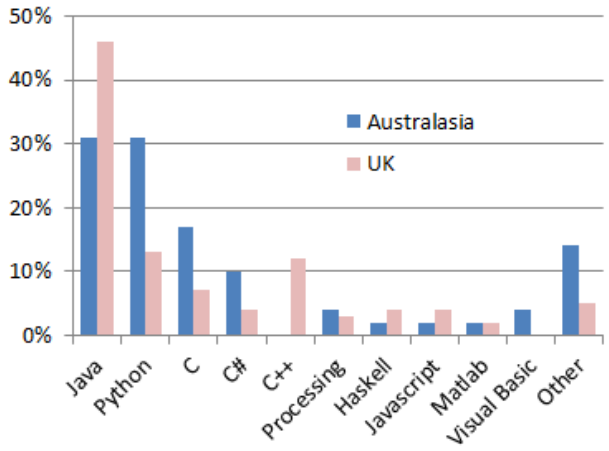

Figure 2: Language use by course

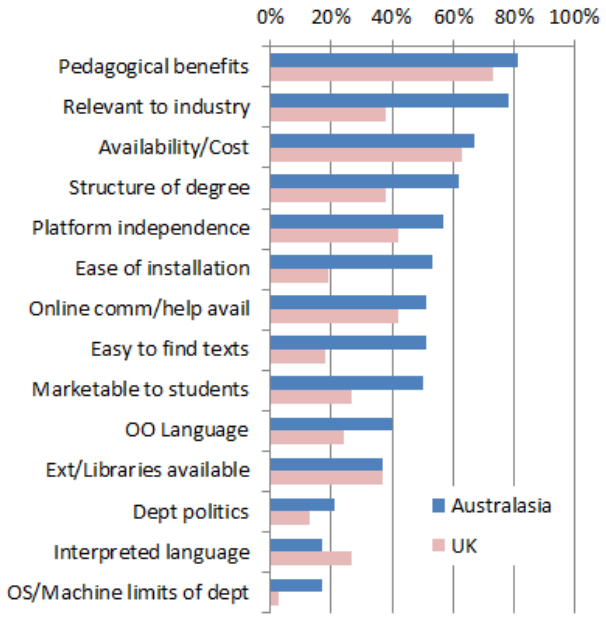

Figure 3: Prevalence of reasons for choice of language

3 COURSE AIMS

In a free-text question, respondents were asked to indicate their three principal aims for their introductory programming courses. The answers were classified by the surveyors into themes, resulting altogether in a list of more than 20 themes. A number of themes were clearly dominant across both surveys:

- Fundamentals of programming, programming concepts
- Problem solving
- Algorithmic/computational thinking
- Programming language syntax and basic code
- Student enjoyment/motivation

It is worth noting that the specifics of particular programming languages were seldom rated as highly as more generic concepts such as problem solving, algorithmic thinking, and programming concepts.

4 PROGRAMMING LANGUAGES

The initial and continuing impetus for this project is to determine which programming languages are being used in introductory programming courses.

4.1 Choice of language

Figure 2 shows the percentages of courses in which each language is used in the UK and in Australasia. We have also produced plots of the percentage of students to whom each language was taught, but these plots are broadly similar between the two surveys, so in this and subsequent analysis we shall base all of our plots on the proportion of courses. The 'Other' category comprises 11 languages, none of which was used in more than 2% of the courses. These are Bash, Delphi/Pascal, machine code, PHP, R, Snap, Swift, VBA, Alice, Objective C, and Perl.

In the UK, Java is the clear leader, whereas in Australasia Java and Python are level. However, the proportion of Australasian students taught in Python is substantially higher than the proportion of students taught in Java, indicating that Python is used more than Java in larger courses. A 2011 study in the US [4] found that Python slightly outranks Java in CS0 courses, courses offered to students to help them decide whether to major in CS; whereas Java holds sway in CS1 and beyond. This suggests that Python might be perceived as easier for novices to grasp. Java was dominant in the earlier papers of de Raadt and colleagues [5–7], but that long-term study has seen Python gradually draw level with Java. This might be indicative of efforts to introduce programming to students who are less experienced and/or less capable.

The results suggest that the status of Java in the UK in 2016 is about where it was in Australasia in 2010. It will be interesting to see whether the UK is on a similar trend, but some five years behind; or whether the current differences are attributable to the differences in degree structures or student cohorts. The high incidence of C in the Australasian survey can be attributed in part to introductory programming courses for engineering students, as such courses are often offered in either C or Matlab. Of course engineering students in the UK are also taught programming, but perhaps not by the schools covered by the Council of Professors and Heads of Computing, so those courses might not have been captured by the survey.

4.2 Reasons for language choice

Knowing what languages are used in introductory courses is only part of the story; an exploration of the choice of languages would not be complete without asking why a particular language has been chosen. Indeed, a recent US paper observed that *"Evidence that...these [language] choices make sense, as a whole or in terms of particular features, cannot be established from the literature"* [22].

To explore this question in the Australasian survey, respondents were given a list of reasons and asked to rate the importance of each on a five-point scale. The UK survey, in contrast, asked respondents to select all of the reasons that applied to their choice of language.

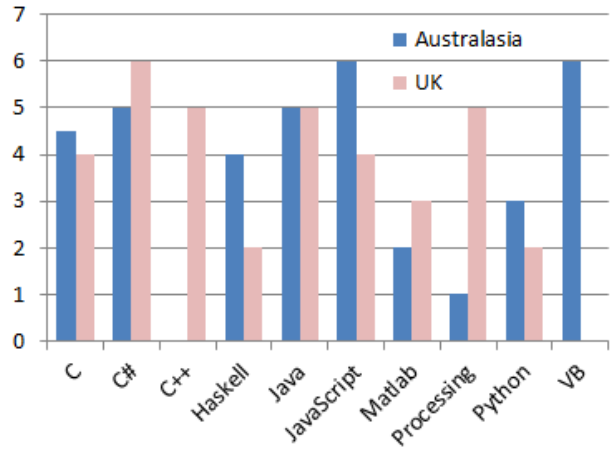

Figure 4: Median perceived difficulty of the language for novices: 1 (lowest) to 7 (highest)

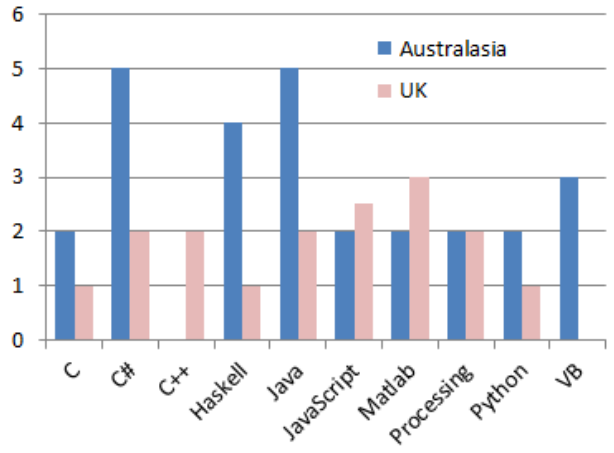

Figure 5: Median perceived usefulness of the language for teaching programming concepts: 1 (lowest) to 7 (highest)

Figure 3 combines these disparate approaches as follows: from the Australasian survey, it shows the proportion of respondents who rated each reason as extremely, very, or moderately important, the top three choices on the scale; from the UK survey it shows the proportion of respondents who selected the reason at all.

Pedagogical benefits ranks highest on both measures. Industry relevance is also seen as important, but surprisingly less so in the UK than in Australasia, especially given the significant policy focus in the UK on teaching excellence and thus, indirectly, on graduate employability. Availability/cost of the language is also clearly important to respondents of both surveys.

4.3 Perceived difficulty and usefulness of languages

Respondents were asked how difficult they felt the chosen language was for novice programmers, on a scale from 1 (least difficult) to 7. Figure 4 shows the median response for each of the major language choices in each survey.

There is a clear feeling from both surveys that Python is one of the easier languages for novice programmers. This finding ties in well with the earlier suggestion that the use of Python in introductory programming courses might be partly because it is perceived as easier to learn.

Haskell and Processing are the languages with greatest disparity between the two surveys. However, as shown in Figure 2, these are languages with very low usage, so the median can be strongly influenced by extreme judgements.

In a related question, respondents were asked how useful they felt the language was for teaching programming concepts. At least on the basis of the median, this question elicited some sharply different responses in the two surveys (Figure 5). The essential impression is that few UK respondents see any of their chosen languages as useful for learning programming concepts, whereas in Australasia three languages have medians on or above the midpoint of the scale.

5 PROGRAMMING LANGUAGE ENVIRONMENTS

Integrated development environments (IDEs), which incorporate colour-coded context-sensitive editors, compiling and linking tools, debugging facilities, and perhaps wizards for GUI creation, can be seen as both assistive and distractive for novices learning to program. It is clearly a distraction to have to learn how to use an IDE software package while simultaneously learning how to program. On the other hand, an IDE can help to enthuse learners by making it easy for them to start with classes and objects (as with BlueJ) or with GUI-based applications rather than command-line interfaces (as with Eclipse or Visual Studio). While a programming language is all but essential in an introductory programming course, some instructors prefer not to use an IDE because of its potential for distraction or obfuscation of processes.

Survey respondents were asked whether they encourage their students to use a particular IDE. Figure 6 shows the major IDEs that were nominated, with the proportion of courses using each IDE from each survey. While some of these IDEs can be used with multiple programming languages, others are strongly linked to particular languages, such as BlueJ with Java and Visual Studio with C#.

As with programming languages, respondents were asked why they chose the IDEs that they did. Also as with programming languages (Figure 3), each survey collected the responses in a different way, so the values plotted in Figure 7 are not entirely comparable. Reasons in the UK survey appear to be dominated by availability and cost to students, ability to work across platforms, and industry relevance, while those in the Australasian survey focus more on simplicity and ease of use, pedagogical benefits, ease of installation and use, and availability and cost to students. As with programming languages, it is possible that the Australasian survey respondents are more concerned to facilitate the learning process for struggling students.

It is possible that the use of installed IDEs will soon diminish; a number of responses indicate the use of cloud-based IDEs such

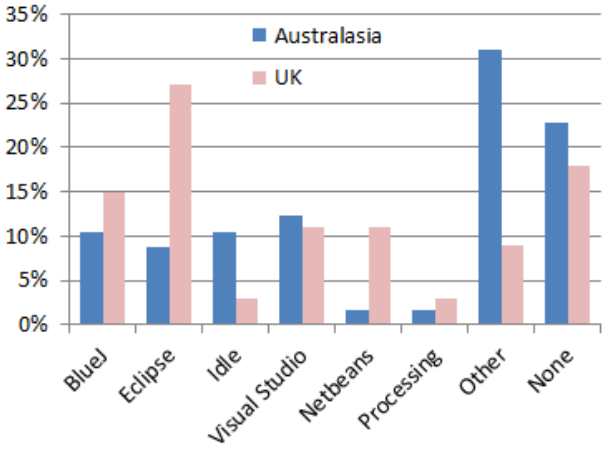

Figure 6: Proportions of courses that encourage the use of particular IDEs

as Brackets, Clara's World, and Grok Learning, which require no infrastructural overhead as they run in a browser window.

6 RESOURCES TO ASSIST STUDENTS

Different courses are known to offer different support and resources to help students in their learning. In living memory, lecturers wrote lecture material on blackboards and students copied it to their own notes. The blackboard was replaced by the overhead projector, which was replaced in turn by computer-based slides. Students were reluctant to copy from the slides to their notes, so it soon became mandatory in some institutions to provide students with copies of the slides. Many other resources have since become available, and survey respondents were asked to indicate what resources they

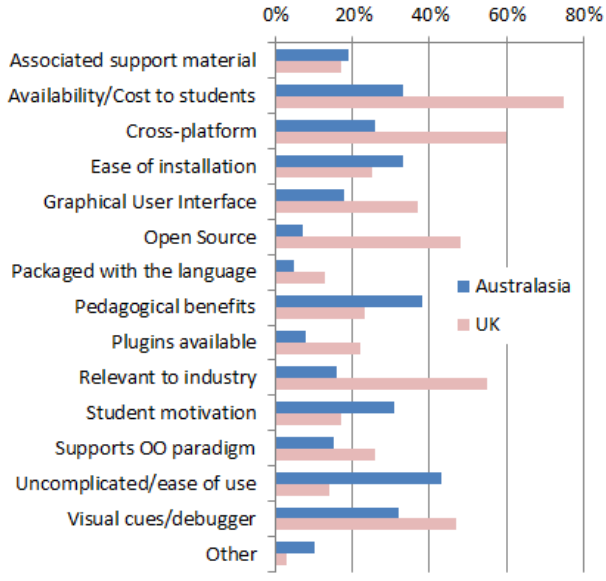

Figure 7: Prevalence of reasons for choice of IDE

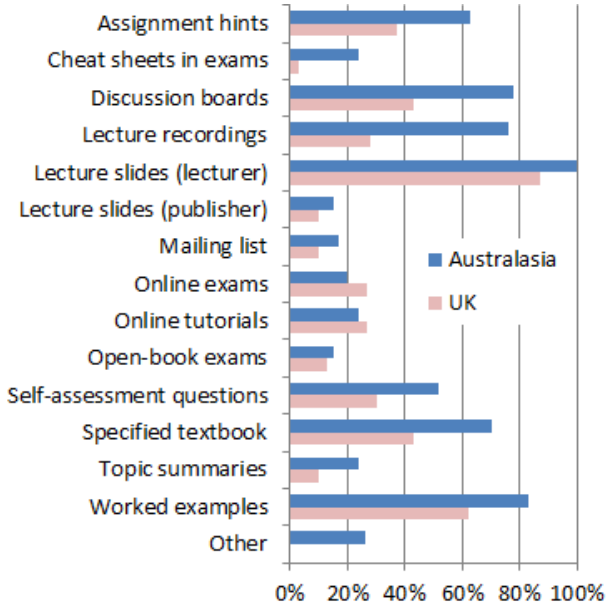

Figure 8: Proportions of courses providing resources/ practices that might assist students

provided to assist students with their learning. The responses are shown in Figure 8.

It seems fairly clear from Figure 8 that the Australasian respondents are taking more effort to offer resources that might assist the students. We might speculate that this shows the Australasian respondents to be more altruistic. However, their reasons are more likely to be pragmatic: perhaps their intakes include more students who are less capable in computing, and they therefore have to try harder in order to maintain a reasonable retention rate.

7 ISSUES AND THREATS TO VALIDITY

Both the Australasian and UK surveys were voluntary, and suffer from the usual issues of self-selection. The UK survey was conducted via email to the Council of Professors and Heads of Computing, which often has multiple representatives per institution. In seven cases we had duplicate responses for the same course, which sometimes differed slightly in numbers and more substantially in the qualitative questions (Figures 3-5). The UK team discussed this with the respondents, and accepted the responses of the person the respondents deemed to be most closely involved in the decisions.

The Australasian surveys are also attempting historical tracking. However, only 14 of the 48 respondent courses were definitively identified as taking part in both the 2013 and 2016 surveys. It is quite likely that a number of others were also repeating, but had, for example, changed their course codes – or even the name of their university – in that time. However, we know that there are some new respondents representing some new courses, and we are fairly confident that some courses represented in the 2013 survey were not represented in 2016. Hence changes over time (few of which are presented in this paper) might represent changes in sample means rather than variations in practice.

Ultimately we must consider whether we are comparing like with like, and clearly in some respects we are not. Even within the UK, most of the courses reported from England are those taught in single-honours (or possibly joint honours) computer science degrees, whereas those from Scotland are typically taught to students who have not yet decided whether to specialise in computer science. In this respect, Wales is more similar to England, and Northern Ireland to Scotland. In another paper [17] we have broken down the figures by nation, and by university groupings in England, and have found some differences correlated with this difference in mission. For example, C is never reported as an introductory language in Scotland or Northern Ireland. While the degree structures in Australasia lie between those in England and Scotland, the situation there is compounded by the fact that some of the respondents taught courses targeted to completely different cohorts, such as engineering students, and others taught mixed cohorts of computing, information systems, engineering, and even business students. The variations within nations are probably at least as great as those between nations.

8 CONCLUSIONS

The initial conclusions are contradictory: Java is clearly more common in the UK than in Australasia (Figure 2), despite industrial relevance appearing to be a more common reason for choice in Australasia than in the UK (Figure 3), and despite both surveys reporting that Python was significantly less difficult for novices. If one assumes that Java is more industrially relevant than Python, it is hard to reconcile these findings. One possible explanation is that the pressure in the UK to be industrially relevant, with departments being measured according to percentage of students in a graduate-level job, is so great that UK respondents have internalised it and no longer consciously consider it. Another possible explanation is inertia; the UK survey is the first of its kind, but the Australasian surveys over time show a slow shift from Java to Python. It is conceivable that the UK is on a similar journey to Australasia, and that a future UK survey will show a growth in the use of Python.

Programming is increasingly being taught in schools. In the near future, universities can expect that more of their intake, but by no means all of it, will already have learnt some programming. It will be interesting to see what impact that will have on the future development of introductory programming courses at universities.

The most important message from this work is not the snapshot findings for 2016 that we have presented here, but the questions addressed by the surveys. The language wars [22] are unlikely ever to reach a definitive resolution, so computing departments will periodically consider which programming language they should use in their introductory courses. They would do well, in their discussions, to address the questions raised in this paper. Which criterion should drive the choice of language for introductory programming? Industrial relevance, ease of learning, or something else? What criteria should drive the choice of IDE, if indeed an IDE should be used? What resources should be provided to accompany the teaching materials? Which people should be assigned the task of teaching the introductory course? These, and all of the other questions that have been addressed in these pages, should play a clear part whenever these choices are made.

9 ACKNOWLEDGEMENTS

We thank the respondents who engaged with the surveys, and the various groups that helped to promulgate them. The survey in the UK was funded by the GW4 Alliance (Universities of Bath, Bristol, Cardiff and Exeter).

REFERENCES

[1] Neil CC Brown, Sue Sentance, Tom Crick, and Simon Humphreys. 2014. Restart: the resurgence of computer science in UK schools. *ACM TOCE* 14, 2 (2014), 1–22.
[2] Nell B Dale. 2006. Most difficult topics in CS1: results of an online survey of educators. *ACM SIGCSE Bulletin* 38, 2 (2006), 49–53.
[3] James H Davenport, Alan Hayes, Rachid Hourizi, and Tom Crick. 2016. Innovative pedagogical practices in the craft of computing. In *Fourth International Conference on Learning & Teaching in Computing and Engineering*. 115–119.
[4] Stephen Davies, Jennifer A Polack-Wahl, and Karen Anewalt. 2011. A snapshot of current practices in teaching the introductory programming sequence. In *42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)*. 625–630.
[5] Michael de Raadt, Richard Watson, and Mark Toleman. 2002. Language trends in introductory programming courses. In *Informing Science + IT Education Conference*. http://proceedings.informingscience.org/IS2002Proceedings/papers/deRaa136Langu.pdf
[6] Michael de Raadt, Richard Watson, and Mark Toleman. 2003. Language tug-of-war: industry demand and academic choice. In *Fifth Australasian Computing Education Conference (ACE 2003)*. 137–142.
[7] Michael de Raadt, Richard Watson, and Mark Toleman. 2004. Introductory programming: what's happening today and will there be any students to teach tomorrow?. In *Sixth Australasian Computing Education Conference (ACE 2004)*. 277–282.
[8] Sally FIncher. 1999. What are we doing when we teach programming?. In *29th Annual Frontiers in Education Conference*.
[9] Philip Guo. 2014. Python is now the most popular introductory teaching language at top US universities. (2014). Retrieved 30 Aug 2017 from http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities
[10] Diwaker Gupta. 2004. What is a good first programming language? *ACM Crossroads* 10, 4 (2004), 7.
[11] Randy M Kaplan. 2010. Choosing a first programming language. In *ACM Conference on Information Technology Education*. 163–164.
[12] Raina Mason and Graham Cooper. 2012. Why the bottom 10% just can't do it – mental effort measures and implication for introductory programming courses. In *14th Australasian Computing Education Conference (ACE2012)*. 187–196.
[13] Raina Mason and Graham Cooper. 2014. Introductory programming courses in Australia and New Zealand in 2013 – trends and reasons. In *16th Australasian Computing Education Conference (ACE2014)*. 139–147.
[14] Raina Mason, Graham Cooper, and Michael de Raadt. 2012. Trends in introductory programming courses in Australian universities – languages, environments and pedagogy. In *14th Australasian Computing Education Conference (ACE2012)*. 33–42.
[15] Raina Mason and Simon. 2017. Introductory programming courses in Australasia in 2016. In *19th Australasian Computing Education Conference*. 81–89.
[16] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin* 33, 4 (2001), 125–180.
[17] Ellen Murphy, Tom Crick, and James H Davenport. 2017. An analysis of introductory programming courses at UK universities. *The Art, Science, and Engineering of Programming* 1, 2 (2017).
[18] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin* 39, 4 (2007), 204–223.
[19] QAA. 2016. *Subject Benchmark Statement: Computing*. UK Quality Assurance Agency.
[20] Carsten Schulte and Jens Bennedsen. 2006. What do teachers teach in introductory programming?. In *Second International Workshop on Computing Education Research*. 17–28.
[21] Nigel Shadbolt. 2016. *Computer science degree accreditation and graduate employability: Shadbolt review*. Department for Business, Innovation & Skills, UK Government.
[22] Andreas Stefik and Stefan Hanenberg. 2014. The programming language wars: questions and responsibilities for the programming language community. In *Onward! 2014; ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*. 283–299.