Using Lisp-based pseudocode to probe student understanding

Christophe Rhodes
Goldsmiths, University of London
London, United Kingdom

ABSTRACT

We describe our use of Lisp to generate teaching aids for an Algorithms and Data Structures course taught as part of the undergraduate Computer Science curriculum. Specifically, we have made use of the ease of construction of domain-specific languages in Lisp to build an restricted language with programs capable of being pretty-printed as pseudocode, interpreted as abstract instructions, and treated as data in order to produce modified distractor versions. We examine student performance, report on student and educator reflection, and discuss practical aspects of delivering using this teaching tool.

CCS CONCEPTS

• **Applied computing** → **Computer-assisted instruction**; • **Social and professional topics** → *Computational thinking*; • **Theory of computation** → *Program constructs*; *Program reasoning*; • **Software and its engineering** → *Multiparadigm languages*;

KEYWORDS

multiple choice questions, Lisp

ACM Reference Format:
Christophe Rhodes. 2018. Using Lisp-based pseudocode to probe student understanding. In *Proceedings of the 11th European Lisp Symposium (ELS'18).* ACM, New York, NY, USA, 8 pages.

1 INTRODUCTION

In this paper, we discuss the development and use of a large question bank of multiple-choice, short-answer and numerical-answer questions in teaching a course in Algorithms & Data Structures, as a component of degree programmes in Computer Science and in Games Programming in the United Kingdom. We report specifically on the use of automation, using Lisp among other tools, to develop questions with specific distractors and specific feedback corresponding to likely or common student mistakes.

Gamification of learning has been experimented with and studied in detail in recent years, with the increasing availability of platforms and integrations allowing for more and varied gamification techniques to be applied at all stages of a student's education; the benefits of gamification include higher student engagement, with the curriculum material (as the tasks are intended to probe or reinforce the course content) and with the rest of the cohort (through the social elements of game-playing). The approach we describe here can be viewed as an application of gamification techniques; in the categorization of a recent systematic mapping [3], we describe an element of gamification in a blended-learning course delivered in conjunction with a learning management system, with rapid feedback and countdown clocks, specifically in the context of Computer Science education but applicable more widely.

In the remainder of this introductory section, we provide some relevant context for the Algorithms & Data Structures course in which we have implemented this pedagogy: the conventions of Higher Education in the UK, of Computing education in UK Higher Education, and of the particular programmes of study at Goldsmiths. Section 2 covers the development of multiple-choice question banks suitable for our purposes, including the use of Lisp to help to generate the questions, specific feedback, and assure their correctness. Section 3 describes the in-course delivery of quizzes including these questions, presenting quantitative results and qualitative reflections from students and educators, and we conclude in section 4.

1.1 UK Higher Education

In the UK, Tony Blair in 1999 famously gave as an aim that half of all young people should go to University. Since giving that aim in a party conference speech, the UK Higher Education landscape has shifted substantially, with the introduction and raising of tuition fees (from £1k per year, to £3k and then £9k per year), and the removal of quotas and caps in student recruitment, and the situation is indeed that half of people under 30 in the UK have started a programme of Higher Education, compared with approximately one quarter two decades ago.

This rapid growth in student numbers has inevitably led to pressure on resources: campus space, lecture halls, and staff time. Additionally, placing more of the costs of Higher Education on the student, even if through a notional student loan (which operates more like a tax), has led to more consumerist and arguably transactional approaches to education from the students: it is more common to hear from students now that they are paying for content that they will consume than it would have been twenty years ago. In addition, students nowadays are digital natives; they are accustomed to online delivery of content, though perhaps not so much of material requiring substantial engagement; they are used to the affordances provided by online platforms, and indeed are somewhat intolerant of the perceived backwardness of some Learning Management Systems.

One might expect, given that the students are at least notionally responsible for the cost of their own higher education, that students would have made an informed choice about their programme of study and have clarity about their reasons for entering Higher Education. However, this is not always the case [5, 6], and even when it is, those reasons may not align with the educator's reason for teaching in Higher Education; a majority of students enter into Higher Education seeing it as a means to an end, of getting a job that would otherwise be inaccessible, or having a better chance at a particular career – whereas few teachers in Higher Education have the career development of their students as their primary motivation.

ELS'18, April 16–17 2018, Marbella, Spain
2018. ACM ISBN 978-2-9557474-2-1.

Teachers in Higher Education do operate under constraints, sometimes quite severe ones. One such constraint is that the system works in a way that expects it to be unlikely for students to fail courses. Even minimal engagement with the material is expected to yield a passing grade; degrees are further *classified*, with classifications of a "first-class" or an "upper second" being considered of high enough quality to act as an entry qualification for typical graduate trainee schemes or study for advanced degrees, while "lower second" or "third-class" classifications, while indicating that the degree was passed, are seen as being of lesser quality[1]. Conventionally throughout the sector, a mark of 40% is a pass, and a mark of 60% is the boundary between lower- and upper-second degree classifications.

1.2 Computing education

When offering a degree programme in Computer Science or a related discipline, we must be conscious of the fact that we will have at least three constituencies in our student cohort. We may have some students who will go on to further academic study of the discipline itself; however, we would expect those students to be small in number compared with the students who are studying Computer Science as a means to an end (such as a career in Informatics) or who do not have a particular reason for studying Computer Science at all.

In designing our curricula and our teaching methods, we must therefore accommodate multiple different styles of learning and a wide range of current and prior engagement. We will have to teach students who are already accomplished programmers and wish to deepen their theoretical understanding, and students who believe that a University course can teach them to programme so that they can go out and get a job. We must therefore be careful to nurture development of applicable and transferrable ways of thinking, helping the students to develop computational thinking [11] or build mental models or "notional machines" [9] of the systems that they interact with.

Teaching students to programme, and to reason about programmes, is difficult – and assessing whether students have mastered individual elements of the skill [1, 8] potentially has a high cost. We do not claim to have found a panacea, but one aspect which we believe is particularly demotivating is the somewhat binary nature of assessment: it is common to see bimodal distributions of outcomes, or at least high failure rates [7], typically corresponding to a failure by the student to get anything working at all – an experience seen in microcosm by anyone faced with inscrutable compiler or linker error messages. As educators, we should aim to find ways to allow students to receive partial credit for partial solutions, so as to recognize forward progress even if it has not yet led to a fully-functional implementation.

1.3 Algorithms & Data Structures at Goldsmiths

We report in this paper on a course in Algorithms & Data Structures at Goldsmiths. There is a particular issue in the delivery of this course: it is taken as a compulsory part of the programme by students on the BSc in Computer Science (CS) programme and those on the BSc in Games Programming (GP). The CS students are taught programming in Java, while the GP students are taught programming in C++. This creates a particular challenge, in that examples need to be in both or neither programming language in order not to give the perception of unfair or second-class treatment to either cohort. In this course, students are given practical programming work in the form of small automatically-marked lab assignments as well as more open tasks, but theory is presented in a language-neutral pseudocode format.

[1]This is a highly simplified description, as there are also distinctions between the perceived quality of degrees awarded by different institutions, being a combination of reputational teaching quality and expected student attainment at intake.

2 QUESTIONS

One of the components of our delivery of this material is a series of multiple choice quizzes, delivered through the Moodle[2] Learning Management System (LMS). These quizzes are intended to be part measurement instrument – the mark achieved contributes to the final grade in the course – but chiefly a pedagogical tool, to help the students recognize whether they have understood the material sufficiently to identify or generate solutions to problems.

The function of our questions is similar to the *root question* concept described in the Gradiance documentation [10]: we aim to produce questions, or question templates, with the following characteristics:

- a student who has understood the material should find answering the question to be straightforward;
- a student who has not begun mastering the material should have a low probability of being able to guess the correct answer;
- individual or groups of students should find it easier to master the material than to acquire and search through a set of questions with corresponding answers;
- for multiple choice questions, distractor answers corresponding to common misunderstandings or misconceptions should be present.

The reason for the last characteristic, that distractor answers should be present, is to be able to identify individual students, or measure the fraction of students, with a particular misunderstanding, and to give them targetted feedback aimed to improve their understanding. There is no need for distractors in numerical- or short-answer questions, but the questions we produce must still be done with that understanding, in order to be able to give targetted feedback for particular wrong answers. The subsections below give examples of targetted feedback in both short-answer and multiple-choice questions.

2.1 Pseudocode

As described in section 1.3, the class taking this course consists of two separate cohorts. To establish a common language, therefore, an early lecture established the pseudocode conventions to be used throughout the course (essentially a subset of the algpseudocode notation provided by the LaTeX algorithmicx package).

One of the questions (see figure 1) asked participants to compute the final value of a variable after it was incremented within a loop:

[2]https://moodle.org/

What is the return value of this block of code? You may assume that the value of all variables before the start of this block is 0.

```
x ← 4
for -5 ≥ i > -15 do
    x ← x + 1
end for
return x
```

Figure 1: example simple loop question, question 6 of the Pseudocode quiz

What is the return value of this block of code? You may assume that the value of all variables before the start of this block is 0.

```
x ← 8
for 4 ≤ i < 16 do
    x ← x + 1
    break
    x ← x + 1
end for
return x
```

Figure 2: example loop question, question 8 of the Pseudocode quiz

the intent of the question was to make sure that the students could identify the number of times the loop body was executed. As well as the generic feedback given to a student after an attempt, specific feedback was included to be shown to the student when they had made an off-by-one error, reminding them to check the boundaries of the iteration carefully.

A subsequent question in the same quiz used the same question format, but introduced the keywords **break** and **continue**. Again, students were given the generic indication for correct or incorrect answers, but also specific feedback for particular wrong answers given if the student had computed the return value for the wrong keyword, or for no keyword present at all (see figure 2).

The Moodle LMS provides for automatic generation of variants of questions through its Calculated question type, where a template is filled in with randomly-chosen values, and a symbolic expression (supporting standard mathematical operators) is interpreted with each variable from the template bound to the corresponding value. This facility is sufficient for questions based on simple calculations, but has disadvantages for our purposes: the interface for writing calculated questions requires a connection to the Moodle server, and cannot be done off-line; it requires hand-editing each question, which is error-prone; and generating non-numerical variants automatically (*e.g.* choosing between **break** and **continue**) is not possible.

We therefore took a different approach. We defined a sexp-based mini-language to represent the constructs supported in our pseudocode, and implemented a pretty-printer and an interpreter in Emacs Lisp. The definition and implementation were extended as necessary from an initial set of six operators (the basic mathematical operators, variable setting, and **return**) to encompass conditionals, loops, function definition, and various elementary data structures and operations on them (such as lists and vectors).

We could then generate valid forms in our mini-language, somewhat reminiscent of generation of random forms for compiler testing [4]; see listing 1, which is the code to generate random examples of the block presented in figure 2. These sexp-based forms are then pretty-printed to Moodle's GIFT input format[3], and surrounded with question markup to produce questions such as the ones presented in figures 1 and 2.

```
(defun make-break-continue-for-form ()
  (let* ((ascend (flip))
         (comps (if ascend '(< ≤) '(> ≥)))
         (lc (elt comps (if (flip) 0 1)))
         (uc (elt comps (if (flip) 0 1)))
         (start (* (maybe-sign) (random 10)))
         (diff (+ (random 10) (random 10) 1))
         (end (if ascend (+ start diff) (- start diff))))
    `(progn
       ,(make-form 'setq 'x)
       (for (,start ,lc i ,uc ,end)
            (progn
              (incf x 1)
              ,(if (flip) `(break) `(continue))
              (incf x 1)))
       (return x))))
```

Listing 1: Emacs Lisp code to generate a loop in our mini-language containing a break or continue within a for loop, with reasonable start- and end-points.

Not only this, but if we could express a likely mistake that a student might make in code (such as the off-by-one errors or the confusion between **break** and **continue**), we could generate the corresponding form, interpret it, and write specific feedback based on that specific mistake, while checking that it did not accidentally replicate the correct answer. Code to pretty-print, add the question, answer and feedback is demonstrated in listing 2.

This approach also allowed for more fine-grained mistake detection in questions such as in figure 1, where instead of generic feedback related to off-by-one errors (or -two, one at each end of the loop), the feedback was generated based on the specific confusions in each randomly-generated question between < and ≤ and between > and ≥.

2.2 Recursive functions

Another aspect that students often struggle with is grasping recursion, though that is a component of computational thinking (and, arguably, a shibboleth to be probed in job interviews). Students are encouraged to think about base cases, and to consider transforming one or more solutions to a subproblem into the solution to the whole problem, but the details are important and it is easy for students to be lulled into a false sense of security by doing a small number of exercises – or, alternatively, to not experience the desired moment of enlightenment, and feel that forward progress is not possible.

In order to help our students measure their understanding of recursion, we generated in our mini-language multiple recursive

[3] https://docs.moodle.org/en/GIFT_format

```
(defun return-break-continue-for (n)
  (dotimes (i n)
    (let* ((form (make-break-continue-for-form))
           (answer (interpret-form form))
           (osub '((break . continue) (continue . break)))
           (oform (sublis osub form)
           (other (interpret-form oform))
           (nsub '((break . progn) (continue . progn)))
           (nform (sublis nsub form)
           (neither (interpret-form nform)))
      (insert (format "::R.%s::" (make-name form)))
      (insert (format "What is the return value from the
following block of pseudocode?\nYou may assume that the
value for all variables before the start of this block
is 0.<br/>\n"))
      (dolist (x (format-form form))
        (insert
         (format
          "  %s<br/>\n"
          (replace-regexp-in-string " " " " x))))
      (insert (format "{#
  =%s\n
  =%%0%%%s#have you mixed up break and continue?\n
  =%%0%%%s#are both increments executed?\n}\n\n"
                      answer other neither)))))
```

Listing 2: Emacs Lisp code using the form generator from listing 1, modifying and interpreting the form in order to generate answers which might be given by students with a mistaken mental model. This function outputs n questions of this form in Moodle's GIFT format, ready to be imported.

What code fragment should replace Z for function A to return the difference between a and b? You may assume that the initial arguments to the function A are positive integers and that b ≤ a.

```
function A(a,b)
    if a = b then
        return 0
    else
        return Z + 1
    end if
end function
```

- ◯ A(a, b + 1)
- ◯ A(a, b - 1)
- ◯ A(a - 1, b - 1)
- ◯ A(a - 1, b + 1)
- ◯ A(a + 1, b - 1)
- ◯ A(a + 1, b + 1)
- ◯ A(a - b, b)
- ◯ none of the other answers

Figure 3: example code building question, question 6 of the Recursive Algorithms quiz

implementations of basic mathematical operations (addition, subtraction, multiplication, division, exponentiation), and used our pretty-printer to generate questions where the conditional, base case, or recursive call had been elided. We collected from the variants the corresponding possibilities for each of these code locations,

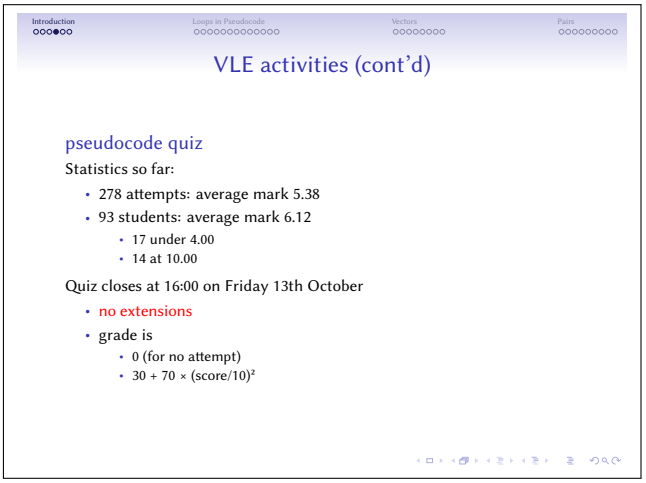

Figure 4: A slide from the lecture given after the quiz on pseudocode had been open for a week.

and generated multiple-choice questions with a subset of these possibilities as choices (see figure 3 for an example). We were able to ensure that we did not mistakenly include an accidentally-correct answer from the possibilities as a distractor, by substituting in each possible response into the corresponding functional form, interpreting it for randomly-chosen arguments, and checking that it did *not* return the mathematically correct answer.

3 DELIVERY AND RESULTS

Throughout the course, a new quiz on an individual topics (such as those described in sections 2.1 and 2.2) was made available to the students each week, with each quiz open for a 12-day period, from Monday at 09:00 until 16:00 on the Friday of the following week. The students were informed that each such quiz would be worth 1% of their final grade, and at the mid-point of the open period were shown a summary of the current cohort performance in that quiz (see figure 4). The non-linear transformation in that slide, from quiz score (out of 10) to awarded mark (out of 100) is to encourage participation (a mark of 30% is a fail, but not a bad one) and to avoid rewarding guesswork (a quiz score of 2 or 3 out of 10, as might occur through chance, still leads to a failing mark).

The cohort of 120 students took the pseudocode quiz 588 times, achieving scores plotted in figure 5. We tracked the improvement in quiz scores relative to each student's first attempt in that quiz, to attempt to measure the effect of practice and feedback (figure 6), which displayed a general improvement with diminishing returns and levelling off at around eight attempts; and students' best scores in the quiz plotted against the time of their first attempt (figure 7), where we found that there was no difference in the final outcome provided the student started the quiz activity before two or three days before the deadline.

The cohort's performance in question six (the simple loop question) is given in table 1, while their performance in question eight (the loop with **break**/**continue**) is shown in table 2.

As can be seen from table 1, student performance in this question is considerably better in the aggregate of the best performance of

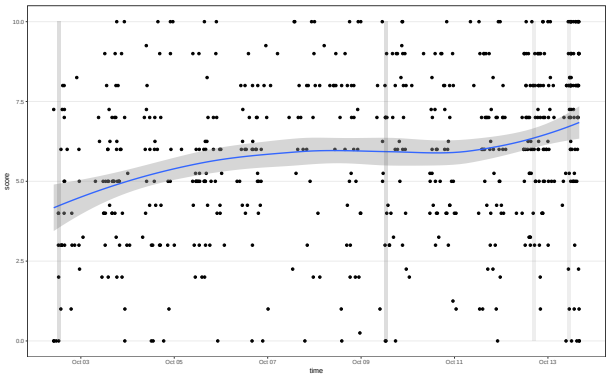

Figure 5: Individual scores (out of 10) in the pseudocode quiz over the period of its availability. The vertical shaded areas represent contact times (lectures and lab sessions).

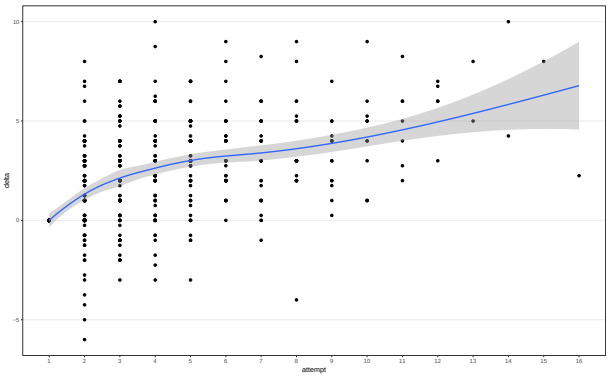

Figure 6: Improvement in student scores in the pseudocode quiz compared with the score attained in their first attempt.

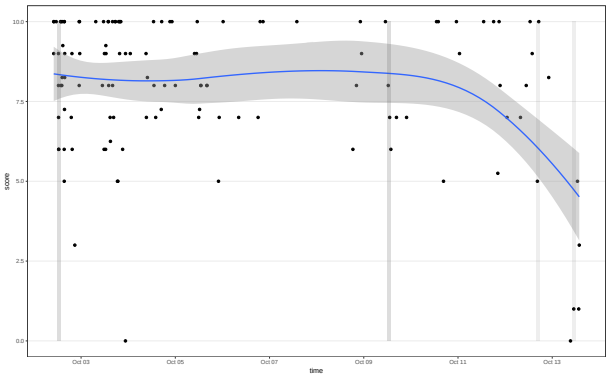

Figure 7: Students' best scores in the pseudocode quiz, plotted against the time when they took their first attempt at the quiz.

each student than in the other questions. This is expected; what might be unexpected is the degree to which the specific issue of off-by-one errors has been reduced. In the questions representing

	correct	off-by-one	incorrect	unanswered
best	83	11	21	6
other	193	90	135	49

Table 1: classified results for question 6 of the Pseudocode quiz: the "incorrect" column refers to answers given but neither correct nor off-by-one.

	correct	incorrect	unanswered
best	76	35	10
other	123	218	124

Table 2: classified results for question 8 of the Pseudocode quiz. Unfortunately the different categories of incorrect answers (confusion between break and continue, failure to consider how it interacts with the for loop) are not easy to differentiate from the Moodle reports.

	correct	incorrect	unanswered
best	68	46	1
other	69	165	14

Table 3: classified results for question 6 of the Recursive algorithms quiz.

the best attempts by each student, the error rate corresponding to off-by-one errors is 11 in 121, or 9.1%, this is a reduction from 19.3% in the population of non-best attempts, or roughly a halving of this error. By contrast, other incorrect answers decreased from 28.9% in general attempts to 17.4% in the best attempts; a decrease of generic errors of roughly 40%. The decrease in the proportion of unanswered question reflects the observed pattern that for many students early attempts at the quiz under time pressure means that they run out of time before answering the harder questions in the quiz.

3.1 Student perspectives

Near the half-way point of the course, the students were asked to provide feedback, first in a non-anonymous custom questionnaire delivered using the Learning Management System, and second anonymously using the standard course evaluation questionnaire provided by the University. Neither method of soliciting feedback reached complete coverage of the students; indeed, only approximately half the cohort (n=61 students) completed the non-anonymous questionnaire, and even fewer (n=40 students) the standard course evaluation.

As well as the self-paced multiple-choice quizzes described in this paper, the students were given:

- automatically-graded lab exercises, typically to implement particular algorithms or data structures, with their implementation assessed for correctness and targetted feedback generated using JUnit[4] and cppunit[5], managed by the INGInious platform [2];

[4]https://junit.org/junit4/
[5]https://freedesktop.org/wiki/Software/cppunit/

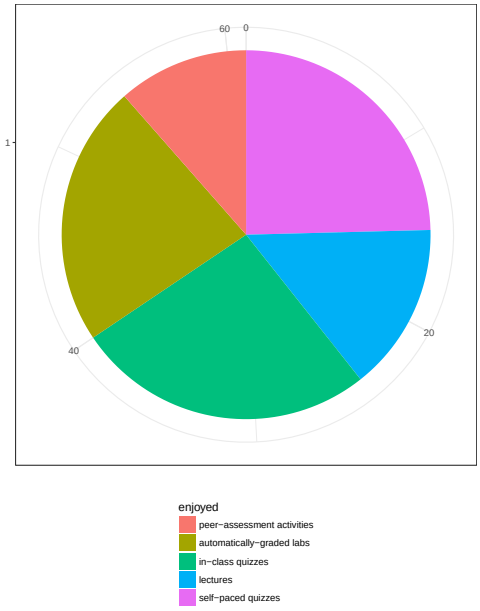

Figure 8: Student responses to the question "Which activity in this course so far have you most enjoyed?"

- peer-assessed extended exercises, with more open briefs than the lab exercises, and an assessment rubric set up for them to assess each others' submissions;
- in-class multiple-choice quizzes, typically given at the half-way point of a lecture, reinforcing or revising particular points, delivered using kahoot![6];
- standard weekly lectures of two hours' duration.

Figure 8 shows the student answers to the question of which of the various activities they most enjoyed in the non-anonymous questionnaire. The 61 respondents divide fairly evenly between the five classes of activity, with a slight preference for self-paced quizzes, in-class quizzes and lab exercises compared with lectures and peer-assessment. The responses to the question of which activities the students considered most or least instructional, however, are starkly different; figure 9 illustrates the answers to those two questions, with positive counts representing answers to the "most" variant and negative counts representing "least". From these responses, we see that the students value highly the automatically-graded lab exercises and particularly the multiple-choice quizzes; very few students considered the quizzes the least instructional activity, compared with over one-third who considered them the most; students are clearly distinguishing between enjoyment and pedagogy, in that the in-class quizzes, which were considered to be most enjoyable by many students, were rated as being most instructional by very few.

Students were also encouraged to leave free-text comments in both questionnaires. Some expressed frustration about particular aspects of multiple-choice quiz delivery, requesting that the time limit for quiz attempts be raised or the enforced gap between attempts be lowered; however, several commented on the level of challenge

[6]https://kahoot.com/

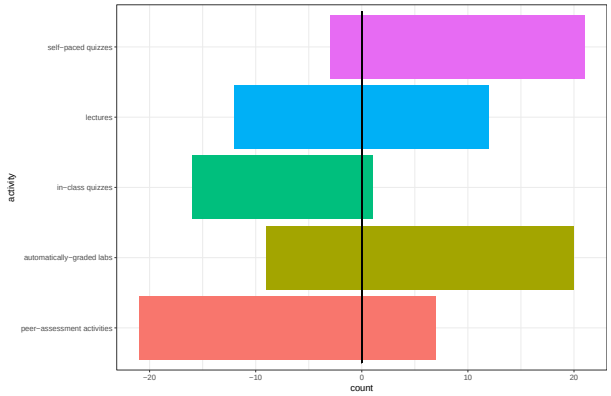

Figure 9: Student responses to the question "Which activity in this course so far have you learnt most [positive counts] / least [negative counts] from?"

posed by the quizzes, and there have been in-person requests to make the quizzes available after the deadline for that quiz to help students guide their further learning and revision.

Student engagement in the quiz activities has remained high; in the 16 completed quizzes in this academic year, the students have submitted 6792 quiz attempts, each with 10 questions (so each student has, on average, received automated feedback on 566 individual questions).

3.2 Educator perspectives

Using multiple-choice questions with the approach given in this paper has several benefits from the perspective of an educator. Firstly, and most importantly, it provides for instruments which the students can take, multiple times, in order to judge their own state of understanding of the foundational components of the material and receive feedback regarding where and how that understanding might be lacking. Importantly, it allows that feedback to be delivered and received at a time of the student's own convenienence; once the questions are generated and the automation set up, there is no additional cost, freeing up educator time to devise more useful activities or provide extra material.

In addition, this approach can scale to the required size; this entire course was delivered to a cohort of 120 students using one instructor and one teaching assistant; this course *does* have some non-automated components of delivery, such as moderation of peer-assessment, marking of an individual written assignment, and marking a final exam with longer-form questions. Scaling to larger student numbers, as in a fully-online or MOOC setting, might require some additional instructor time to monitor student questions on online forums – though our experience in running this course this year is that the students themselves are well equipped to assist each other on public forums, and indeed it is acknowledged that helping each other in this way helps to consolidate learning and build mastery; the teaching staff participation on the forums is largely limited to administrative announcements and to provision of material beyond the formal syllabus.

Further, it is important for us to know whether our students (as a whole) have a good understanding of the material, a mixed understanding, or maybe that a substantial part of the cohort has misunderstood some topic. One benefit of having the quizzes open for 12 days was that, at the half-way point, we could examine the results so far and identify whether any specific part of the quiz showed substantially worse (or worse than expected) performance – and if so, that specific item could be addressed in a plenary session such as a lecture.

A concern sometimes raised about using self-paced, remotely-administered tests as a component of a final grade is that students might be incentivised to cheat, for example by asking other people to take the test on their behalf. One mitigation is that, since each of these tests is worth 1% of their final grade, and students can get a mark of 30% simply by submitting a blank entry, there is limited upside to cheating; meanwhile, we performed spot-checks on individual elements of suspicious behaviour, by requiring some students to take quizzes under controlled conditions after identifying anomalies in the logs (such as two students taking the same quiz from the same IP address in quick succession – the students replicated their scores of 10, and revealed that they had been racing each other!). If the concern is strong, there is nothing in the approach described here which would prevent quizzes being used primarily formatively, and possibly assessed for part of a course mark under controlled conditions.

We would not expect to be able to be able to build a community of users of our specific mini-language and toolset; it grew to meet immediate needs, and it fulfils those needs minimally. The general approach – identifying potential pitfalls or barriers to understanding, designing assessments that probe those barriers, and providing specific feedback in the case of students demonstrating that they are struggling with those barriers – is, we believe, sound, and we have demonstrated that at least in some subject areas this can be done in a scalable way. It is perhaps surprising not to see this approach taken up more widely; we speculate that this is because the technical sophistication level required to operate the toolchain is fairly high; the up-front cost of development is steep; and that the pedagogical approach taken implies more empathy with the student and more responsibility for the learning journey, which are not necessarily aspects selected for in hiring teaching staff at University.

4 CONCLUSIONS AND FUTURE WORK

We believe that providing automated tools where students can probe in detail their own understanding of the fundamentals of the curriculum that they are studying is valuable. This provision will also become more necessary as Higher Education Institutions become more resource-constrained, as students expect more for their tuition fees, and as competitors such as OpenCourseWare and MOOCs establish the principle in students' minds that pedagogical materials are available for anyone to access for free.

In the specific case of a Computer Science curriculum, and Algorithms & Data Structures specifically, we identified tangible benefits to the practical pedagogy from the use of a Lisp with strengths both in treating code-as-data and data-as-code, and for text manipulation. As well as the Lisp-based mini-language for interpreting and formatting pseudocode described in this paper, we implemented an Emacs major mode for editing GIFT-format files, to make hand-edits to generated questions fast and practical; we implemented simplified versions of many elementary and more complex data structures, in order to be able to generate questions on the behaviour of hash-tables or the properties of graphs; and all this under time pressure and on a budget.

For this specific course, one potential improvement would be to implement a parser for the surface syntax of pseudocode, converting it back to our sexp-based language. This, in combination with a plugin for the LMS, would allow us to set free-text rather than multiple-choice questions for topics such as recursive algorithms, where even with the large number of distractor questions there is some chance that some students will select right answers by pattern-matching, without a full understanding of the material.

There are improvements that can be made in delivering multiple-choice quizzes compared with this year. While giving specific feedback to the students about particular mistakes is helpful, it is also useful for instructors to know that this has happened. The Moodle LMS does not make it easy to see the frequency that particular wrong answers are given from the reports that it produces; however, we could allocate distinct fractional marks, rather than zero, to specific wrong answers; this would not substantially affect the quiz score (a wrong answer scoring 0.01 points instead of 0.00 will not have a material effect on a student outcome) but would make it much more straightforward to analyse data extracted from the LMS.

REFERENCES

[1] Jaap Boender, E. Currie, M. Loomes, Giuseppe Primiero, and Franco Raimondi. Teaching functional patterns through robotic applications. In *Proceedings of the 4th and 5th International Workshop on Trends in Functional Programming in Education, TFPIE 2016, Sophia-Antipolis, France and University of Maryland College Park, USA, 2nd June 2015 and 7th June 2016.*, pages 17–29, 2016. doi: 10.4204/EPTCS.230.2. URL https://doi.org/10.4204/EPTCS.230.2.

[2] Guillaume Derval, Anthony Gego, Pierre Reinbold, Benjamin Frantzen, and Peter Van Roy. Automatic grading of programming exercises in a MOOC using the INGInious platform. In *Proceedings of the European MOOC Stakeholder Summit*, pages 86–91, Mons, May 2015.

[3] Darina Dicheva, Christo Dichev, Gennady Agre, and Galia Angelova. Gamification in Education: A Systematic Mapping Study. *Educational Technology & Society*, 18(3):75–88, 2015.

[4] Paul F. Dietz. The GNU ANSI Common Lisp Test Suite. Presented at International Lisp Conference, Stanford, July 2005, 2005. http://cvs.savannah.gnu.org/viewvc/*checkout*/gcl/gcl/ansi-tests/doc/ilc2005-slides.pdf.

[5] Susan Harter. A new self-report scale of intrinsic versus extrinsic motivation in the classroom: motivational and informational components. *Developmental Psychology*, 17:302–312, 1981.

[6] Stephen E. Newstead, Arlene Franklyn-Stokes, and Penny Armstead. Individual differences in student cheating. *Journal of Educational Psychology*, 88(2):229–241, 1996.

[7] Elizabeth Patitsas, Jesse Berlin, Michelle Craig, and Steve Easterbrook. Evidence that computer science grades are not bimodal. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, pages 113–121, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4449-4. doi: 10.1145/2960310.2960312. URL http://doi.acm.org/10.1145/2960310.2960312.

[8] Franco Raimondi, Giuseppe Primiero, Kelly Androutsopoulos, Nikos Gorogiannis, Martin Loomes, Michael Margolis, Puja Varsani, Nick Weldin, and Alex Zivanovic. A racket-based robot to teach first-year computer science. In *European Lisp Symposium*, IRCAM, Paris, May 2014.

[9] Juha Sorva. Notional machines and introductory programming education. *Trans. Comput. Educ.*, 13(2):8:1–8:31, July 2013. ISSN 1946-6226. doi: 10.1145/2483710.2483713. URL http://doi.acm.org/10.1145/2483710.2483713.

[10] Jeffrey D. Ullman. Gradiance on-line accelerated learning. In *Proceedings of the Twenty-eighth Australasian Conference on Computer Science - Volume 38*, ACSC '05,

pages 3–6, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc. ISBN 1-920-68220-1. URL http://dl.acm.org/citation.cfm?id=1082161.1082162.

[11] Jeannette M. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, March 2006. ISSN 0001-0782. doi: 10.1145/1118178.1118215. URL http://doi.acm.org/10.1145/1118178.1118215.

ACKNOWLEDGMENTS

Thanks to Alexandre Rademaker for pointing out the similarity of this approach to the *root question* concept of Gradiance. Jonas Bernoulli and Alvarez Gonzales Sotillo made contributions to the Emacs Lisp `gift-mode` library[7] for editing GIFT-format quizzes, and Steve Purcell accepted a pull request for its inclusion in MELPA. Thanks are due also to colleagues and (particularly) students at Goldsmiths for making helpful suggestions, pointing out errors, and participating in the activities.

[7] https://github.com/csrhodes/gift-mode