AAEE 2012 CONFERENCE
Melbourne, Australia
www.aaee.com.au/conferences/2012/

Exams in computer programming: what do they examine and how complex are they?

Simon[a]; Judy Sheard[b]
University of Newcastle[a], Monash University[b]
Corresponding Author Email: simon@newcastle.edu.au

BACKGROUND
Computer programming is taught to students of engineering, IT, science, and other disciplines. It is taught using many different programming languages, and with no clear agreement as to what material should be covered and to what depth. We are investigating the exams in introductory programming courses to explore what they have in common and how they differ. For each question in an exam we note the topics covered, the overall difficulty of the question, and six different measures of question complexity.

PURPOSE
Our purpose is to acquire an understanding of the variability to be found in computer programming exams, with a view to establishing the extent to which they can be said to be assessing the same thing, and the extent to which they share a similar level of complexity.

DESIGN/METHOD
We analysed every question in 15 introductory programming exams using a set of measures designed for the purpose in light of the literature that we were able to find on related topics. The classifying was carried out by a team of academics after an inter-rater reliability test to ensure that the members of the team were classifying with reasonable agreement.

RESULTS
We have found clear variation in the exams that we studied. Most have a good mix of easy, moderate, and hard questions, although a third of them have no questions that we considered hard, and in two exams half or more of the marks are for questions that we considered hard. All of the other complexity measures correlate with overall question difficulty, suggesting that a number of different factors can contribute to a question being considered hard.

CONCLUSIONS
This study finds that, beyond the inevitable variation in exams, it can reasonably be concluded that introductory programming exams are assessing the same topics, and doing so with questions at comparable levels of complexity. The study also serves to make examiners aware of the distinction between ‘good’ complexity, such as questions involving inherently complex tasks, and ‘bad’ complexity, such as unnecessarily convoluted questions.

KEYWORDS
Computer programming, assessment, examination, question complexity, question difficulty.

Introduction

There is some suggestion that computer programming should be considered part of a broad liberal education and should be taught to most or all university students (Guzdial & Ericson, 2012). While that suggestion has not taken hold, programming is certainly taught to many students in engineering, IT, science, and other disciplines. Yet many of these students struggle to come to grips with programming (McCracken et al., 2001), and the anecdotal evidence is that these students would be a great deal happier if programming were not a required part of their studies. Nevertheless, it does tend to be a requirement. Therefore we have chosen to investigate the extent to which there is any sort of standard across programming courses, and to do that by examining the final exams of the courses to see what they have in common.

Programming is generally taught using a specific programming language as a vehicle, and this purpose is served by numerous languages including C, Matlab, Java, C++, Python, and more. While it is generally agreed that the object of this teaching is programming itself rather than the particular language, there is little evidence of agreement as to what programming concepts should be covered and to what depth. As a way of exploring what is taught in different introductory programming courses, we are investigating the exams in those courses to explore what they have in common, in terms both of the programming concepts that they cover and of the complexity of the questions. Armed with the knowledge of what material the exams cover and how complex they are, we feel that we will be in a position to start addressing the question of what the generic introductory programming course actually teaches, and to what extent individual programming courses mesh with the generic model.

Background

Four years ago Goldfinch, Carew, Gardner, Henderson, McCarthy, and Thomas (2008) presented a cross-institutional comparison of mechanics exams at AAEE, with the principal goal of identifying the most common areas of difficulty for students. Their subgoals included establishing what variation could be found between the exams at the four institutions they studied, identifying similarities between the exam papers, and identifying the concepts assessed and the difficulty levels of the exam questions. We have carried out similar work in introductory programming (Simon et al 2012). Also in the computing area, although with less obvious relevance to engineering, Simon, Clancy, McCartney, Morrison, Richards, and Sanders (2010) conducted a similar investigation into exams in data structures.

All three of these projects listed the topics covered in the exams they studied, and all three noted that the participating academics found it hard to reach agreement on the level of difficulty of each question.

Listing the topics covered in exams – what do they examine? – is relatively straightforward, but remains a necessary aspect of any thorough investigation of the exams. Our principal goal in this paper is to address the second question in the title – how complex are they? We are still concerned with assessing the difficulty of the exam questions, but unlike the previous studies, which have assessed question difficulty entirely holistically, we have examined a number of aspects of question complexity, which we believe are potentially easier to determine than overall difficulty, and then assessed the overall question difficulty in the light of those complexity measures.

While we believe that many teachers of programming would argue that the ‘real’ course content is independent of the programming language used, we do not entirely agree. While this is probably true for similar languages such as Java, C++, and Visual Basic, we note, for example, that while iteration would be a fairly elementary topic in each of these languages, iteration is not to be found at all in functional programming languages such as Scheme, in which recursion is the ‘natural’ way to achieve repetitive processing. We have therefore chosen to limit this particular study to programming courses that use the same language as

their vehicle. We have gathered nearly 30 introductory programming exams from 13 universities in five countries. The languages taught in these courses include Java, C, C#, Alice, Scheme, Haskell, Python, and Visual Basic; but one language, Java, is used by 15 of the courses, so we have restricted this analysis to those 15 exams.

The classification scheme

For each question we noted its percentage mark in the exam, which we subsequently used to weight the question's other properties. To address the question 'what do they examine?' we listed up to three topics that were covered by each question in each exam. To address the question 'how complex are they?' we applied six distinct measures of question complexity, some of which are based upon those originally used by Williams and Clarke (1997) in the domain of mathematics. Finally, we assessed the overall difficulty of each question. These measures are summarised in Table 1, and explained in the Results section.

While there is not space to go into detail here, we conducted a rigorous inter-rater reliability test, similar to that reported in Simon et al (2012), to ensure that the classifiers were classifying consistently. We then classified the questions in pairs, with the members of each pair first classifying individually and then discussing any differences before deciding on an agreed value for each classification.

Table 1: Exam question classification scheme. The middle column indicates whether the measures apply only to the question or to both the question and the answer.

Measure	Focus	Classification values
Percentage mark	Q & A	number between 1 & 100 inclusive
Topics	Q & A	selection from a list of 30 topics
External domain references	Q only	low, medium, high and specify the external domain
Explicitness	Q only	high, medium, low (note order of levels)
Linguistic complexity	Q only	low, medium, high
Conceptual complexity	Q & A	low, medium, high
Code length	Q & A	low, medium, high, NA
Intellectual complexity (Bloom level)	Q & A	knowledge, comprehension, application, analysis, evaluation, synthesis
Degree of difficulty	Q & A	low, medium, high

Results

In this section we present the results of analysis of questions from the 15 introductory programming exam papers according to the different measures presented in Table 1. These exams contained a total of 371 questions, with the number of questions in an exam ranging from seven to 41. For each question the percentage mark allocated was used as a weighting factor when calculating the question's contribution to the values in each measure.

Exam paper characteristics

The 15 exam papers were sourced from nine institutions in five countries – Australia, New Zealand, Finland, England, and the United States of America. The exams were used to assess students in introductory programming courses, 14 at the undergraduate and one at the postgraduate level. Enrolments in the courses varied from 25 students on a single campus to 800 students over six campuses, two of these being international campuses. The language of instruction in all courses was Java, used with a variety of development environments (BlueJ, JCreator, Netbeans, Eclipse). Most of the exams were to be written entirely on paper, but two of them included a substantial component undertaken at the computer.

Topics covered

Whereas Goldfinch et al (2008) identified only nine mechanics topics that they found in two or more of the exams they classified, we have identified 30 distinct topics that we might expect to find covered in the assessment for a computer programming course. The list of topics was sourced initially from the computing education literature. Dale (2005, 2006) listed the topics that emerged from a survey of computing academics; Schulte and Bennedsen (2006) surveyed teachers of introductory programming to determine the topics that were taught; Elliott Tew and Guzdial (2010) identified topics by analysing the content of relevant textbooks. Having compiled a list based on these sources, we added to it from our own experience before starting to classify the exam questions.

For each question we recorded up to three topics that we considered were central to the question. A total of 26 topics were identified in the 15 exams. The percentage coverage of the topics is presented in Table 2. The most prevalent topic was object-oriented concepts, which was covered by 42% of the marks. The next three most prevalent topics were methods, loops, and arrays. Eight topics had less than 2% coverage and four topics from our list were not found at all in the 15 exams.

Table 2: Coverage of topics in the 15 exams

Topic	Coverage
object-oriented concepts	42%
methods and functions	34%
loops	32%
arrays	26%
program design	14%
selection	13%
input/output; file input/output; parameter passing	8% each
assignment; strings; data types and variables	5% each

Topic	Coverage
error handling	4%
arithmetic operators; collections other than arrays	3% each
relational operators; scope (includes visibility); testing	2% each
constants; events; expressions; lifetime; logical operators; notional machine; programming standards; recursion	<1% each
graphical user interfaces; operator overloading; algorithm complexity; class libraries	0

Figures 1, 2, 3 and 4 show the coverage in each exam of the four most prevalent topics – object-oriented concepts, methods, loops and arrays. With one or two obvious exceptions, each of these dominant topics is covered by 20%-60% of the weighted questions in each exam.

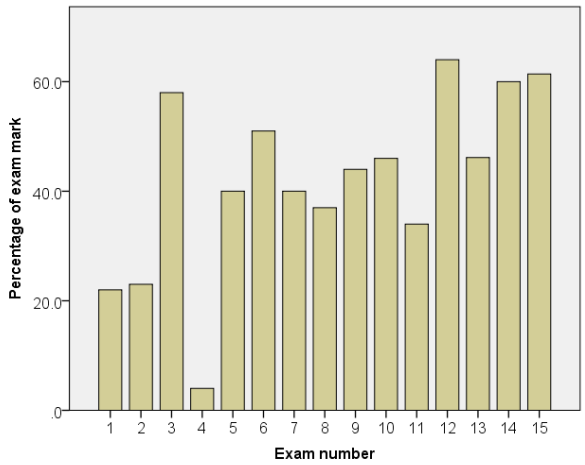

Figure 1: Coverage of OO concepts

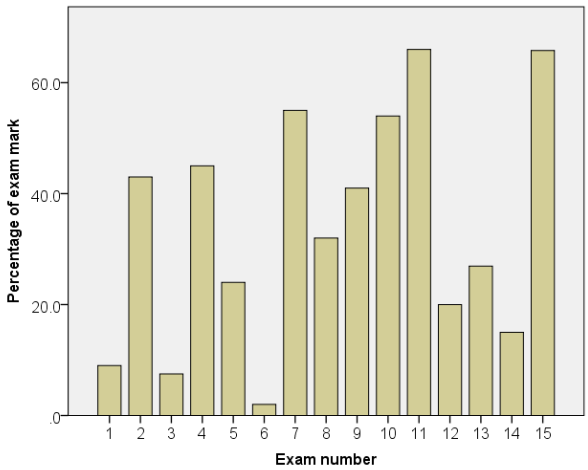

Figure 2: Coverage of methods

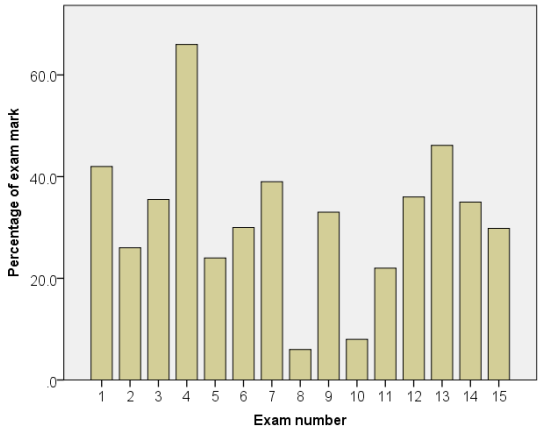

Figure 3: Coverage of loops

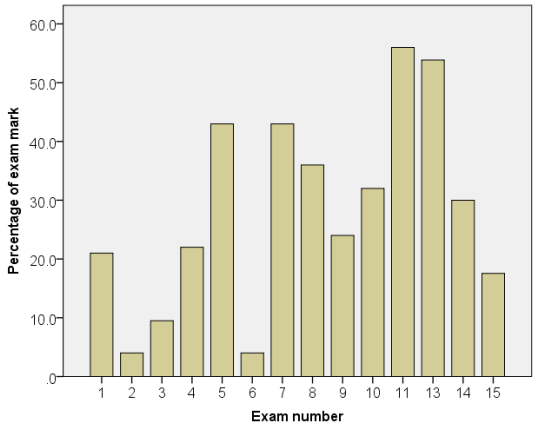

Figure 4: Coverage of arrays

Bearing in mind that we were nominating up to three topics for each question, we are not concerned that some topics have coverage of 60% or more; this does not suggest an unhealthy concentration on single topics, because each exam could potentially record up to 300% total coverage of topics.

External domain references

External domain references are terms, activities, or scenarios that may be specific to a culture or context in a way that might influence the ability of those outside the relevant group to answer a question. For example, a question might ask students to write code to implement the scoring scheme in a game of Australian football. If the question then explains the scheme fully, we would mark it as having low external domain references; if the question explains the scheme adequately, but in a way that might leave some students believing they need more information, we would mark it as having medium external domain references; and if the question simply assumes that students know and understand Australian football scoring, we would mark it as having high external domain references.

Overall, 96% of the weighted marks involved a low (that is, non-existent) level of external domain knowledge. Questions requiring a medium level of external domain knowledge were found in less than half of the exams, and no questions involved a high level of external domain knowledge.

Linguistic complexity

Linguistic complexity is concerned with the length, sophistication, and general comprehensibility of the question. As with external domain references, this has a clear potential to affect a student's ability to answer the question.

Overall, the level of linguistic complexity was low for most questions (92%). Only two questions of high linguistic complexity were found; however, questions of medium level complexity were found in more than half of the exams.

Explicitness

Explicitness assesses the extent to which the question specifies what the students need to know in order to answer the question. A question with low explicitness will assume that students already know, or can deduce, much about the task to be completed, whereas a highly explicit question will more or less tell the students exactly what to do, in English, requiring them just to translate those instructions into program code.

Unlike the rest of the complexity measures, a high rating in this measure suggests that the question will be easier to answer, and a question with low explicitness will tend to be harder to answer.

The level of explicitness of questions in each exam is shown in Figure 5. All exams had highly explicit questions. Only a third of the exams had questions with a low level of explicitness. (Totals of more than 100% indicate exams that offer students some choice in the questions they answer.)

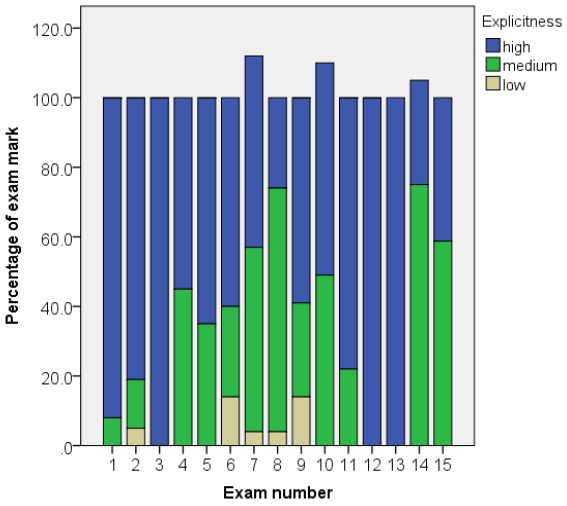

Figure 5: Explicitness of questions

Conceptual complexity

Conceptual complexity measures the types and combinations of the concepts that students must know in order to correctly answer a question.

Schulte and Bennedsen (2006) surveyed teachers of introductory programming to determine the topics that were taught and the perceived difficulty of those topics; other surveys of computer programming topics, such as those of Dale (2005, 2006) also sought consensus on the difficulty of each topic included in the survey. We have used these perceived difficulties to organise topics into low, medium, or high conceptual complexity. We also recognise the compounding effect: a question involving a combination of medium-level topics might be classified as high conceptual complexity if the manner of combination is enough to add complexity to the overall task. Figure 6 shows the conceptual complexity of the questions in each exam.

Code length

Even when students are asked to read program code rather than write it, the length of the code has a clear potential to affect the question difficulty, so this measure was applied to questions involving code in the question, the answer, or both. As a rule of thumb, questions involving up to six lines of code were coded as low code length, questions from six to 24 lines of code were classified as medium, and questions over 24 lines were classified as high code length. There was a large variation in the amount of code involved in questions in the exams, as shown in Figure 7.

Intellectual complexity

Intellectual complexity gives a measure of where the demands of a question fit into Bloom's taxonomy (Anderson & Sosniak, 1994). The Bloom level of questions in each exam is shown in Figure 8, which shows wide variation across the exams. Every exam had questions at the comprehension level. Questions at the highest Bloom levels were less common, but in most exams the majority of questions fall into the application level, which is perhaps not surprising for courses in computer programming.

Degree of difficulty

Finally we estimated how difficult the average student would find the question at the end of an introductory course. This classification is similar to that used by Goldfinch et al (2008) in their analysis of mechanics examination papers, Simon et al (2012) in their analysis of introductory programming exams, and Simon et al (2010) in their analysis of data structures exam papers.

As shown in Figure 9, there is clear variation in the perceived level of difficulty of questions across the 15 exams. Overall, half the marks were allocated to questions rated as medium (49%), with 28% for questions rated as low difficulty and 23% for question rated as high

difficulty. Almost three quarters of the exams had a mix of low, medium and high questions; however, a quarter of the exams had no high-difficulty questions and one exam had no low-difficulty questions.

Despite the variation, there is a clear trend to be observed: in most exams some 30% of marks are for questions of low difficulty, some 50% for questions of medium difficulty, and the rest for questions of high difficulty.

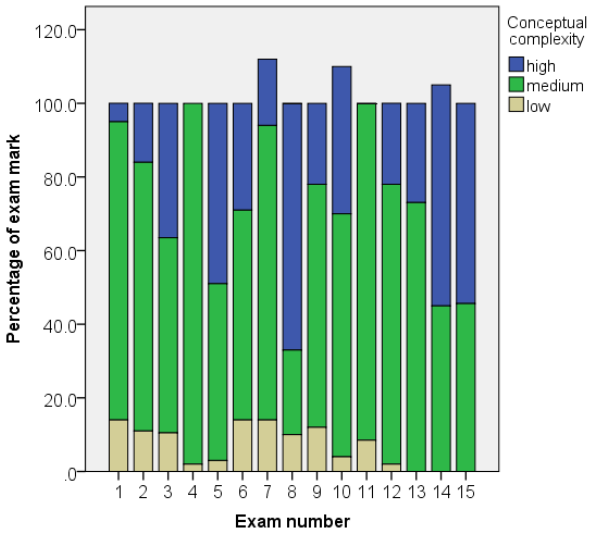

Figure 6: Conceptual complexity of questions

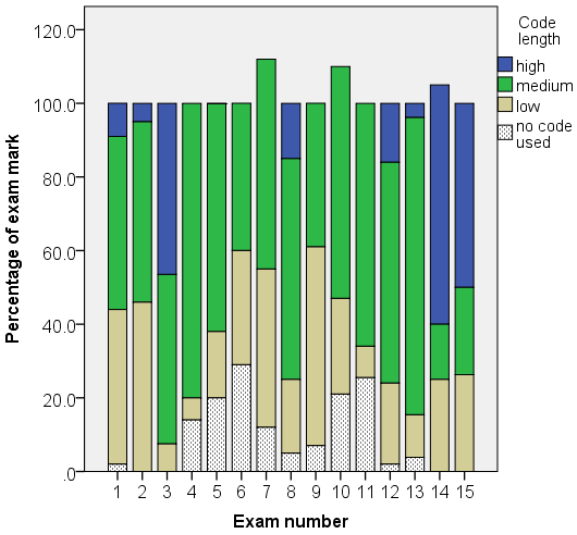

Figure 7: Amount of code in questions

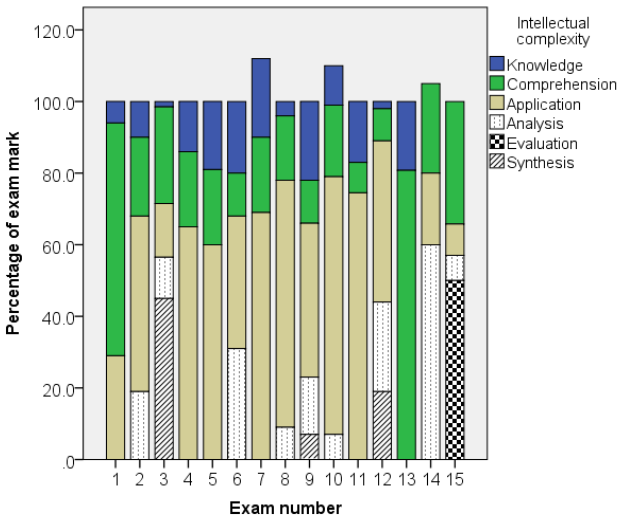

Figure 8: Intellectual complexity of questions

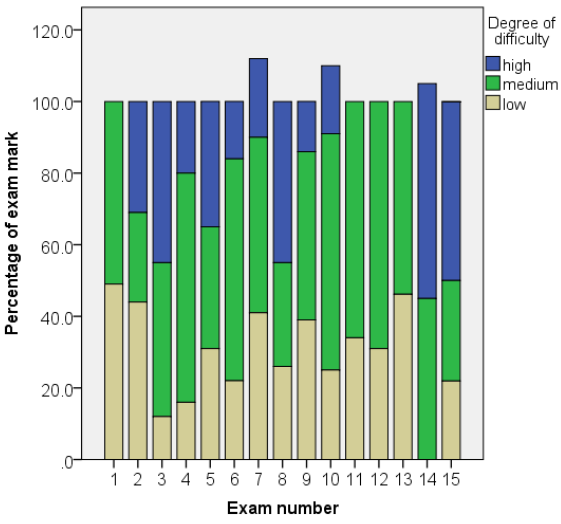

Figure 9: Difficulty of questions

What makes a question difficult?

We were interested in determining what characteristics of a question might contribute to the perception that it is difficult. To explore this we investigated relationships between the perception of difficulty and the six complexity measures. As the measures of complexity and difficulty were at the ordinal level, a Spearman's Rank order correlation was used. The results show a relationship between the degree of difficulty and all measures of complexity, all of which were significant at $p < 0.01$. The strongest relationships with degree of difficulty were code length and intellectual complexity.

This supports our belief that these six measures of question complexity all contribute to the overall difficulty of a question, and suggests that it will be easier for researchers to

investigate the difficulty of exam questions if they are willing to first consider the various forms of complexity by which the question might be classified.

Good and bad complexity

Of the six measures of complexity, which of them do we think are reasonable in an exam and which might we consider not so reasonable?

We believe that there is a clear place in a final exam for conceptual complexity (we must examine the more challenging topics that we have taught), for intellectual complexity (it is reasonable to ask questions that take students beyond the lower levels of Bloom's taxonomy), and code length (it is reasonable, even in a written exam, to ask students to read and/or write pieces of code that are more than half a dozen lines long). In that sense, these measures refer to what we might call good complexity. At the same time, we believe that questions at the higher levels of these complexity measures should be used sparingly: it would not be reasonable to write a whole exam that tests just the more complex concepts, with large chunks of program code, at the higher levels of Bloom's taxonomy.

We believe that external domain references and linguistic complexity could be called bad complexity. It is not reasonable to write exam questions that rely on students to be familiar with extraneous material that was not taught in the course – material such as the scoring system for Australian football. It is perfectly reasonable to write questions that involve such material; but if that is to be done, the material should be adequately explained in the questions. It is also not reasonable to couch questions in complex language when simpler language could be used to say the same thing. It is tempting to suggest that this is particularly the case when we are teaching so many students for whom English is not the first language; however, it seems to be increasingly the case that even students with English as their first language are not necessarily at home with the intricacies of that language, and will deal better with exam questions if they are expressed in very simple terms and structures.

The final measure of complexity is explicitness, and on that measure we have no simple recommendation. There is a case for being reasonably explicit when telling students what they are required to do. However, if every question in an exam lays out a problem solution in minute detail and simply asks the students to write the corresponding code, the students are not being tested on the exceedingly important skill of reading the statement of a problem and devising the solution to that problem in order to code the solution as a computer program. Problem solving is often hailed as a vital skill to be imparted in programming courses, and it is therefore reasonable to assess that skill. On balance, therefore, we would group explicitness with the good measures of complexity, and recommend that a good mix of explicitness levels be built into the questions of a programming exam.

Conclusion

We have analysed 15 final exams, from courses that teach programming by way of Java, to find out what the exams assess and how complex they are. In terms of what the exams assess, we found six topics with more than 10% coverage in the exams, six more topics with over 5% coverage, and further topics with lower coverage. Examining the coverage on each exam of the top four topics, we conclude that while there is clear variability in the exams, there is a case for suggesting that the exams are assessing the same topics.

With regard to complexity, we rated each question according to six measures of complexity; we confirmed that each of these measures correlates to our assessment of overall question difficulty; and we have proposed that four of these measures are good complexity, something that it is reasonable to include to an appropriate extent in exams, while the other two are bad complexity, and should generally be avoided when writing exams. We have noted the range of values of each complexity in each of the exams, and again, while there are clear differences from one exam to another, we believe that there is a reasonable comparability across the 15 exams we have analysed.

We can therefore conclude that, with the level of variability that must be expected when the courses are taught by different people and the exams are written by different people, these exams are indeed assessing the same thing, and at about the same levels of complexity.

References

Anderson, L. W., & Sosniak, L., A. (1994). Excerpts from the "Taxonomy of Educational Objectives, The Classification of Educational Goals, Handbook I: Cognitive Domain. In L. W. Anderson & L. Sosniak, A. (Eds.), *Blooms's Taxonomy: A Forty Year Retrospective* (pp. 9-27). Chicago, Illinois, USA: The University of Chicago Press.

Dale, N. (2005). Content and emphasis in CS1. *inroads - The SIGCSE Bulletin, 37*(4), 69-73.

Dale, N. (2006). Most difficult topics in CS1: Results of an online survey of educators. *inroads - The SIGCSE Bulletin, 38*(2), 49-53.

Elliott Tew, A., & Guzdial, M. (2010). *Developing a validated assessment of fundamental CS1 concepts.* Paper presented at the SIGCSE'10, Milwaukee, Wisconsin, USA.

Goldfinch, T., Carew, A. L., Gardner, A., Henderson, A., McCarthy, T., & Thomas, G. (2008). *Cross-institutional comparison of mechanics examinations: A guide for the curious.* Paper presented at the Australasian Association for Engineering Education conference (AAEE), Yeppoon.

Guzdial, M., & Ericson, B. (2012). *Introduction to Computing and Programming in Python - a Multimedia Approach*: Pearson Education.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Haga, D., Ben-David, K. Y., . . . Wilusz, T. (2001). ITiCSE 2001 working group reports: A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin, 33*(4).

Schulte, C., & Bennedsen, J. (2006). *What do teachers teach in introductory programming?* Paper presented at the Second International Computing Education Research workshop (ICER'06), Canterbury, UK.

Simon, Sheard, J., Carbone, A., Chinn, D., Laakso, M.-J., Clear, T., . . . Warburton, G. (2012). *Introductory programming: Examining the exams.* Paper presented at the 14th Australasian Computing Education conference, Melbourne, Australia.

Simon, B., Clancy, M., McCartney, R., Morrison, B., Richards, B., & Sanders, K. (2010). *Making sense of data structure exams.* Paper presented at the Sixth International Computing Education Research workshop (ICER'10), Aarhus, Denmark.

Williams, G., & Clarke, D. (1997). *Mathematical task complexity and task selection.* Paper presented at the Mathematical Association of Victoria 34th Annual Conference - 'Mathematics: Imagine the Possibilities', Clayton, Victoria, Australia.

Acknowledgements

We wish to thank the following colleagues who helped in the classification of exam papers: Angela Carbone, Donald Chinn, Tony Clear, Malcolm Corney, Michael de Raadt, Daryl D'Souza, Joel Fenwick, James Harland, Mikko-Jussi Laakso, Raymond Lister, Anne Philpott, James Skene, Donna Teague and Geoff Warburton.