How difficult are exams? A framework for assessing the complexity of introductory programming exams

Judy Sheard
Monash University
judy.sheard@monash.edu.au

Simon
University of Newcastle
simon@newcastle.edu.au

Angela Carbone
Monash University
angela.carbone@monash.edu.au

Donald Chinn
University of Washington, Tacoma
dchinn@u.washington.edu

Tony Clear
Auckland University of Technology
tony.clear@aut.ac.nz

Malcolm Corney
Queensland University of Technology
m.corney@qut.edu.au

Daryl D'Souza
RMIT University
daryl.dsouza@rmit.edu.au

Joel Fenwick
University of Queensland
joelfenwick@uq.edu.au

James Harland
RMIT University
james.harland@rmit.edu.au

Mikko-Jussi Laakso
University of Turku
milaak@utu.fi

Donna Teague
Queensland University of Technology
d.teague@qut.edu.au

Abstract

Student performance on examinations is influenced by the level of difficulty of the questions. It seems reasonable to propose therefore that assessment of the difficulty of exam questions could be used to gauge the level of skills and knowledge expected at the end of a course. This paper reports the results of a study investigating the difficulty of exam questions using a subjective assessment of difficulty and a purpose-built exam question complexity classification scheme. The scheme, devised for exams in introductory programming courses, assesses the complexity of each question using six measures: external domain references, explicitness, linguistic complexity, conceptual complexity, length of code involved in the question and/or answer, and intellectual complexity (Bloom level). We apply the scheme to 20 introductory programming exam papers from five countries, and find substantial variation across the exams for all measures. Most exams include a mix of questions of low, medium, and high difficulty, although seven of the 20 have no questions of high difficulty. All of the complexity measures correlate with assessment of difficulty, indicating that the difficulty of an exam question relates to each of these more specific measures. We discuss the implications of these findings for the development of measures to assess learning standards in programming courses.

Keywords: Standards, quality, examination papers, CS1, introductory programming, assessment, question complexity, question difficulty.

Copyright © 2013, Australian Computer Society, Inc. This paper appeared at the 15th Australasian Computing Education Conference (ACE 2013), Adelaide, South Australia, January-February 2013. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 136. A. Carbone and J. Whalley, Eds. Reproduction for academic, not-for-profit purposes permitted provided this text is included.

1 Introduction

In Australia there has been an increasing amount of attention placed on the government's higher education standards agenda, which aims to achieve quality assurance in a number of areas including the standard of qualifications and the learning outcomes of students in higher education institutions. The Tertiary Education Quality and Standards Agency (TEQSA) has been established to register and evaluate the performance of higher education providers against a new Higher Education Standards Framework (Tertiary Education Quality and Standards Agency, 2012). To ensure that standards are developed the government has formed a Standards panel (Evans, 2011) to set the benchmark for quality in higher education.

The interest in learning standards is not restricted to government agencies. In a recent online survey of Australian academics, with more than 5,000 respondents across 20 universities, 46.7% of respondents felt that academic standards were in decline (Bexley et al, 2011). From the student perspective, in a survey of nearly 10,000 graduates in 2008, 67% nominated "Challenge students to achieve high academic standards" as an area of potential improvement for undergraduate education (Coates & Edwards, 2008).

In this environment, the challenge currently facing academics in the Australian tertiary sector is how to develop learning standards and assess learning outcomes. As a way forward, the Australian government has funded eight groups to work within specific disciplines to develop learning standards: the minimum required knowledge, skill and capabilities expected of a graduate. The combined discipline group for Information and Communication Technology (ICT) and Engineering has begun its quest for learning standards by drawing on existing learning outcomes developed from the relevant professional bodies (Cameron & Hadgraft, 2010).

Proposals currently under consideration to assess the attainment of learning standards include the development

of national standardised tests of generic and disciplinary learning outcomes. In the engineering field, a feasibility study is looking into testing discipline-specific skills as part of an Assessment of Higher Education Learning Outcomes (AHELO) (Australian Council for Educational Research, 2011). As yet there is no comparable venture in ICT, where the idea of standardised testing seems to be a difficult one to address. There appear to be no clear processes, pathways, or in some cases, communities, to offer a coherent way forward to assessment of discipline-specific learning standards.

The work reported here forms part of the BABELnot project (Lister et al, 2012), a principal goal of which is to explore a possible approach for the development and assessment of learning standards in programming courses. Formal written examinations are a common form of assessment in programming courses, and typically the form to which most marks are attached. The approach we have taken is to analyse examination papers to investigate levels of, and variations in, assessment of learning outcomes across institutions. In prior work (Simon, Sheard, Carbone, Chinn, et al, 2012) we analysed programming exam papers to identify the range of topics covered and the skills and knowledge required to answer exam questions in introductory programming. Here we extend that approach to determine the complexity and difficulty of exam questions and the level of knowledge required to answer them correctly. Our approach is similar to that taken by Crisp et al (2012), who explored the types of assessment tasks used to assess graduate attributes.

In this study we analyse exam questions to determine the levels of difficulty and complexity of the exams, which leads to an understanding of the standards being assessed. We first explore the concepts of task complexity and difficulty. We then develop a framework to measure the complexity of programming exam questions from which we can infer the level of achievement we expect from our programming students. Next, we apply this to a set of programming exam papers from multiple institutions to compare the levels of knowledge and skills being assessed.

2 Task Complexity and Task Difficulty

In a comprehensive review of the literature on the concept of task complexity, Campbell (1988) proposed that task complexity can be defined objectively as a function of task attributes that place high cognitive demands on the performer of the task. Braarud (2001) further distinguished between the *objective* task complexity, which is a characteristic of the task itself, and *subjective* task complexity, which is the user's perception of the complexity of a task. Both Campbell and Braarud argued that task *difficulty* is distinct from task complexity, incorporating additional aspects – such as the task context – that can entail high effort in doing a task. Campbell (1988) proposed that complex tasks are often ill-structured and ambiguous. He observed that while complex tasks are necessarily difficult, difficult tasks are not necessarily complex. For example, tracing a path through a maze with a pencil can be quite complex, but is seldom difficult.

Complexity is clearly a key concept for determining the difficulty of a task, but there seems to be little consensus amongst researchers about what attributes can be used to determine the complexity of a task. Campbell (1988) proposed four properties that influence task complexity and used these to develop a task typology. Mennin (2007) distinguished between simple, complicated and complex problems, but did not explain the distinction, instead using examples to illustrate the categories. Haerem and Rau (2007) developed an instrument to measure variability and analysability, which they suggested are fundamental aspects of complexity. An investigation of task complexity by Stahl, Pieschl and Bromme (2006) used Bloom's taxonomy to classify tasks of different levels of complexity. They further classified according to level of difficulty within these tasks, but did not define what they meant by this.

Williams and Clarke (1997) completed the most comprehensive work in this area. They proposed six dimensions of complexity (linguistic, contextual, representational, operational, conceptual and intellectual) and applied these to problems in the mathematics domain. Carbone (2007) later applied these six dimensions to tasks in the computer programming domain.

3 Exam Question Complexity

In our work we wished to investigate the level of difficulty of exam questions as a means to assess the level of skills and knowledge being tested in introductory programming courses. A search of the literature on programming exam questions indicated a number of factors that contribute to the complexity and hence difficulty of these assessment tasks.

A common factor identified was the *cognitive load* placed on the student by the question, which is defined as the number of discrete pieces of information that the student is required to understand in order to answer the question (Sweller, 1988). An investigation of second-year data structures exam questions by Simon et al (2010) proposed that the phrasing and construction of a question can add to cognitive load and therefore increase the difficulty of a question. They argued that cognitive load also increases when questions involve multiple concepts. In a review of 15 introductory programming exams from 14 schools, Petersen et al (2011) investigated the content and concepts covered by each question, proposing that the more concepts the students need to deal with to answer a question, the higher the cognitive load and hence the difficulty of the question. They assessed cognitive load simply by counting the distinct concepts dealt with in a question, without considering whether different concepts might have different intrinsic levels of difficulty. They found that code-writing questions had the highest number of concepts per question. In a study of data structures exams, Morrison et al (2011) found few long questions, and proposed that this was due to the exam setters wishing to avoid the increased cognitive load that would come with extra length.

The conceptual level of topics covered by a question has also been proposed as an influence on question difficulty. A survey by Schulte and Bennedsen (2006) gathered 242 academics' opinions of the difficulty of CS1 topics. The topics found most difficult were design,

Measure	Focus*	Classification values	Description
External domain references	Q only	low, medium, high; if medium or high, the external domain is specified	Reference to a domain beyond what one would reasonably expect introductory programming students to know
Explicitness	Q only	high, medium, low (note order of levels)	Extent of prescriptiveness as to how to answer the question
Linguistic complexity	Q only	low, medium, high	Length and sophistication of the natural language used to specify the question
Conceptual complexity	Q & A	low, medium, high	Classification of the individual programming concepts required to answer the question
Code length	Q & A	low, medium, high, NA	Whether code is up to half a dozen lines long, up to two dozen lines long, or longer
Intellectual complexity (Bloom level)	Q & A	knowledge, comprehension, application, analysis, evaluation, synthesis	Bloom's taxonomy as applied to programming questions by Gluga et al (2012)
Level of difficulty	Q & A	low, medium, high	Subjective assessment of difficulty of question

* The second column, Focus, indicates whether the measures apply only to the question or to the question and answer.

Table 1: Six complexity measures and level of difficulty used to classify exam questions

recursion, advanced OO topics (polymorphism & inheritance) and pointers & references. This aligns well with a survey of 35 academics by Dale (2006) which showed that design, problem solving, control structures, I/O, parameters, recursion, and OO concepts were seen as the difficult concepts for novice programming students.

Based on a detailed statistical analysis of student answers to introductory programming exam questions, Lopez et al (2008) proposed a hierarchy of programming-related skills. In an attempt to interpret results that were not intuitively obvious they concluded that there were characteristics of a task other than its style that could explain its level of difficulty. They proposed that the size of the task and the programming constructs used also influenced the difficulty of a question.

A corpus of work has used Bloom's taxonomy (Anderson & Sosniak, 1994) to classify questions according to the cognitive demand of answering them (Thompson et al, 2008); or the SOLO taxonomy (Hattie & Purdie, 1998) to classify the intellectual level demonstrated by answers to questions (Clear et al, 2008; Sheard et al, 2008).

From a different perspective, the study of engineering exam questions by Goldfinch et al (2008) concluded that the style and structure of questions influenced perceptions of difficulty.

4 Classifying Exam Question Complexity

In the studies of programming exam questions that we have reviewed, the assessments of question difficulty were impressionistic; however, the reasons given for difficulty usually pointed to specific aspects of complexity. Some of these related to the question itself, some to what was required as a response. We considered that complexity could be inherent both in the question and in the response to the question. This led us to propose a framework for assessing the aspects of complexity of exam questions which could then be used to identify areas of difficulty for the student.

To determine the factors that influence complexity we considered four perspectives.

1. How is the question asked? How readily will the students be able to understand what the question is asking them to do? Question phrasing or style can lead to ambiguity and uncertainty in how to respond (Goldfinch et al, 2008; Simon et al, 2010). To address these questions we consider the *linguistic* complexity of the question and references to *external domains* beyond the scope of the course, which we called 'cultural references' in our previous work (Sheard et al, 2011).
2. How much guidance does the question give as to how it should be answered (Goldfinch et al, 2008; Simon et al, 2010)? Here we consider the *explicitness* of the question.
3. What is the student required to do in order to answer the question? Here we consider the *amount of code* to be read and/or written and the *intellectual complexity* level demanded (Lopez et al, 2008; Morrison et al, 2011; Petersen et al, 2011).
4. What does the student need to understand in order to answer the question? This relates to the number of concepts involved in the question and to their intrinsic complexity. Here we consider the *conceptual* complexity (Dale, 2006; Schulte & Bennedsen, 2006).

The aspects of complexity highlighted by these questions led to the development of an exam question complexity classification scheme, a framework for determining the levels of complexity of a question. There are six dimensions to the scheme, as shown in the first six rows of Table 1. The first three are concerned with the exam question alone and the next three are concerned with the question and answer combined. For example, linguistic complexity applies to the language in which the question is expressed, while code length assesses the combined length of any code provided in the question and any code that the student is required to write in the answer. Four of the six measures of complexity are closely aligned to the dimensions of Williams and Clarke (1997). These are external domain references and linguistic, conceptual and intellectual complexities. The last row of the table shows the measure of level of difficulty, which we consider to be distinct from the various measures of complexity.

For each measure, the possible classification values and a brief description of the measure are given. More detailed explanations of the complexity measures are given in the results section.

5 Research Approach

The first version of the exam question complexity classification scheme emerged from a brainstorming session that was framed by the perspectives listed in Section 4 and informed by the literature discussed in the preceding sections. This was followed by a number of iterations in which about a dozen researchers applied the measures to the questions in an exam. After each round of classification the measures were clarified and adjusted as appropriate until the classification scheme appeared to have stabilised.

At that point an inter-rater reliability test was conducted on the complexity measures, with all researchers classifying all 37 questions in a single exam, first individually and then in pairs. As all of the measures are ordinal, reliability was calculated using the Intraclass Correlation (ICC) (Banerjee et al, 1999), and was found to be satisfactory on all measures, with pairs proving distinctly more reliable than individuals.

Following this test, the remaining exams were classified by pairs of researchers, who first classified the questions individually and then discussed their classifications and reconciled any differences.

6 Results

This section presents the results of analysing 20 introductory programming exam papers using the complexity measures listed in Table 1. A total of 472 questions were identified in these exams, with the number of questions in an exam ranging from four to 41.

The 20 exam papers were sourced from ten institutions in five countries. All were used in introductory programming courses, 18 at the undergraduate level and two at the postgraduate level. The latter two courses are effectively the same as courses taught to first-year undergraduate students, but are taught to students who are taking a postgraduate computing qualification to supplement a degree in some unrelated area. Course demographics varied from 25 students on a single campus to 800 students over six campuses, two of these being overseas campuses. Most courses used Java with a variety of IDEs (BlueJ, JCreator, Netbeans, Eclipse), two used JavaScript, one used C# with Visual Studio, one used Visual Basic, one used VBA (Visual Basic for Applications), and one used Python.

Most of the exams were entirely written, but two were separated into a written part and a computer-based part, each worth 50% of the complete exam.

Note that for the analysis, the percentage mark allocated to each question has been used as a weighting factor for the other measures.

6.1 Overall Complexity Measures

Each question was classified according to six measures of complexity.

The results for five levels of complexity (external domain references, explicitness, linguistic complexity, conceptual complexity and code length) are summarised in Table 2. Because the percentage mark allocated to each question was used as a weighting, the figures in the table represent the percentage of the exam marks allocated, not the percentage of the number of questions.

Measure of complexity	low	medium	high
External domain references	**95%**	5%	0%
Explicitness	3%	30%	**67%**
Linguistic complexity	**80%**	17%	3%
Conceptual complexity	8%	**67%**	25%
Code length*	27%	**54%**	10%

* 9% of questions (weighted) did not involve code

Table 2: Overall levels of complexity of questions from the 20 exams, with mode values shown in bold

While these five measures are all classified as low, medium, or high, intellectual complexity is classified according to Bloom's six-point scale, so its classifications are shown separately in Figure 1. These classifications ranged from 3% for Evaluation to 44% for Application.

Considering the mode values, we can see that, over all the exams, questions are predominantly low in external domain references, highly explicit, low in linguistic complexity, of medium conceptual complexity and medium code length, and at the Application level of Bloom's taxonomy.

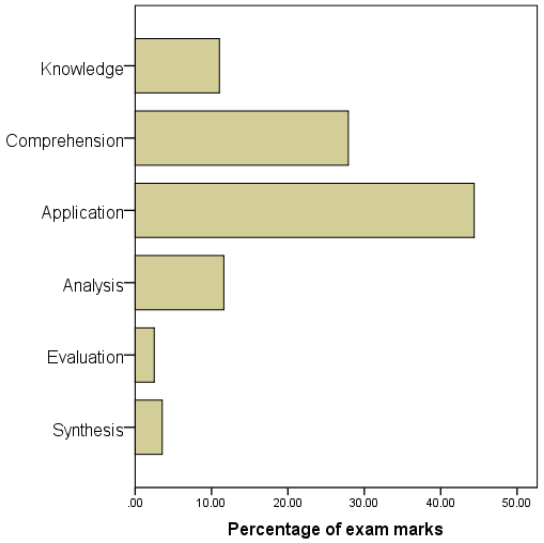

Figure 1: Overall measure of intellectual complexity (Bloom's taxonomy)

6.2 External Domain References

Many exam questions involve some sort of scenario, referring to a domain beyond what would necessarily be taught in the programming course. These scenarios have the potential to make a question more complex.

An external domain reference is any use of terms, activities, or scenarios that may be specific to a particular group and may reduce the ability of those outside the group to understand the question. For example, if a question refers to the scoring scheme of Australian Rules football, students would require specific knowledge to fully understand it – unless the question explicitly includes all of the knowledge that is needed to deduce the answer. Another question might display a partly complete backgammon game and ask students to write a program to determine the probability of winning on the next throw

of the dice. Unless the question fully explains the relevant rules, students who do not know backgammon will clearly not be able to answer it.

Programming knowledge does not constitute an external domain reference, because it is assumed to be taught in the course or prior courses. General knowledge is not considered as an external domain reference so long as the classifier is confident that it really is general: that all introductory programming students could reasonably be expected to know it.

We classify external domain references as high if students cannot understand the question without knowing more about an external domain; medium if they are given all the information they need, but the wording might lead them to think otherwise; and low if all students should be able to understand the question as it is.

None of the questions that we analysed relied upon a high level of knowledge from an external domain. Only seven exams contained questions with a medium level of external domain knowledge; these comprised at most 20% of any exam, and made up only 23 questions (less than 1% of the 472 questions).

Of those 23 questions, a few assumed some knowledge of the business domain (interest, profit, taxes), and a couple assumed knowledge of mathematical concepts (complex numbers, log arithmetic) or scientific concepts (storm strength). A few questions assumed knowledge of computing concepts (codes/encryption, pixellation, domain name format) beyond what would be considered reasonable for an introductory programming student. Some references were culturally based (name format, motel, vehicle registration, sports, card games). One referred to a sorting hat, a concept from a popular series of books and films. All of these references were at the medium level, so students did not need the external domain knowledge in order to answer the questions.

Of particular interest are questions that refer to external domain knowledge but make it clear that this knowledge was covered thoroughly during the course, perhaps being the subject of a major assignment. Such questions would not constitute external domain knowledge for this particular cohort of students. However, if the question were to be placed in a repository for the use of other academics, it would be wise to flag that there are external domain references for other students; therefore such references were classified as requiring external domain knowledge, with the domain in question being specified as the course assignment.

6.3 Explicitness

How strongly does the question tell the student what steps to use in writing an answer? How strongly does it prescribe, for example, what programming constructs and/or data structures to use? There is a fairly high level of explicitness in "Write a method that takes an array of integers as a parameter and returns the sum of the numbers in the array". There is a very low level of explicitness in "Write a program to simulate an automatic vending machine," which requires the students to determine the purchase process of the vending machine and identify the corresponding programming operations. Another question might require students to specify a Card class to use in a card game program. A highly explicit

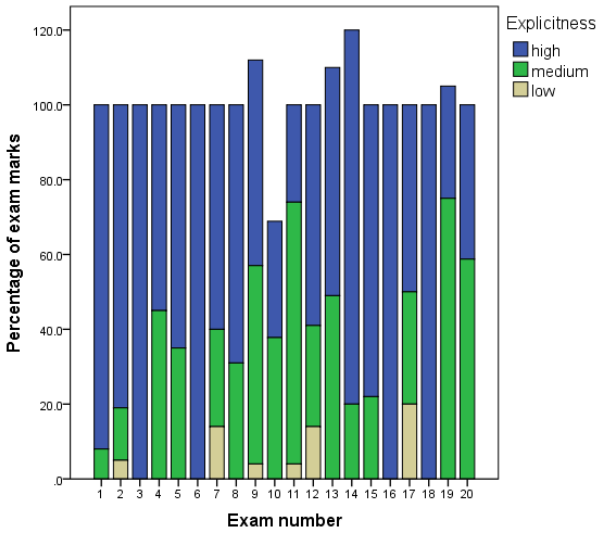

Figure 2: Explicitness of questions in each exam

version would list the methods required and the attributes and their types. A version with a low level of explicitness would not specify methods, attributes, or operations. A version with medium explicitness would perhaps specify some of the attributes and/or methods and require the student to deduce the rest. Note that the level of explicitness of a question would be expected to have an inverse relationship with the question's difficulty; that is, the more explicit a question, the easier we would expect students to find it.

Figure 2 shows a summary of the explicitness levels of questions over the 20 exams. Two thirds of the marks (67%) were allocated to questions expressed with a high level of explicitness. The graph shows that less than a third of the exams (6) contained questions of low explicitness, and the marks allocated for these questions comprised 20% or less of these exams. In most of the exams more than half the questions were highly explicit.

Note: in Figure 2, and those following, the four exams that exceed 100% do so because they include some choice, so students do not have to complete all questions to score 100%. The exam that is less than 100% is from a course that included non-programming topics and has questions that do not relate to programming. We classified only the programming-related questions in this exam.

6.4 Linguistic Complexity

Linguistic complexity is related to the length and sophistication of the natural language used to specify the question. Some questions have lengthy descriptions or use unusual words, which could affect the ability of a student to answer them. One possible view of linguistic complexity is that it is an approximation of the likelihood that a student not fluent in the natural language of the question would have trouble understanding the question.

Overall, most marks were allocated to questions involving a low level of linguistic complexity (80%). In about a third of the exams (7), all questions were classified as having a low level of linguistic complexity. Only two exams had questions with high linguistic complexity. In one of these, the high linguistic complexity was in a single question, comprising 50% of the exam, which was to be answered at the computer.

6.5 Conceptual Complexity

Questions in programming exams usually require students to understand a number of different ideas or concepts. On the basis of our own experience of teaching introductory programming, and of other people's survey findings (Dale, 2006; Schulte & Bennedsen, 2006), we have classified a number of programming concepts as being of low, medium, or high conceptual complexity. For example, variables and arithmetic operators are of low conceptual complexity; methods, and events are of medium conceptual complexity; and recursion, file I/O, and arrays of objects are of high conceptual complexity. Note that these levels were defined specifically for Java-like procedural or object-oriented introductory programming courses; when we come to classify exams in courses that use functional programming, we might need to redefine them, with recursion, for example, shifting from high to a lower conceptual complexity.

We classified the questions using the initial categorisation as a guide, while remaining conscious that particular usage might affect the classification. For example, although loops are generally classified as medium, a classifier could argue for high complexity when classifying a particularly tortuous loop.

The conceptual complexity findings are summarised in Figure 3. Overall, two thirds of the marks (67%) were allocated to questions involving a medium level of conceptual complexity. Most exams showed a range of conceptual complexity, with the majority of marks allocated to questions involving concepts of medium complexity. Only four exams had no questions of high conceptual complexity, and only four had no questions of low conceptual complexity.

6.6 Code Length

The questions were classified according to the amount of code involved in reading and answering the question, with a simple guide that up to about half a dozen lines of code would be considered low, between there and about two dozen lines would be considered medium, and any more than about two dozen lines of code would be considered high. A summary of the results is shown in Figure 4. Overall, more than half the marks (54%) were allocated to questions involving a medium amount of code. About two thirds of the exams (14) contained questions that did not involve code; however, these were usually only a small component of the exam. Most of the marks in most of the exams were allocated to questions involving low and medium code length. Less than half the exams (8) had questions involving large amounts of code, and only three exams had large weightings of marks (more than 40%) involving high code length.

6.7 Intellectual Complexity

Bloom's cognitive domain is a long-recognised measure of the intellectual complexity of a question in terms of its expected answer. There has been debate about whether Bloom's domains can be usefully applied to programming questions, but there is some consensus that they can (Thompson et al, 2008). Gluga et al (2012) provide a clear explication, with examples and a tutorial, of one way of doing this.

The summary in Figure 5 shows a great variation in levels of intellectual complexity across the exams. Considering the three lowest levels of intellectual complexity, most exams (17) contained questions at the Knowledge level, all exams contained questions at the Comprehension level, and all but one exam contained questions at the Application level. Questions at the Analysis level were found in just over half the exams (12). At the highest Bloom levels there were very few questions, with only one exam containing Evaluation level questions and three exams containing Synthesis level questions.

6.8 Degree of Difficulty

Assessing the degree of difficulty entails classifying a question according how difficult an average student at the end of an introductory programming course is likely to find it. This is a holistic measure. We would expect there to be a correlation between question difficulty and students' marks on the question: the higher the difficulty, the lower the average mark we might expect students to attain.

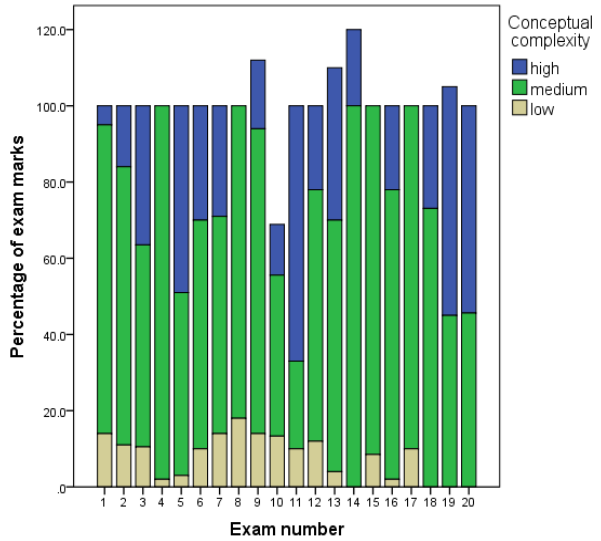

Figure 3: Conceptual complexity of questions in each exam

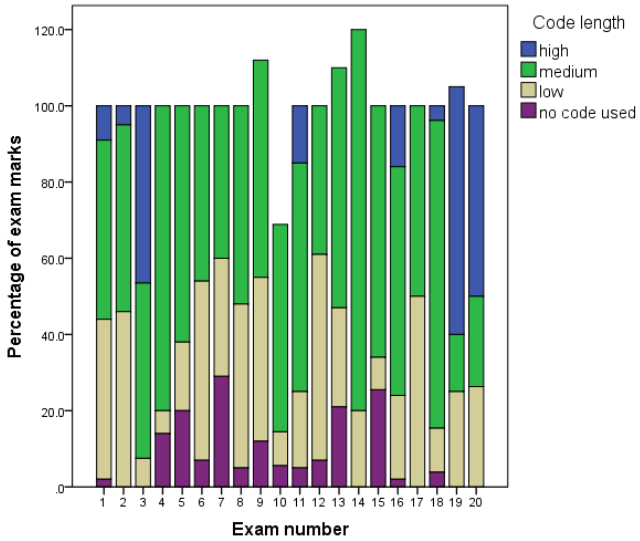

Figure 4: Length of code involved in questions in each exam

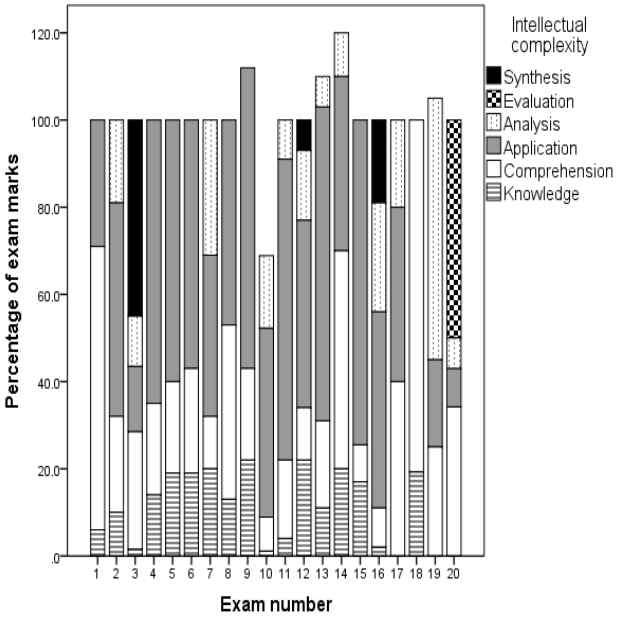

Figure 5: Intellectual complexity of questions in each exam

Question difficulty assesses the student's ability to manage all of the demands of a task. It is concerned with the student's perception of and response to the question, whereas task complexity is static and defined by the nature of the question itself.

The questions were classified according to the perceived level of difficulty for a student at the end of an introductory programming course. Overall, half the marks (50%) were allocated to questions rated as medium, with 30% for questions rated as low difficulty and 20% for question rated as high difficulty. The summary in Figure 6 shows the variation across the exams. Although some exams have a fairly wide spread of low/medium/high difficulty questions, about a third (7) of the exams have no high difficulty questions, and one exam has high difficulty only in a bonus question. One exam had no questions of low difficulty, just 45% medium and 55% high.

We have been asked why we bother to subjectively assess question difficulty when the students' marks on the questions would provide a more reliable measure. The answer is simple. We were fortunate enough to have been provided with these 20 exams to analyse. It would be too much to have also asked for student performance data on each question of each exam. First, it is possible that for many of the exams the only data now available is students' overall marks in the course, and perhaps even that is no longer available. Second, ethics approval is required before students' results can be analysed for research purposes, and we did not feel it appropriate to ask everyone who gave us an exam to follow this up by applying for ethics approval in order to give us their students' results as well.

However, we have conducted a separate study (Simon, Sheard, Carbone, D'Souza, et al, 2012), on a set of questions for which we do have access to student performance data, and have confirmed the expected link between our assessment of question difficulty and the students' performance on the same questions.

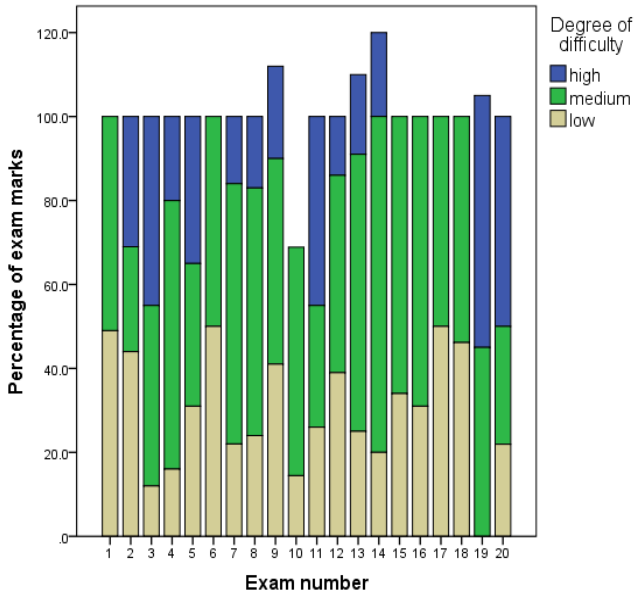

Figure 6: Degree of difficulty of questions in each exam

6.9 Relationship between Complexity and Difficulty

Each of the measures of complexity focuses on particular characteristics of a question which could be seen to contribute to an overall complexity for the question, whereas the degree of difficulty is a perception of how difficult the average student at the end of the introductory programming course would find the question. A correlation test was performed to explore the relationship between complexity and difficulty. As the measures of complexity and difficulty are at the ordinal level, a Spearman's Rank order correlation was used. The results, summarised in Table 3, show relationships between the degree of difficulty and each measure of complexity, all of which are significant at $p < 0.01$. The strongest relationships with degree of difficulty are code length and intellectual complexity: questions involving more program code, and questions at the higher levels of Bloom's taxonomy, are more difficult questions.

To further explore the relationship between complexity and difficulty, the levels of difficulty within each complexity measure were determined. The following results were found.

- Most questions with a low level of difficulty are highly explicit (97%).
- Most questions with a low level of difficulty are of low linguistic complexity (98%).

	Degree of difficulty (r)
External domain references	0.197 *
Explicitness	-0.408 *
Linguistic complexity	0.326 *
Conceptual complexity	0.412 *
Code length	0.564 *
Intellectual complexity	0.501 *

* all significant at p <0.01

Table 3: Relationship between degree of difficulty and complexity measures

- No questions with a high level of difficulty have a low conceptual complexity.
- No questions with a high level of difficulty involve a low amount of code or no code.
- No questions with low level of difficulty involve a high amount of code.
- No questions with a high level of difficulty are classified at the two lowest levels of Bloom's cognitive domain (Knowledge and Comprehension). Figure 7 shows the breakdown of the degree of difficulty within each level of Bloom's taxonomy.

7 Discussion

We have found from our analysis that there is wide variation in the final examinations of introductory programming courses, with variations in all the complexity measures and in the level of difficulty. As the pressure grows to determine standards of assessment for university courses, the framework that we have devised is likely to prove extremely helpful. We do not propose that all introductory programming courses should be identical, or that they should all assess at the same level; what we do propose is that there should be a means to determine the extent of similarity between the courses and their outcomes, a means to compare the levels at which they assess their students. As we see it, the push to standardise is not an attempt to impose uniformity but a desire to be explicitly aware of the spread and variety of what is taught and assessed.

An interesting consequence of our findings is that, notwithstanding their substantial overlap, different introductory programming courses do assess somewhat different material at somewhat different levels. Students migrating between programs, and academics charged with assigning credit on the basis of courses completed elsewhere, would do well to be aware of this.

Of the complexity measures addressed in this work, it is useful to distinguish between 'good' complexity and 'bad' complexity. High-level external domain references and high linguistic complexity can be undesirable, and we were pleased to see little evidence of those in the exams we assessed. Conceptual and intellectual complexity can be intentional and purposeful, and it seems quite reasonable to test these to some extent – through there is still an open question as to what levels we can reasonably expect students to attain in an introductory programming course.

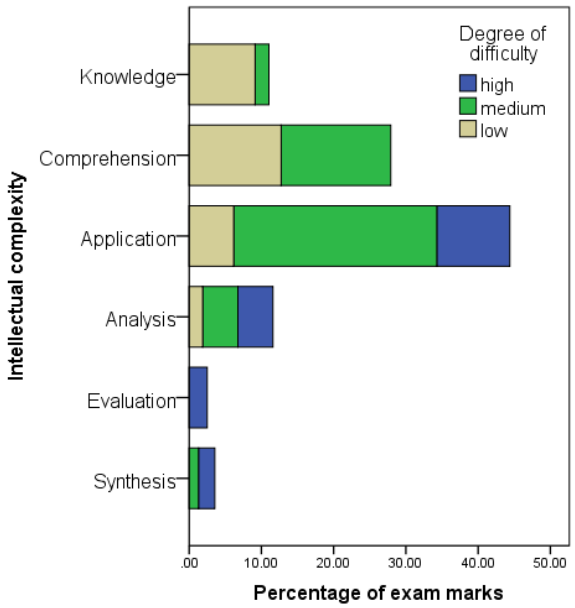

Figure 7: Degree of difficulty of questions within each level of intellectual complexity

With regard to intellectual complexity, academics from other disciplines might be surprised to see such a preponderance of questions at the Application level in the exam for an introductory course. In other disciplines it might be expected that the first course will deal more with Knowledge and Comprehension, with the higher levels of the taxonomy reserved for higher-level courses. If this is indeed the case, we need to be confident that this high level of Application is a necessary consequence of the nature of teaching programming; the alternative is to recognise that we are asking too much sophistication of students in our introductory courses.

Does it help students or hinder them to have a practical, computer-based exam? Is it more acceptable in a computer-based exam than a paper-based exam to have a large question, worth 50% of the exam, that has high linguistic complexity, high conceptual complexity, high code length, a Bloom level of Evaluation, and a high perceived overall difficulty? We do not propose answers to these questions. Rather, we note that they have emerged from our study of these exams, and that they are worthy of consideration by the computing education community.

The assignment of topics to low, medium, and high conceptual complexity, while certainly not arbitrary, is clearly open to debate. The choices appear to have been reasonable, given the correlation between this measure and the overall question difficulty. However, we need to consider whether conceptual complexity is an intrinsic feature of a topic, or more a function of what was taught and how it was taught in each course. Just as the conceptual complexity of recursion might be high in a procedural programming course and low in a functional programming course, might it be the case that the conceptual complexity of any topic is dependent on when and how that topic was taught in each specific course? We also note in passing that while selection and iteration appear as topics in the surveys on which our own lists of topics were based, the topic of sequence is notable by its absence, although it has been identified by Simon (2011) and others as a topic that some students have difficulty grasping. In retrospect, we accept that it would have been wise to list sequence as a topic, assigning it a low level of conceptual complexity.

With regard to the various complexity measures described in this paper, is it possible and reasonable to suggest what mix of low, medium, and high values should normally be found in an introductory programming exam? Can we use these measures to suggest that particular exams are inappropriately complex or inappropriately simple? Or do we accept that there is a wide variety in the courses themselves, and simply note where each exam fits into the broader picture?

For some of the measures it is possible to make clear recommendations to the people who write exams. It

would appear reasonable to expect exam questions to have low linguistic complexity and not to rely on students' knowledge of domains outside what is being taught. For other measures the choice is more personal. For example, some examiners might like to be entirely explicit about what students are required to do, while others might prefer to test the students' problem-solving abilities by framing some questions with low explicitness and leaving the students to fill in the gaps in the specifications. However, in a couple of exams we assessed, more than 75% of the questions were of medium level explicitness; would most examiners consider this a little high for an introductory programming exam?

The analysis reported in this paper is exploratory: its purpose is as much to identify questions as to answer them. Its contribution is that it raises questions such as those discussed above, at the same time providing a framework in which the questions can be discussed, and possibly, eventually, answered.

8 Conclusions and Future Work

In this study we analysed programming examination papers across institutions, both national and international, as a window into the levels of learning expected in foundation programming courses. The complexity measures applied in this study highlight the variability of introductory programming exams. This could be taken as reinforcing the suggestion that exams are highly personal; but it leaves open the question: are the exams all assessing the same or comparable things? If not, can we be sure that each and every one of these exams is a valid assessment instrument?

Future work will include exploring the thinking of the people who write the exams, and whether they do so with any awareness of the sorts of issue addressed in this analysis. This will entail interviewing a number of exam writers and conducting a qualitative analysis of the interview transcripts.

In addition, we intend to analyse a number of introductory programming exams that use functional programming, and to extend our analysis to the exams for second- and third-level programming courses.

With regard to the increasing role of standardisation, further aspects of the BABELnot project (Lister et al, 2012) include the establishment of a repository of fully classified programming exam questions with accompanying performance data, and the benchmarking of a subset of these questions across multiple institutions.

9 Acknowledgements

The authors would like to thank the Learning and Teaching Academy of the Australian Council of Deans of ICT for its support of the ACE 2012 workshop, and also the Australian Federal Government's Office for Learning and Teaching for its support of the BABELnot project.

10 References

Anderson, L. W. and 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.), *Bloom's Taxonomy: A Forty Year Retrospective* (pp. 9-27). Chicago, Illinois, USA: The University of Chicago Press.

Australian Council for Educational Research. (2011): Assessment of Higher Education Learning Outcomes (AHELO) Retrieved 24 August, 2012, from http://www.acer.edu.au/aheloau

Banerjee, M., Capozzoli, M., McSweeney, L. and Sinha, D. (1999): Beyond kappa: a review of interrater agreement measures. *Canadian Journal of Statistics*, **27**, 3-23.

Bexley, E., James, R. and Arkoudis, S. (2011): The Australian academic profession in transition. Canberra: Department of Education, Employment and Workplace Relations, Commonwealth of Australia.

Braarud, P. (2001): Subjective task complexity and subjective workload: Criterion validity for complex team tasks. *International Journal of Cognitive Ergonomics*, **5**(3), 261-273.

Cameron, I. and Hadgraft, R. (2010): Engineering and ICT Learning and Teaching Academic Standards Statement. Strawberry Hills, NSW, Australia: Australian Learning and Teaching Council.

Campbell, D. (1988): Task complexity: A review and analysis. *Academy of Management Review*, **13**(1), 40-52.

Carbone, A. (2007): *Principles for Designing Programming Tasks: How task characteristics influence student learning of programming.* PhD, Monash University, Melbourne, Australia.

Clear, T., Whalley, J., Lister, R., Carbone, A., Hu, M., Sheard, J., Simon, B. and Thompson, E. (2008): *Reliably classifying novice programmer exam response using the SOLO taxonomy.* Paper presented at the 21st Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ 2008), Auckland, New Zealand.

Coates, H. and Edwards, D. (2008): The 2008 Graduate Pathways Survey. Canberra: Department of Education, Employment and Workplace Relations, Commonwealth of Australia.

Crisp, G., Barrie, S., Hughes, C. and Bennison, A. (2012): *How can I tell if I am assessing learning outcomes appropriately?* Paper presented at the Higher Education Research and Development Society of Australasia (HERDSA), Macquarie Hotel, Hobart. http://conference.herdsa.org.au/2012/

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

Evans, C. (2011): Professor Alan Robson to take on key higher education quality role *Media Release* Retrieved 24 August, 2012, from http://ministers.deewr.gov.au/evans/professor-alan-robson-take-key-higher-education-quality-role

Gluga, R., Kay, J., Lister, R., Kleitman, S. and Lever, T. (2012): *Coming to terms with Bloom: An online tutorial for teachers of programming fundamentals.* Paper presented at the 14th Australasian Computing Education conference, Melbourne, Australia.

Goldfinch, T., Carew, A. L., Gardner, A., Henderson, A., McCarthy, T. and 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.

Haerem, T. and Rau, D. (2007): The influence of degree of expertise and objective task complexity on perceived task complexity and performance. *Journal of Applied Psychology*, **92**(5), 1320-1331.

Hattie, J. and Purdie, N. (1998): The SOLO model: Addressing fundamental measurement issues. In M. Turpin (Ed.), *Teaching and Learning in Higher Education* (pp. 145-176). Camberwell, Victoria, Australia: ACER Press.

Lister, R., Corney, M., Curran, J., D'Souza, D., Fidge, C., Gluga, R., Hamilton, M., Harland, J., Hogan, J., Kay, J., Murphy, T., Roggenkamp, M., Sheard, J., Simon and Teague, D. (2012): *Toward a shared understanding of competency in programming: An invitation to the BABELnot project.* Paper presented at the 14th Australasian Computing Education conference, Melbourne, Australia.

Lopez, M., Whalley, J., Robbins, P. and Lister, R. (2008): *Relationships between reading, tracing and writing skills in introductory programming.* Paper presented at the Fourth International Computing Education Research workshop (ICER 2008), Sydney, Australia.

Mennin, S. (2007): Small-group problem-based learning as a complex adaptive system. *Teaching and Teacher Education*, **23**, 303-313.

Morrison, B., Clancy, M., McCartney, R., Richards, B. and Sanders, K. (2011): *Applying data structures in exams.* Paper presented at the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11), Dallas, Texas, USA.

Petersen, A., Craig, M. and Zingaro, D. (2011): *Reviewing CS1 exam question content.* Paper presented at the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11), Dallas, Texas, USA.

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

Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E. and Whalley, J. (2008): *Going SOLO to assess novice programmers.* Paper presented at the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'08), Madrid, Spain.

Sheard, J., Simon, Carbone, A., Chinn, D., Laakso, M.-J., Clear, T., de Raadt, M., D'Souza, D., Harland, D., Lister, R., Philpott, A. and Warburton, G. (2011): *Exploring programming assessment instruments: a classification scheme for examination questions.* Paper presented at the Seventh International Computing Education Research workshop (ICER 2011), Providence, Rhode Island, USA.

Simon (2011): *Assignment and sequence: why some students can't recognise a simple swap.* Paper presented at the 10th Koli Calling International Conference on Computing Eduction research, Finland.

Simon, Sheard, J., Carbone, A., Chinn, D., Laakso, M.-J., Clear, T., de Raadt, M., D'Souza, D., Lister, R., Philpott, A., Skene, J. and Warburton, G. (2012): *Introductory programming: Examining the exams.* Paper presented at the 14th Australasian Computing Education conference, Melbourne, Australia.

Simon, Sheard, J., Carbone, A., D'Souza, D., Harland, J. and Laakso, M.-J. (2012): *Can computing academics assess the difficulty of programming examination questions?* Paper presented at the 11th Koli Calling International Conference on Computing Education Research, Finland.

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

Stahl, E., Pieschl, S. and Bromme, R. (2006): Task complexity, epistemological beliefs and metacognitive calibration: An exploratory study. *Journal of Educational Computing Research*, **35**(4), 319-338.

Sweller, J. (1988): Cognitive load during problem solving. Effects on learning. *Cognitive Science*, **12**(2), 257-285.

Tertiary Education Quality and Standards Agency. (2012): Higher Education Standards Framework, from http://www.teqsa.gov.au/higher-education-standards-framework

Thompson, E., Luxton-Reilly, A., Whalley, J., Hu, M. and Robbins, P. (2008): *Bloom's taxonomy for CS assessment.* Paper presented at the 10th Australasian Computing Education Conference (ACE 2008), Wollongong, Australia.

Williams, G. and 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.