Introductory programming: examining the exams

Simon
University of Newcastle
simon@newcastle.edu.au

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

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

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

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

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

Michael de Raadt
Moodle
michaeld@moodle.com

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

Raymond Lister
University of Technology Sydney
raymond.lister@uts.edu.au

Anne Philpott
Auckland University of Technology
aphilpot@aut.ac.nz

James Skene
Auckland University of Technology
james.skene@aut.ac.nz

Geoff Warburton
Australia
geoffw173@gmail.com

Abstract

This paper describes a classification scheme that can be used to investigate the characteristics of introductory programming examinations. The scheme itself is described and its categories explained. We describe in detail the process of determining the level of agreement among classifiers, that is, the inter-rater reliability of the scheme, and we report the results of applying the classification scheme to 20 introductory programming examinations. We find that introductory programming examinations vary greatly in the coverage of topics, question styles, skill required to answer questions and the level of difficulty of questions. This study is part of a project that aims to investigate the nature and composition of formal examination instruments used in the summative assessment of introductory programming students, and the pedagogical intentions of the educators who construct these instruments.

Keywords: examination papers, computing education, introductory programming

1 Introduction

There are several common forms of assessment in introductory programming courses. In-class computer-based tests and programming assignments are good ways of assessing the interactive skill of designing and writing computer programs. Written quizzes and examinations are appropriate for assessing students' familiarity with relevant theoretical knowledge. In addition, written tests can examine some aspects of program designing and coding, although they are perhaps not ideally suited for the assessment of these skills.

Formal examinations are widely used in the summative assessment of students in programming courses. Writing an examination paper is an important task, as the exam is used both to measure the students' knowledge and skill at the end of the course and to grade and rank students. Yet it is often a highly individual task, guided by the whims, preferences, beliefs, and perhaps inertia of the examiner. Lister (2008) observes that there is a great deal of 'folk pedagogy' in computing education, and acknowledges that his early examinations were based upon folk-pedagogic misconceptions.

In constructing an exam, educators must consider what they wish to assess in terms of the course content. They must consider the expected standards of their course and decide upon the level of difficulty of the questions. Elliott Tew (2010) suggests that "the field of computing lacks valid and reliable assessment instruments for pedagogical or research purposes" (p.xiii). If she is right, and the instruments we are using are neither valid nor reliable, how can we make any credible use of the results?

An analysis of research papers about programming education published in computing education conferences from 2005 to 2008 found that 42% of the studies gathered data from formal exam assessment (Sheard, Simon, Hamilton & Lönnberg 2009). It seems critical that we understand the nature of these assessment instruments. Lister (2008) urges computing educators to base their decisions upon evidence. At least part of the relevant evidence should be an overview of introductory programming exams as a whole, and we have therefore set out to examine the examinations in introductory programming courses.

In this paper we describe an exam question classification scheme that can be used to determine the content and nature of introductory programming exams. We apply this instrument to a set of exam papers and describe the process of establishing a satisfactory inter-rater reliability for the classifications. We report what we have found about the content and nature of the exam papers under consideration. This study is the first step of large-scale investigation of the nature and composition of

formal examinations used in introductory programming courses, and the pedagogical intentions of the educators who write and use these examinations.

2 Background

Assessment is a critical component of our work as educators. Formative assessment is a valuable tool in helping students to understand what they have achieved and in guiding them to further achievement. Summative assessment is a tool used to determine and report the students' achievement, typically at the end of a course, and to rank the students who have completed the course. When we write and mark summative assessment instruments we are standing in judgment on our students. Yet concern has been expressed that very little work has gone into understanding the nature of the instruments that we use (Elliott Tew & Guzdial 2010).

A number of research studies have used examination instruments to measure levels of learning and to understand the process of learning. A body of work conducted under the auspices of the BRACElet project has analysed students' responses to examination questions (Clear et al 2008, Lister et al 2010, Lopez et al 2008, Sheard et al 2008, Venables et al 2009). Interest in this work stemmed from earlier studies, such as that of Whalley et al (2006), which attempted to classify responses to examination questions using Bloom's taxonomy (Anderson & Sosniak 1994) and the SOLO taxonomy (Dart & Boulton-Lewis 1998). The BRACElet project has focused on exam questions that concern code tracing, code explaining, and code writing. In an analysis of findings from these studies, Lister (2011) proposes that a neo-Piagetian perspective could prove useful in explaining the programming ability of students; this proposal could well guide future investigations into assessment in programming courses.

Few studies were found that investigated the characteristics of examination papers and the nature of exam questions. A cross-institutional comparative study of four mechanics exams by Goldfinch et al (2008) investigated the range of topics covered and the perceived level of difficulty of exam questions. Within the computing discipline, Simon et al (2010) analysed 76 CS2 examination papers, but considered only particular data structures questions, which made up less than 20% of the marks available in the exams. Their analysis focused on the cognitive skills required and the level of difficulty of the questions. Following this study, a further analysis of 59 CS2 papers in the same dataset (Morrison et al 2011) investigated the range of question styles that can be used to test students' skills in the application of data structures. Petersen et al (2011) analysed 15 CS1 exam papers to determine the concepts and skills covered. They found a high emphasis on code writing questions, but with much variation across the exams in the study. Shuhidan et al (2010) investigated the use of multiple-choice questions in summative assessment of four levels of programming courses (CS0-CS3) and found that the use of these questions remained controversial.

Our study focuses on introductory programming examination papers, developing a classification scheme for the purpose of analysing these papers to give a comprehensive view of the style of questions that make up these instruments.

The study was initiated at a workshop associated with the Fourth International Workshop on Computing Education (ICER 2008). A small group developed the ideas and a provisional classification scheme, which they presented at a subsequent workshop associated with the 13th Australasian Computing Education Conference (ACE 2011). At the second workshop the scheme was trialled on a few exam questions and adjusted in the light of the trial. Full details of the scheme's development are described elsewhere (Sheard et al 2011).

3 The classification scheme

The classification scheme consists of eleven different measures, one of which is administrative and the other ten of which describe features that we believe are useful in trying to form an understanding of an examination.

The remainder of this section briefly describes each of the features, and, where appropriate, the reasons for their inclusion.

Percentage of mark allocated. This is the feature described above as administrative. While it might be inherently useful, for example in noting whether comparable questions are worth comparable marks in different exams, its principal purpose in this scheme is for weighting, determining what proportion of a complete exam covers the mastery of particular topics or skills.

Topics covered. In the classification system used here an exam question is assigned at most three of the following topics: data types & variables, constants, strings, I/O, file I/O, GUI design and implementation, error handling, program design, programming standards, testing, scope (includes visibility), lifetime, OO concepts (includes constructors, classes, objects, polymorphism, object identity, information hiding, encapsulation), assignment, arithmetic operators, relational operators, logical operators, selection, loops, recursion, arrays, collections (other than arrays), methods (includes functions, parameters, procedures and subroutines), parameter passing, operator overloading.

In the list above, topics that follow 'assignment' tend to subsume data types & variables, so any question that is categorised with these later topics need not include data types & variables. Similarly, a topic such as selection or loops usually subsumes operators, and arrays generally subsumes loops. Having assigned one of these broader topics to a question, we would not also assign a topic subsumed by that broader topic.

The list of topics was compiled from a number of different sources, including the computing education literature. Dale (2005, 2006) lists 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 and the perceived difficulty of those topics; Elliott Tew and Guzdial (2010) identified topics by analysing the content of relevant textbooks.

Skill required to answer the question. Some questions can be answered purely by recalling knowledge that has been imparted during the course. Others require the application of different skills: tracing code (which

includes evaluating expressions), explaining code, writing code, modifying code (which includes refactoring or rewriting code), debugging code, designing programs, and testing programs. When classifying a question we require a single skill to be nominated. If a question appears to require two or more skills (for example, designing programs and writing code), we would classify it with the skill that appears dominant: in a question involving program design and code-writing, the code-writing would probably dominate.

Style of question. This feature indicates what form of answer is expected by the question. The choices are: multiple choice, short answer (including definitions, results of tracing or debugging, and tables), program code, Parsons problem (Parsons & Haden 2006), and graphical representation (for example, concept, flow chart, class diagram, picture of a data structure). Only one of the above can be chosen. Similar categories were used by Petersen et al (2011).

Open/closed. A question that has only one possible correct answer is classified as closed. All others are classified as open.

Cultural references. Is there any use of terms, activities, or scenarios that may be specific to a cultural group and may influence the ability of those outside the group to answer the question? There might be references to a particular ethnic group and their customs, but a cultural reference need not be ethnic. For example, a question might use vocabulary or concepts that refer to a specific sport, such as cricket.

Degree of difficulty. Low, medium, or high. This is an attempt to estimate how difficult the average student would find the question at the end of an introductory course. This classification is similar to that used by Simon et al (2010) in their analysis of CS2 exam papers and Goldfinch et al (2008) in their analysis of mechanics examination papers.

For reasons explained in the next section, the remaining five measures were not used in the current analysis, and therefore their description here is far more brief than the description given to and used by the classifiers.

Explicitness. Low, medium, or high. Extent to which the question states explicitly 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.

Operational complexity. Low, medium, or high. The number and sophistication of the tasks to be performed.

Conceptual complexity. Low, medium, or high. The types and combinations of the concepts that must be known in order to correctly answer the question.

Linguistic complexity. Low, medium, or high. The length, sophistication, and general comprehensibility of the question.

Intellectual complexity. Where the question fits into Bloom's taxonomy (Anderson & Sosniak 1994).

The measures of complexity were originally used by Williams and Clarke (1997) in the domain of mathematics, and were applied to the computing domain by Carbone (2007).

4 Inter-rater reliability

As mentioned in Section 2, a number of studies have classified examinations in various ways. However, none of those studies has really established whether their classification systems are reliable across multiple raters.

Simon et al (2010) report at least 80% agreement on their classification, but in each instance this was between just two classifiers, one of whom classified a full set of questions and the other of whom classified 20% of those questions to check the level of agreement. Furthermore, their analysis deals only with questions in highly specific topics, and the questions classified by each main classifier were all in the same topic area. It is conceivable that all of these factors would contribute to a higher level of agreement than might be expected among a large number of classifiers analysing a broader range of questions.

Petersen et al (2011) did not conduct an inter-rater reliability test. After classifying the questions they were considering, the individual classifiers discussed their classifications in an attempt to reach consensus. Even then, they report difficulty in reaching consensus on most of the measures they were applying.

Goldfinch et al (2008) do not report an attempt to measure agreement among the classifiers. Like Petersen et al (2011) they classified individually and then attempted to reach consensus, and like Petersen et al they found it remarkably difficult to do so.

For this project we chose to conduct a formal and transparent test of inter-rater reliability. With few such tests reported in the computing education literature, we felt it important to conduct and to report on this test.

4.1 Reliability test 1: individual

The first test of inter-rater reliability was carried out on the full scheme of 11 categories. All 12 participants independently classified the 33 questions of the same examination in all 11 categories.

All categories but one were analysed using the Fleiss-Davies kappa for inter-rater reliability (Davies & Fleiss 1982). Because the scheme permits multiple topics to be recorded for a question, the Topics category could not be analysed by this measure, which depends upon the selection of single values.

Table 1 shows the results of the inter-rater reliability test. On kappa measurements of this sort, an agreement of less than 40% is generally considered to be poor; between 40% and 75% is considered fair to good; and more than 75% is rated excellent (Banerjee et al 1999).

Perhaps the most startling figure in Table 1 is the 73% agreement on the percentage mark for each question. This was simply a matter of copying the mark for each question from the exam paper to the spreadsheet used for classifying. The bulk of the disagreement was due to one classifier who neglected to enter any values for the percentage mark. Once this was remedied, the agreement was still only 98%, because two classifiers had each wrongly copied one value. This is a salutary reminder that data entry errors do happen, and we resolved that all of

Category	Reliability	Reliability range
Percentage	73%	fair to good
Skill required	73%	fair to good
Style of question	90%	excellent
Open/closed	60%	fair to good
Cultural references	15%	poor
Degree of difficulty	43%	fair to good
*Explicitness	31%	poor
*Operational complexity	52%	fair to good
*Conceptual complexity	34%	poor
*Linguistic complexity	47%	fair to good
*Intellectual complexity	27%	poor

Table 1: Inter-rater reliability for 11 categories of the initial scheme (the 12th cannot be analysed by this measure). *The categories marked with asterisks were dropped for the classifying reported in this paper.*

our subsequent classifying would be conducted by pairs, in the expectation that this would help to eliminate such errors.

While we did not expect full agreement on the other measures, we were still surprised at the extent of disagreement. More often than not, each classifier felt reasonably confident that they could at least determine how difficult a question is; yet agreement on that measure was an uninspiring 43%. Like Goldfinch et al (2008) and Petersen et al (2011) we realised that the difficulty of a question is strongly dependent on what was taught in the course and how it was taught, and without that information we could only rate the questions according to how difficult we believed our own students would find them.

Following this rather disappointing result, the five categories dealing with complexity, marked in Table 1 with asterisks, were dropped from the scheme until we could find a way to improve the reliability of classification on those measures.

In view of its exceedingly poor agreement, it might seem strange that we did not drop the cultural references category at this point. One reason is that the nature of the disagreements was different. On the complexity measures the classifications tended to be spread more or less evenly across the possible values, and we hope that with further clarification and practice it will be possible to reduce the spread. On the cultural references measure the disagreement was invariably that one classifier saw a reference that others had not seen, but tended to acknowledge after discussion. This is discussed further in section 5.6.

4.2 Reliability test 2: individual

Having thoroughly classified one exam paper in the first inter-rater reliability test, we classified a further ten exams according to the remaining categories. Classifiers worked in pairs, first classifying each question individually, then discussing their classifications and seeking consensus where there was disagreement.

	Fleiss-Davies Kappa		
Category	**Test 1 (solo)**	**Test 2 (solo)**	**Test 3 (pair)**
Percentage	73%	100%	100%
Skill required	73%	73%	84%
Style of question	90%	89%	93%
Open/closed	60%	73%	86%
Cultural references	15%	33%	37%
Degree of difficulty	43%	54%	60%

Table 2: Inter-rater reliability for six categories of the interim scheme (the seventh cannot be analysed by this measure)

A second inter-rater reliability test was then conducted to determine whether the additional practice and the experience of working in pairs had improved the level of agreement. Again all 12 participants classified a single complete examination, this one consisting of 28 questions. For completeness, it should be noted that at this point one of the original 12 members became unavailable to continue with the work, and a new member joined the project.

4.3 Reliability test 3: pairs

When two classifiers disagree, this is either because one of them has made a minor error, which should be picked up immediately, or because there is genuine scope for disagreement. In the latter case, two people discussing the question might be more likely than one person alone to reach the same conclusion as others. For this reason, immediately following the second inter-reliability test the individual classifiers were formed into pairs and asked to agree on each of the classifications of that same examination.

The pairs for this third test were not self-selected, and were generally not the same as the pairs that had worked together on the first set of classifications. Instead they were selected by their order of completion of the individual reliability test. When the first two classifiers had completed their individual classification of the exam questions, they were formed into a pair and asked to come up with an agreed classification for the same questions; when the next two individuals had finished, they were formed into the second pair; and so on.

4.4 Comparing the reliability tests

Table 2 shows the results of all three inter-rater reliability tests on the six categories that they have in common.

It is pleasing to see that between the first two tests, reliability generally improved with time and practice.

It is also pleasing to see that the agreement between pairs in the third test was an improvement on the agreement between individuals in the second test. On the basis of this finding, we conclude that pair classification is more reliable than individual classification.

Neither of these findings is surprising, but such findings are seldom reported, so we feel that there is value in explicitly reporting them here. On the basis of the second finding, we plan to conduct all of our subsequent classification in pairs.

5 Results

This section presents the results of analysing 20 introductory programming exam papers using the exam classification scheme. A total of 469 questions were identified in these exams, with the number of questions in an exam ranging from 4 to 41. For each question the percentage mark allocated was recorded, and this was used as a weighting factor when calculating the contribution of each question to the values in each category.

5.1 Exam paper demographics

The 20 exam papers in the study were sourced from ten institutions in five countries. They were all used in introductory programming courses, eighteen at the undergraduate level and two at the postgraduate level. Course demographics varied from 25 students on a single campus to 800 students over four domestic and two overseas campuses. Most courses used Java with a variety of IDEs (BlueJ, JCreator, Netbeans, Eclipse), one used JavaScript, one used C# with Visual Studio, one used Visual Basic, one used VBA (Visual Basic for Applications) and one used Python. Table 3 shows further specific information about the 20 papers and the courses in which they are used.

5.2 Topics covered

For each question we recorded up to three topics that we considered were central to the question. From our original set of 26 topics, two topics (algorithm complexity and operator overloading) did not appear in the data set; and during analysis we added four further topics (events, expressions, notional machine and class libraries), giving a final list of 28 topics.

Table 4 shows the topics classified and their percentage coverage over the exams in the sample. Topics with the greatest coverage were OO concepts, methods, loops, arrays, program design, I/O and

Paper	Paper source	Exam characteristics				Teaching context			
	Country	Format	Style	% of final mark	Duration (hrs)	Enrolment	Mode	Approach	Programming language
1	New Zealand	Paper	Closed book	40	2	150-200	Campus	Objects first	Java
2	New Zealand	Paper	Closed book	40	2	180	Campus	Objects first	Java (Karel the robot)
3	Australia	Paper	Closed book	40	3	240	Campus	Objects first	Java
4	Australia	Paper	Closed book	50	2	450	Online	Programming logic, then Java	Alice, Java
5	Australia	Paper	Closed book	50	3	120	Campus	Objects later	Java
6	Australia	Paper	Closed book	50	3	250	Campus	Objects later	Visual Basic
7	Australia	Paper	Closed book	50	3	50	Mixed	Objects first	Java
8	Australia	Paper	Closed book	50	3	255	Campus	Objects later	C#
9	Australia	Paper	Closed book	60	2	250	Campus	Objects first	Java
10	Australia	Paper	Closed book	60	2.5	60	Campus	Objects later	VBA
11	Australia	Paper	Closed book	60	3	700-800	Campus	Objects later	Java
12	Australia	Paper	Closed book	60	3	700-800	Mixed	Objects later	Java
13	Australia	Paper	Closed book	60	3	700-800	Mixed	Objects later	Java
14	Finland	Paper	Closed book	70	3	20	Campus	Procedural	Python
15	Finland	Paper	Closed book	80	3	60	Mixed	Objects later	Java
16	England	Paper	Closed book	80	3	100	Campus	Objects later	Java
17	Australia	Paper	Mixed	50	2	180	Mixed	Web script, procedural	JavaScript
18	Australia	Paper	Mixed	70	2	337	Campus	Objects first	Java
19	USA	Paper & online	Open book	25	2	25	Campus	Objects later	Java
20	USA	Paper & online	Open book	25	4	30	Campus	Objects later	Java

Table 3: Exam papers classified in this study

Topic	% Coverage
OO concepts (includes constructors, classes, objects, polymorphism, object identity, information hiding, encapsulation)	35.8
Methods (includes functions, parameters, procedures and subroutines)	34.5
Loops (subsumes operators)	32.3
Arrays	26.3
Program design	16.9
I/O	12.3
Selection (subsumes operators)	11.3
Assignment	8.2
File I/O	6.8
Parameter passing	6.7
Strings	6.2
data types& variables	4.4
Arithmetic operators	3.5
Error handling	3.1
Collections (other than arrays)	2.8
Relational operators	1.9
Scope (includes visibility)	1.8
GUI	1.8
Testing	1.3
Constants, Events, Expressions, Lifetime, Logical operators, Programming standards, Recursion	< 1 each

Table 4: Topics and their coverage over the 20 exams

selection. Eleven topics had less than 2% coverage.

The study conducted by Elliott Tew and Guzdial (2010) identified a set of eight concepts most commonly covered in CS1 courses; six of these appear in the top seven topics listed in Table 4. Their top eight concepts did not include program design, but included recursion and logical operators, which we found had low coverage (0.7% and 0.9% respectively).

5.3 Skill required

From a list of eight skills, each question was classified according to the main skill required to answer the question. Figure 1 shows the overall percentage coverage of each required skill over the 20 exams in the dataset. The most frequently required skill was code writing (48%). The five skills concerning code (writing, tracing, explaining, debugging and modifying) together covered 81% of all exams, the remainder being taken by knowledge recall (10%), design (7%) and (2%) testing. We recognise that writing code often also involves a degree of program design, but we classified questions under program design only if they did not involve coding.

Figure 3 shows a summary of the skills required in each exam. In this graph the five skills associated with coding have been combined into a single coding category. The graph shows that coding in these various forms is the predominant skill assessed in introductory programming exams.

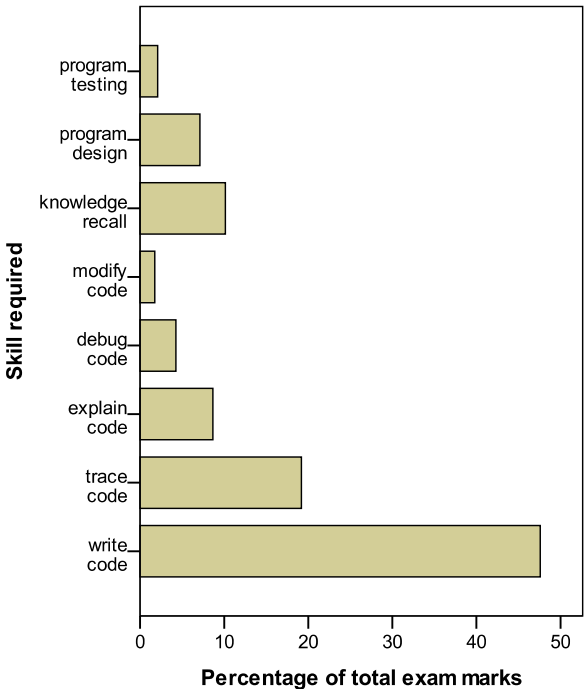

Figure 1: Skills required to answer questions

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 one exam that falls below 100% does so because it includes material other than programming, and we analysed only the programming-related questions.

5.4 Question style

The most common question style involved producing code, with 54% of the marks allocated for code-writing questions (including Parson's problems). Short-answer questions make up 28% of the exams, multiple-choice questions 17%, and graphical style less than 2% (see Figure 2). These findings are somewhat comparable with those of Petersen et al (2011), whose study of CS1 exam content found that 59% of the exam weight was for coding questions, 36% for short answer, 7% for multiple choice, and 3% for graphical questions.

Figure 4 summarises the question styles in each exam, and shows a wide variation across the exams. One exam consists entirely of multiple choice questions, while more than half have no multiple choice questions. It is interesting to note that although coding is the predominant style overall, in two exams there is no code writing required. Petersen et al (2011) also found that the percentage of code writing varied across the CS1 exams they studied.

5.5 Open/closed

The questions were coded according to whether they were open or closed in nature. More marks were allocated to questions that were open (61%) than closed (39%), but this varied markedly over the exams in our sample, as shown in Figure 5. In two exams all questions were closed, and in one exam all questions were open.

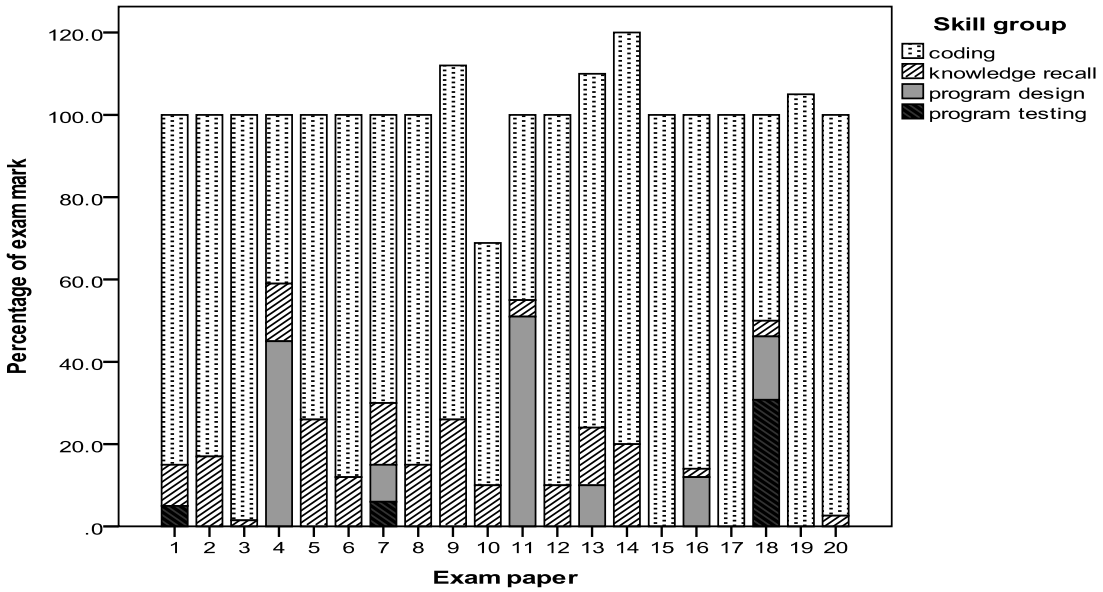

Figure 3: Skills required in each exam

5.6 Cultural references

Cultural references were identified in only eight of the 469 questions analysed, making up a little more than 2% of the available marks. This is so small as to suggest that it might not be worth assessing or reporting – especially as the trial classification showed that any cultural references tended to be spotted first by a single classifier, and only then agreed to by others. However, one likely extension of this work will be to establish a repository of exam questions for the use of educators. In such a repository, this category would serve to alert users that somebody feels a particular question may cause problems for some students outside a particular context or culture.

5.7 Level of difficulty

The questions were classified according to the perceived level of difficulty for a student at the end of an introductory programming course. Overall, half of the marks (50%) were allocated to questions rated as medium difficulty, while low difficulty (26%) and high difficulty (24%) scored about the same. As with other categories, levels of difficulty varied greatly over the exams in our sample, as shown in Figure 6. By comparison, in the data structures questions that they analysed, Simon et al (2010) classified more questions as high (42%), fewer questions as medium (40%), and about the same proportion as low in difficulty.

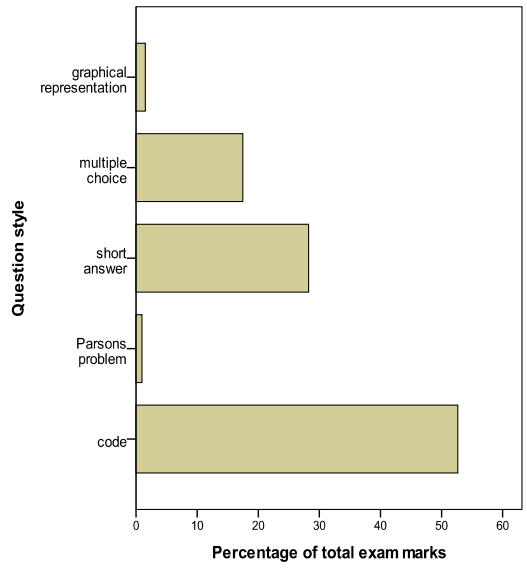

Figure 2: Marks awarded for each style of question

6 Discussion

A number of computing education research groups are undertaking classification of various kinds, presumably sharing our belief that being able to accurately describe a concept is an important step on the road to understanding it. However, there is little point to a classification system unless it can be clearly established that the system is reliable across multiple classifiers.

In this paper we lay out the steps that were taken to assess the reliability of our scheme for classifying exam questions. We explicitly apply a recognised inter-rater reliability measure, developed and verified by statisticians, and we explain at which stages of the classification we applied this measure. We explain our decision to drop several categories of our scheme until we can find a way to improve the inter-rater reliability of those categories.

We are therefore able to provide evidence that reliability appears to improve as the classifiers do more classifying, and that classifying in pairs is more reliable than classifying by individuals.

We believe that an approach of this rigour is essential if readers are to have faith in the findings that we report.

The exams that we have analysed show a very heavy emphasis on coding skills, and the topics covered are concerned mainly with programming elements and constructs. This is not surprising in courses that teach programming, but it is worth noting that, while there was some coverage of the related topic of program design, there was very little focus on examining programming standards and program testing.

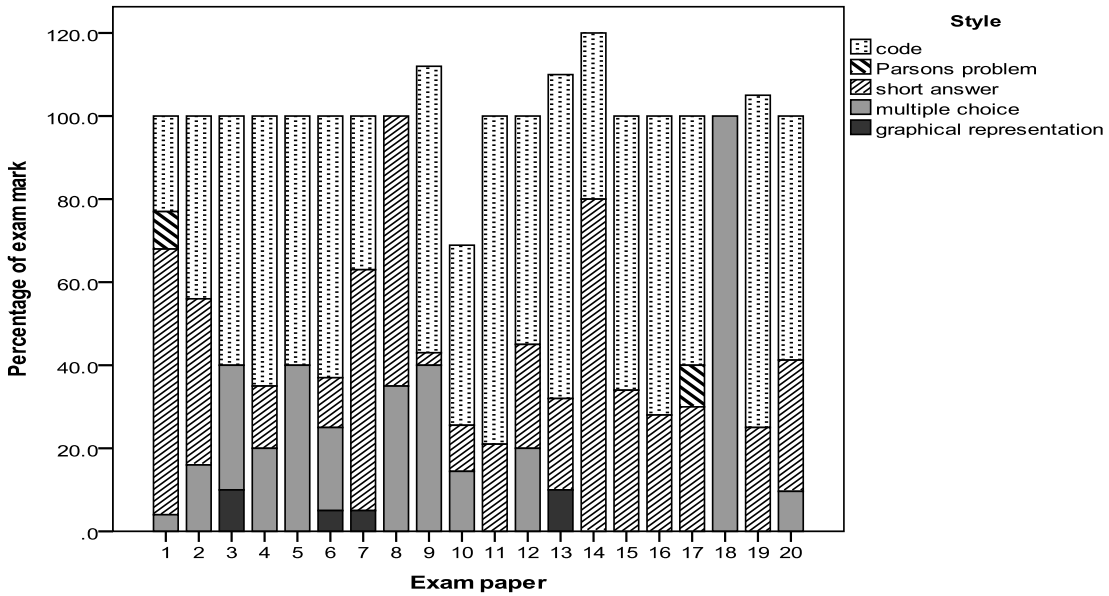

Figure 4: Marks for styles of question in each exam

Of course the skills being examined might not represent the full extent of what is taught in the course. Further material might be covered by assessment items other than the final exam, or indeed might not be assessed at all, even though it is explicitly taught. It is worth noting that a recent study by Shuhidan et al (2010) found that half the instructors surveyed believed that summative assessment was a valid measure of a student's ability to program.

We have found a wide variation in the styles of question used in exams. Within a single exam, this variety could offer students a number of different ways to demonstrate what they have learned. Between exams it raises the question of whether different forms of questions are equally valid assessments of acquired programming skills. For example, more than half of the exams we analysed had a multiple choice component, and one of them was entirely multiple choice. The study by Shuhidan et al (2010) found that the use of multiple choice questions is controversial. At this stage of our study we have not tried to determine why particular styles of question were used; we intend to pursue this question in our future work.

The variation among raters in the trial raises some interesting questions. Most of the participants are or have been involved in teaching introductory programming courses, yet the agreement on level of difficulty was only 43% in the first trial and 54% in the second, rising to 60% for the pair classification. Essentially, there was little or no consensus on whether questions were easy, moderate, or difficult. Both at the workshop and following the trial, discussion of specific questions brought out good arguments for each possible level of difficulty, making it clear that what we are trying to determine is highly subjective, and depends not just upon the feelings of individual participants but on their knowledge of the

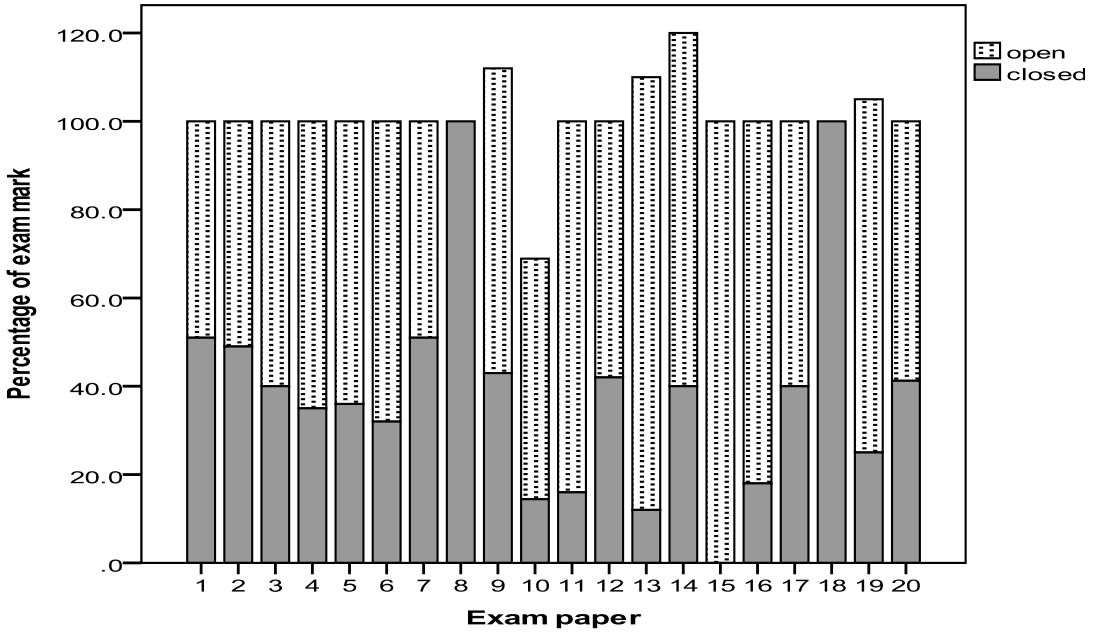

Figure 5: Marks for open and closed questions for each exam

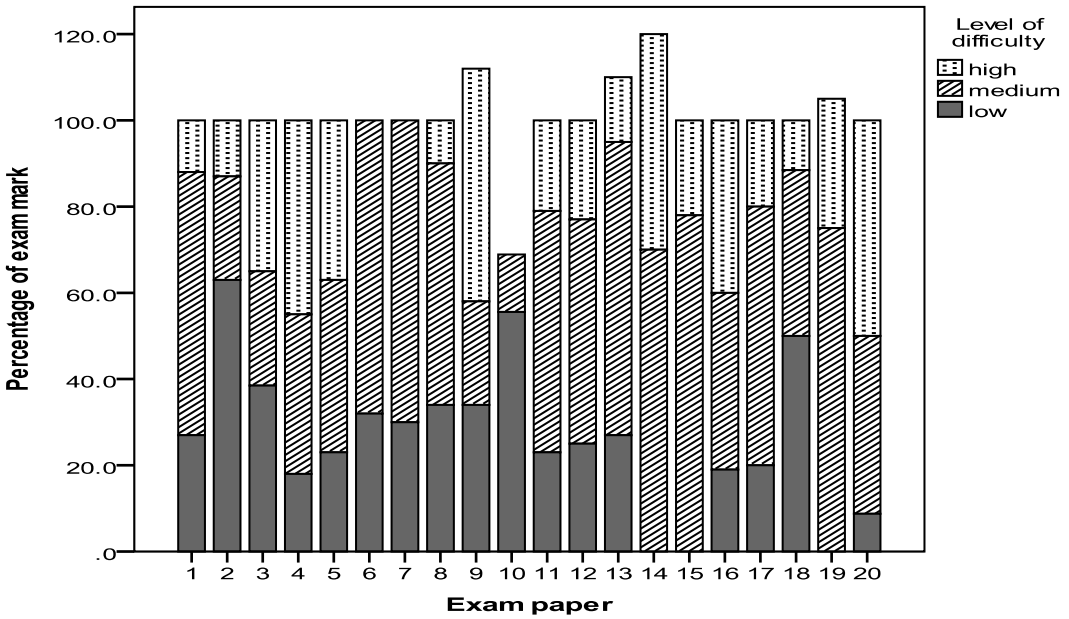

Figure 6: Level of difficulty for questions in each exam

courses that they teach and how their students would therefore respond to those particular questions. Perhaps it is also influenced in some small way by aspects of the culture of the institutions at which the individual participants are employed.

It is a consequence of this line of thought that, even with the appropriate training, the author of an exam is unlikely to classify it as we do except on the trivial measures. The author is fully conversant with the course and its context, and is thus better able to classify the exam within that context. Our classification, on the other hand, is being conducted in the context of the other introductory programming exams that we classify, with no detailed knowledge of how each individual course was taught. We are exploring the range of exams and exam questions that we encounter, from what we hope is a reasonably consistent perspective.

7 Future work

We have classified 20 introductory programming examinations, but this is not yet a large enough set to furnish a general picture of examinations in introductory programming courses. For example, all of these exams are in procedural and/or object-oriented programming. We plan to classify a broader set of examinations, including some from functional programming courses and some from logic programming courses. With this expanded data set we hope to be able to form a broad view of what introductory programming exams consist of.

In parallel with this further classification we intend to explore the role of formal examinations in programming courses. It is not obvious that a written examination of short duration is the best way to assess students' acquisition of a skill that is arguably best applied while working for longer periods at a computer. Why, then, do so many programming courses include a written exam as a major component of their assessment? We intend to interview a number of teachers of introductory programming courses in the hope of eliciting an answer to this question.

In addition, we hope that the interviews will give us an insight into how academics design and create their exams, and to what extent that process is tied in with the stated learning objectives of the course.

Once we have completed the additional classification and the interviews, we hope to be able to present a rich picture of the nature and role of examinations in introductory programming courses.

8 References

Anderson, LW & LA Sosniak (1994). Excerpts from "Taxonomy of Educational Objectives, The Classification of Educational Goals, Handbook I: Cognitive Domain". In Bloom's Taxonomy: A Forty Year Retrospective, LW Anderson and LA Sosniak, Eds. Chicago, Illinois, USA: The University of Chicago Press, 9-27.

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

Carbone, A (2007). Principles for designing programming tasks: how task characteristics influence student learning of programming. PhD dissertation, Monash University, Australia.

Clear, T, J Whalley, R Lister, A Carbone, M Hu, J Sheard, B Simon, & E Thompson (2008). Reliably classifying novice programmer exam response using the SOLO taxonomy. 21st Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ 2008), Auckland, New Zealand, 23-30.

Dale, N (2005). Content and emphasis in CS1. SIGCSE Bulletin 37:69-73.

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

Dart, B & G Boulton-Lewis (1998). The SOLO model: Addressing fundamental measurement issues. Teaching and Learning in Higher Education, M. Turpin, Ed. Camberwell, Victoria, Australia: ACER Press, 145-176.

Davies, M & JL Fleiss (1982). Measuring agreement for multinomial data, Biometrics 38:1047-1051.

Elliott Tew, A (2010). Assessing fundamental introductory computing concept knowledge in a language independent manner. PhD dissertation, Georgia Institute of Technology, USA.

Elliott Tew, A & M Guzdial (2010). Developing a validated assessment of fundamental CS1 concepts. SIGCSE 2010, Milwaukee, Wisconsin, USA, 97-101.

Goldfinch, T, AL Carew, A Gardner, A Henderson, T McCarthy, & G Thomas (2008). Cross-institutional comparison of mechanics examinations: a guide for the curious. Conference of the Australasian Association for Engineering Education, Yeppoon, Australia, 1-8.

Lister, R (2008). After the gold rush: toward sustainable scholarship in computing. Tenth Australasian Computing Education Conference (ACE 2008), Wollongong, Australia, 3-17.

Lister, R (2011). Concrete and other neo-Piagetian forms of reasoning in the novice programmer. 13th Australasian Computing Education Conference (ACE 2011), Perth, Australia, 9-18.

Lister, R, T Clear, Simon, DJ Bouvier, P Carter, A Eckerdal, J Jacková, M Lopez, R McCartney, P Robbins, O Seppälä, & E Thompson (2010). Naturally occurring data as research instrument: analyzing examination responses to study the novice programmer. SIGCSE Bulletin 41:156-173.

Lopez, M, J Whalley, P Robbins, & R Lister (2008). Relationships between reading, tracing and writing skills in introductory programming. Fourth International Computing Education Research Workshop (ICER 2008), Sydney, Australia, 101-112.

Morrison, B, M Clancy, R McCartney, B Richards, & K Sanders (2011). Applying data structures in exams. SIGCSE 2011, Dallas, Texas, USA, 631-636.

Parsons, D & P Haden (2006). Parson's programmimg puzzles: a fun and effective learning tool for first programming courses. Eighth Australasian Computing Education Conference (ACE 2006), Hobart, Australia, 157-163.

Petersen, A, M Craig, & D Zingaro (2011). Reviewing CS1 exam question content. SIGCSE 2011, Dallas, Texas, USA, 631-636.

Schulte, C & J Bennedsen (2006). What do teachers teach in introductory programming? Second International Computing Education Research Workshop (ICER 2006), Canterbury, UK, 17-28.

Sheard, J, A Carbone, R Lister, B Simon., E Thompson, & J Whalley (2008). Going SOLO to assess novice programmers. 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2008), Madrid, Spain, 209-213.

Sheard, J, Simon, M Hamilton, & J Lönnberg (2009). Analysis of research into the teaching and learning of programming. Fifth International Computing Education Research Workshop (ICER 2009), Berkeley, CA, USA, 93-104.

Sheard, J, Simon, A Carbone, D Chinn, M-J Laakso, T Clear, M de Raadt, D D'Souza, J Harland, R Lister, A Philpott, & G Warburton (2011). Exploring programming assessment instruments: a classification scheme for examination questions. Seventh International Computing Education Research Workshop (ICER 2011), Providence, RI, USA, 33-38.

Shuhidan, S, M Hamilton, & D D'Souza (2010). Instructor perspectives of multiple-choice questions in summative assessment for novice programmers, Computer Science Education 20:229-259.

Simon, A Carbone, M de Raadt, R Lister, M Hamilton, & J Sheard (2008). Classifying computing education papers: process and results. Fourth International Computing Education Research Workshop (ICER 2008), Sydney, NSW, Australia, 161-171.

Simon, B, M Clancy, R McCartney, B Morrison, B Richards, & K Sanders (2010). Making sense of data structures exams. Sixth International Computing Education Research Workshop (ICER 2010), Aarhus, Denmark, 97-105.

Venables, A, G Tan, & R Lister (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. Fifth International Computing Education Research Workshop (ICER 2009), Berkeley, CA, USA, 117-128.

Whalley, J, R Lister, E Thompson, T Clear, P Robbins, PKA Kumar, & C Prasad (2006). An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies, Eighth Australasian Computing Education Conference (ACE 2006), Hobart, Australia, 243-252.

Williams, D & D Clarke (1997). Mathematical task complexity and task selection. Mathematical Association of Victoria 34th Annual Conference, Clayton, Vic, Australia, 406-415.