# Exploring Programming Assessment Instruments: a Classification Scheme for Examination Questions

Judy Sheard
Monash University
judy.sheard@infotech.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

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

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

Michael de Raadt
University of Southern Queensland
deraadt@usq.edu.au

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

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

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

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

Geoff Warburton
RMIT University
geoffw@cs.rmit.edu.au

## ABSTRACT

This paper describes the development of a classification scheme that can be used to investigate the characteristics of introductory programming examinations. We describe the process of developing the scheme, explain its categories, and present a taste of the results of a pilot analysis of a set of CS1 exam papers. 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.

## Categories and Subject Descriptors

K3.2 [**Computers and education**]: Computer and Information Science Education – *computer science education*

## General Terms

Measurement

## Keywords

Examination papers, CS1, introductory programming

## 1. INTRODUCTION

End of course formal examinations are one of the main techniques used for summative assessment of students in programming courses. Construction of an examination instrument is an important task, as the exam is used both to measure the level of knowledge and skill that students have reached at the end of the course and to grade and rank the students. A poorly constructed exam may not give a fair assessment of students' abilities, perhaps affecting their grades and their progression through their program of study.

Developing an examination paper is often an individual task, with the exam's format depending on the examiner's own preferences as well as on examination questions inherited from colleagues in previous offerings of the course. There are a number of ways that skills and knowledge may be assessed, and exams typically have a number of questions in a variety of styles, giving students different ways to demonstrate their knowledge and skills. 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. Considering the central role of the formal examination in assessing our students, it is important to understand the nature of the instruments we are using for this task. Tew [22] claims that "the field of computing lacks valid and reliable assessment instruments for pedagogical or research purposes" (p.xiii). This is a concern, because if the instruments we are using are neither valid nor reliable, how can we rely upon our interpretation of the results?

In this paper we report the development of 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 report a sample of the results. We also explain the role of the scheme as part of a larger project which aims to investigate the nature and composition of formal examination instruments and the pedagogical intentions of the educators who construct these instruments.

## 2. ASSESSMENT BY EXAMINATION

Assessment is a critical component of our work as educators. In assessment we stand in judgment on our students to determine the level of learning that they have achieved in the curriculum of the course. It is critical that the instruments we use should give students the opportunity to demonstrate what they have learned about the course topics. However, assessment is also a strong influence on student learning behavior, as students tend to focus on and direct their efforts to the assessment tasks. As Ramsden maintains, "Students will study what they think will be assessed" [16] (p70). Biggs' theory of constructive alignment proposes an alignment of teaching and assessment and stresses the importance of ensuring that the assessment tasks mirror the desired learning outcomes [3].

Given the importance of assessment, surprisingly few studies have investigated the content and characteristics of the instruments that we use [23]. A number of research studies have used examination instruments to measure levels of learning and to explore the process of learning. A body of work under the auspices of the BRACElet project has analyzed students' responses to examination questions [4, 11, 12, 18, 24]. Interest in this work stemmed from earlier studies (such as that of Whalley et al [25]) that attempted to classify responses to examination questions using Bloom's taxonomy [1] and the SOLO taxonomy [7]. 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 [10] proposed that a neo-Piagetian perspective could prove useful in explaining the programming ability of students.

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 [9] investigated the range of topics covered and the perceived level of difficulty of exam questions. Within the computing discipline, Simon et al [21] analyzed 76 CS2 (data structures) examination papers, but considered only questions on the topics of stacks and hash tables, which make up less than 20% of the marks available in those exams. Their analysis focused on the question style (e.g. multiple choice), the cognitive skills required and the level of difficulty of the questions. They further classified questions according to whether they required knowledge of the implementation of the data structure, the operations available with that data structure, or how to apply these operations. Following this study, a further analysis of 59 CS2 papers in the same dataset by Morrison et al [13]explored the *apply* questions, showing the range of question styles that can be used to test students' skill in applying data structures concepts.

Petersen et al [15] analyzed 15 CS1 exam papers to determine the concepts and skills covered. They found that there was a high emphasis on code writing questions, but much variation across the exams in the study. After finding that many questions required students to deal simultaneously with a high number of concepts, they proposed that single- or dual-concept questions might give students a better chance to demonstrate understanding. They suggested that more effort was needed to develop alternative types of question that focus on smaller sets of concepts,

A study by Shuhidan et al [19] investigated the use of multiple-choice questions in summative assessment, and found that the use of these questions remains controversial.

Like Petersen's, our study focuses on introductory programming examination papers: we develop a classification scheme for the purpose of analyzing these papers to give a comprehensive view of the course coverage, the styles of question employed, and the skills that students require to answer the questions.

## 3. RESEARCH APPROACH

This section describes how the exam question classification scheme was developed and applied, detailing the iterative process that was followed. After an introductory workshop, a project working team trialed the classification scheme on a single exam paper. This generated data that was used to refine the scheme with a view to achieving good inter-rater reliability results. Finally, a pilot study was conducted whereby the scheme was applied to a variety of exam papers.

### 3.1 Development of the Exam Question Classification Scheme

The idea of classifying exam questions stemmed from a BRACElet workshop following ICER 2008 in Sydney, a workshop funded by a fellowship of what subsequently became the Australian Learning and Teaching Council. The idea was formalized by a group of four project leaders, who produced a working draft of an exam question classification scheme.

Our intentions in developing a classification scheme are not unlike those of Bloom et al [1]: we wish to develop a common language to express the kinds of concepts and skills that students are expected to acquire in the first semester of studying object-oriented programming. As a first step we aim to discover, through analysis of actual CS1 exam papers, what questions instructors use to assess their students' mastery of concepts and skills.

Our initial scheme considered an individual question as the unit of analysis. The goal was to create a set of properties that describes different aspects of an exam question. We chose to characterize exam questions by the topic areas they cover, the style of the question, and the skills a student would need to have mastered to answer the question. We were also interested in a number of measures of complexity, such as the linguistic complexity of the question and its intellectual complexity as according to Bloom's taxonomy.

### 3.2 Presenting and Refining the Exam Question Classification Scheme at a Workshop

The exam question classification scheme was introduced to the computing education community at a half-day workshop at the 2011 Australasian Computing Education conference (ACE 2011) in Perth. The workshop was used to present the scheme and to obtain a sense of its usefulness to the broader computing education community. Sixteen participants attended, from eleven tertiary institutions. The workshop was conducted in four stages: description, trial, and refinement of the classification scheme, and discussion of its applicability and further development.

Following the introduction of the classification scheme, participants applied it to a selection of questions from a first-year introductory programming exam paper. Participants spent approximately 60 minutes classifying multiple-choice questions, code interpretation questions and small coding questions. During this time they discussed the categories of the classification scheme, clarifying their meaning and understanding, so that it became easier to approach consensus on the classifications.

### 3.3 A Revised Scheme, Trial Classification, and Inter-Rater Reliability

Following the workshop, 12 participants continued working on the project. Guided by feedback from the workshop, the project leaders refined the classification scheme and clarified some of the explanations behind the categories. Each member of the project working team then individually classified all of the questions on a single introductory exam paper using the refined question classification scheme, with a view to measuring the agreement among classifiers and thus the reliability of the scheme.

All categories but one of the trial classifications were analyzed using the Fleiss-Davies kappa for inter-rater reliability [8]. Because the scheme permits multiple topics to be recorded for a question, the Topics category could not be analyzed 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 [2].

Some members of the team found these reliability measures surprising or even alarming. When classifying a question we appear to be confident not only in our own classification but in the belief that like-minded people will choose the same values. The trial classification made it clear that this is not the case.

Perhaps most startling, but also most instructive, was the 73% rating for Percentage. This value is simply the percentage of the exam that each question is worth. The bulk of the disagreement was due to one member who neglected to complete this field at all. Once these values were added the reliability rose, but only to 98%, as two members had miscopied marks from the exam paper. The lesson from this is that even when raters agree, the measure of agreement can be reduced by data entry error.

The only excellent agreement was for Style of question, which most of the team expected to generate 100% reliability. Style of question has only five possible values, including multiple choice, short answer, and program code; yet still there was not complete agreement.

The cultural references question was simply whether the question included references that might be less meaningful to some groups of students than to others. In most cases such references were signaled by just one member, but others tended to agree when the references were explained. One question, for example, was based on a class for 'Chat'; one member observed that less computer-literate students might not be fully aware of what Chat is, and so potentially be disadvantaged by such a question unless the explanation was highly explicit. While others then agreed, the measured agreement was low because only one member had made this observation in the first instance.

**Table 1: Inter-rater reliability for 11 categories of the trial scheme; some categories are explained in the next section**

| 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 |

* Use of these categories was discontinued for the pilot study

## 3.4 Further Revision

Based on the results of the trial and inter-rater reliability measurement, further clarifications were made to the scheme, and a number of categories – the last five in Table 1 – were dropped pending further consideration of how agreement might be improved. Participants reviewed their classifications in the light of these revisions and of other members' classifications. As these reclassifications were not independent it was not appropriate to recalculate the inter-rater reliability, but the new classifications did narrow the gap between the trial classification and full agreement.

## 3.5 Pilot Study

Following the clarifications and revisions, the next step was to pilot the scheme on a wider range of papers. Members of the project working team and two other colleagues supplied eleven exam papers from eight universities in five countries (Australia, Finland, New Zealand, UK, and USA).

The team formed pairs to apply the revised scheme to one or two exam papers each, as there is some evidence [20] that pairs classify more reliably than individuals. Members were required to classify the exam questions individually, then to discuss any differences and come to a consensus.

## 4. THE CURRENT CLASSIFICATION SCHEME

The refined scheme, as used to reclassify the first exam and classify the additional ten exams, is described briefly below.

**Percentage of mark allocated.** This represents the percentage of the entire exam that the question is worth. This category can be used as a weighting to determine what proportion of a complete exam covers the mastery of particular topics or skills.

**Topics covered.** An exam question can be assigned at most three of the following topics: data types and 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.

Note that in the list above, topics that follow 'assignment' tend to subsume data types and variables, so any question that is categorized with these later topics need not include data types and 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, which is probably not yet fixed, was compiled from the ACM curriculum, brainstorming among the project leaders, findings from the trial classification, and the computing education literature: a number of studies have investigated the topics taught in introductory programming courses. A survey of 351 CS academics by Dale [5, 6] explored the emphasis placed on different topics in CS1 courses and the perceived difficulty of these topics. Schulte [17] surveyed a similar number of teachers to determine the topics that they teach in introductory programming courses and their opinions about the importance, relevance and difficulty of these topics. Tew and Guzdial [23] analyzed the content of twelve CS1 textbooks to identify the concepts covered in CS1 courses.

**Skill required to answer the question**. The skills included in this category are: pure knowledge recall, trace code (includes expression evaluation), explain code, write code, modify code (includes refactor, rewrite), debug code, design program, test program. Only one skill can be chosen for a question. A question involving both design skills and coding would be classified as coding.

**Style of question.** This is a description of how the student's answer is represented. The choices are: multiple choice, short answer (includes definition, results of tracing or debugging, and tables), program code, Parsons problem [14], and graphical representation (for example, concept, flow chart, class diagram, picture of a data structure). A Parsons problem is a question in which a code segment is 'written' by correctly ordering the lines of code, which are provided in a jumbled order. Only one style can be chosen for each question. Similar categories were used by Petersen [15].

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

**Cultural references**. Yes or no. 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. A question might refer to a sport, such as cricket, using vocabulary or concepts that are specific to that sport.

**Degree of difficulty**. Low, medium, or high. This is an attempt to estimate how difficult the average student at the end of an introductory course would find the question. This classification is similar to that used by Simon et al [21] in their analysis of CS2 exam papers and Goldfinch et al in their analysis of mechanics examination papers [9].

## 4.1 Some Issues

A number of issues arose during the development and trial of the scheme. We describe a few of them here.

**Multi-part questions.** Some questions have multiple parts. Should these be classified as single questions or as multiple questions? This was left to the judgment of the people classifying the exam. A multi-part question asking students to "write down the results of evaluating the following expressions" could generally be classified as one question – although it might be difficult to remain within the limit of three topics. But a question that provides some code and asks students to do several distinct tasks such as explaining, refactoring, and writing additional code would have to be considered as multiple questions.

**Skill required to answer question.** It is sometimes difficult to determine whether a question involves pure knowledge recall or something more. For example, if a multiple choice question asks student to choose which of four lines of code constructs an object of some type and assigns it to a variable of a superclass of that type, does the question involve just knowledge recall, or might it be considered a code-tracing question?

**Open/closed.** There was need for resolution on whether a question that had essentially one answer (that is, there are multiple correct answers, but all express the same idea) should be considered open or closed. It was decided to consider such questions open, limiting the closed response to those questions that literally have only a single correct answer.

**Degree of difficulty.** This was perhaps the most subjective category to classify, as classifiers' judgments will be based on their own individual teaching experience and the pedagogy that they use. For example, a question on a concept or skill that is heavily emphasized in a class could be considered of low difficulty for that class, while in a different class it might be considered of medium difficulty.

**Topics.** The somewhat arbitrary limitation to three topics arose from the trial classification, where as many as 14 different topics were assigned to a single multiple-choice question. It was generally agreed that while all of these topics were pertinent, the restriction would force classifiers to choose the topics that were most pertinent to the question. It was also agreed that certain topics tended to subsume others – for example, arrays tend to subsume loops – and that there was no need to include both a subsumed topic and the one that subsumes it.

## 5. SAMPLE RESULTS

This section presents a taste of the results of the pilot analysis. The 11 exam papers comprised a total of 252 questions, 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 11 exam papers in the study were sourced from eight institutions in five countries. They were all used in introductory programming courses, ten at the undergraduate level and one at the postgraduate level, for classes from 25 students on a single campus to 800 students over four domestic and two international campuses.

The exams were mainly paper-based, though one had an online component, and mainly closed book, though one was open book and two were mixed. They were all of two or three hours' duration, and their weighting in the overall assessment for the course ranged from 25% to 80%. Some were for online students, some for on-campus students, and some for both. Most of them used Java, but Javascript, Alice, and C# were also represented.

## 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 25 topics, three (recursion, GUI, and operator overloading) did not appear in the data set; and during analysis we added two further topics (events and expressions), giving a final list of 24 topics.

Table 2 shows the topics classified and their percentage coverage over the exams in the sample. Topics with the greatest coverage were OO concepts, loops, methods, arrays, program design, and selection. Ten topics had less than 2% coverage.

The study conducted by Tew and Guzdial [23] identified a set of eight concepts most commonly covered in CS1 courses; six of these appear in the top seven topics listed in Table 2. Tew and Guzdial's top eight concepts did not include program design but did include logical operators, which we found had low coverage (1.2%), and recursion, which we found had no coverage at all.

### 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 11 exams in the dataset. The most frequently required skill was code writing (45%). The five skills concerning code (writing, tracing, explaining, debugging, and modifying) together covered 81% of all exams, the remainder being taken by knowledge recall (7%), design (8%) and (4%) testing. We recognize 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.

### 5.4 Cultural References

Cultural references were identified in only 15 of the 252 questions analyzed, making up a little more than 1% 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 possible 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.

## 6. DISCUSSION

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%. Essentially, there was little or no consensus on whether questions were easy, moderate, or difficult. Discussion at the workshop and following the trial brought out some very good arguments for all the classifications, 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 courses that they teach and how their students would therefore respond to each particular question. Perhaps it is also influenced in some small way by aspects of the culture of the institutions at which the individual participants are employed.

**Table 2: Topics and their coverage over the 11 exams**

| Topic | Percentage |
|---|---|
| OO concepts (includes constructors, classes. objects, polymorphism, object identity, information hiding, encapsulation) | 39.0 |
| Loops (subsumes operators) | 36.4 |
| Methods (includes functions, parameters, procedures and subroutines) | 28.6 |
| Arrays | 28.1 |
| Program design | 22.4 |
| Selection (subsumes operators) | 12.2 |
| Assignment | 8.0 |
| I/O | 7.9 |
| Parameter passing | 7.8 |
| File I/O | 4.5 |
| Strings | 3.9 |
| data types& variables | 3.7 |
| Collections (includes collections other than arrays) | 3.6 |
| Error handling | 2.6 |
| Arithmetic operators | 1.8 |
| Relational operators | 1.7 |
| Testing | 1.6 |
| Scope (includes visibility) | 1.6 |
| Logical operators | 1.2 |
| Events | 1.1 |
| Lifetime, Programming standards, Expressions, Constants | < 1 each |

## 7. CONCLUSION AND FURTHER WORK

We have established a number of categories to describe features of and variation among introductory programming exams, and have found that the 11 exams we have considered so far vary greatly in a number of aspects, from their broad characteristics to more specific details such as the topics covered. This is perhaps not surprising, when we consider that there is possibly a great variation in the pedagogic styles and beliefs of the people who set these exams. The next stage of this research will explore the pedagogical foundations of the exams through investigating the motivations and ideas of the educators who design these instruments.

In further analysis of the data we will explore the possibility of relationships between the categories of the classification system, such as between question style and skills required.

Ultimately, we plan to create a repository of introductory programming exams and exam questions for the use of educators. The variety that we have already found suggests that such a repository would be a rich source of material for academics seeking to inject change into their final assessments.

## 8. ACKNOWLEDGMENTS

Our thanks to the 2008 BRACElet workshop attendees who contributed to the initial brainstorming of ideas for classifying exams, and to the 2010 ACE Exam Taxonomy workshop attendees who provided feedback on the initial classification scheme. Thanks also to Jacqueline Whalley and Errol Thompson,

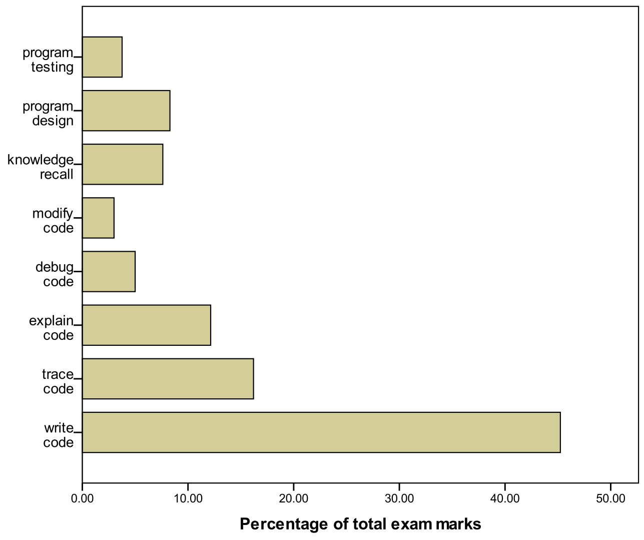

**Figure 1: Skills required to answer questions**

who contributed exams for classification.

## 9. REFERENCES

[1] Anderson, L. W. and L. Sosniak, A., "Excerpts from the "Taxonomy of Educational Objectives, The Classification of Educational Goals, Handbook I: Cognitive Domain," in *Bloom's Taxonomy: A Forty Year Retrospective*, L. W. Anderson and L. Sosniak, A., Eds., ed Chicago, Illinois, USA: The University of Chicago Press, 1994, 9-27

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

[3] Biggs, J. B., "What the Student Does: teaching for enhanced learning," *Higher Education Research and Development*, 18:57-75, 1999

[4] Clear, T., J. Whalley, R. Lister, A. Carbone, M. Hu, J. Sheard, B. Simon, and E. Thompson, "Reliably classifying novice programmer exam response using the SOLO taxonomy," in *NACCQ 2008*, Auckland, New Zealand, 2008

[5] Dale, N., "Content and emphasis in CS1," *inroads - The SIGCSE Bulletin*, 37:69-73, 2005

[6] Dale, N., "Most difficult topics in CS1: Results of an online survey of educators," *inroads - The SIGCSE Bulletin*, 38:49-53, 2006

[7] Dart, B. and G. Boulton-Lewis, "The SOLO model: Addressing fundamental measurement issues," in *Teaching and Learning in Higher Education*, M. Turpin, Ed., ed Camberwell, Victoria, Australia: ACER Press, 1998, 145-176

[8] Davies, M. and J. L. Fleiss, "Measuring agreement for multinomial data," *Biometrics*, 38:1047-1051, 1982

[9] Goldfinch, T., A. L. Carew, A. Gardner, A. Henderson, T. McCarthy, and G. Thomas, "Cross-institutional comparison of mechanics examinations: A guide for the curious," in *AaaE conference*, Yeppoon, 2008, 1-8

[10] Lister, R., "Concrete and other neo-piagetian forms of reasoning in the novice programmer," in *13th Australasian Computing Education conference*, Perth, Australia, 2011

[11] Lister, R., T. Clear, Simon, D. J. Bouvier, P. Carter, A. Eckerdal, J. Jacková, M. Lopez, R. McCartney, P. Robbins, O. Seppälä, and E. Thompson, "Naturally occurring data as research instrument: Analyzing examination responses to study the novice programmer," *inroads - The SIGCSE Bulletin*, 41:156-173, 2010

[12] Lopez, M., J. Whalley, P. Robbins, and R. Lister, "Relationships between reading, tracing and writing skills in introductory programming.," in *Fourth International Workshop on Computing Education Research (ICER '08)*, Sydney, Australia, 2008, 101-112

[13] Morrison, B., M. Clancy, R. McCartney, B. Richards, and K. Sanders, "Applying data structures in exams," in *SIGCSE'11*, Dallas, Texas, USA, 2011, 353-358

[14] Parsons, D. and P. Haden, "Parson's programmimg puzzles: A fun and effective learning tool for first programming courses," in *Eighth Australasian Computing Education conference (ACE2006)*, Hobart, Australia, 2006, 157-163

[15] Petersen, A., M. Craig, and D. Zingaro, "Reviewing CS1 exam question content," in *SIGCSE'11*, Dallas, Texas, USA, 2011, 631-636

[16] Ramsden, P., *Learning to Teach in Higher Education*. New York, NY, USA: Routledge, 1992

[17] Schulte, C. and J. Bennedsen, "What do teachers teach in introductory programming?," in *Second International Computing Education Research workshop (ICER'06)*, Canterbury, UK, 2006, 17-28

[18] Sheard, J., A. Carbone, R. Lister, B. Simon, E. Thompson, and J. Whalley, "Going SOLO to assess novice programmers," in *13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'08)*, Madrid, Spain, 2008, 209-213.

[19] Shuhidan, S., M. Hamilton, and D. D'Souza, "Instructor perspectives of multiple-choice questions in summative assessment for novice programmers," *Computer Science Education*, 20:229-259, 2010

[20] Simon, A. Carbone, M. De Raadt, R. Lister, M. Hamilton, and J. Sheard, "Classifying computing education papers: Process and results," in *4th International Workshop on Computing Education research (ICER 2008)*, Sydney, NSW, Australia, 2008, 161-171

[21] Simon, B., M. Clancy, R. McCartney, B. Morrison, B. Richards, and K. Sanders, "Making sense of data structure exams," in *International Computing Education Research workshop (ICER'10)*, Aarhus, Denmark, 2010, 97-105

[22] Tew, A., "Assessing Fundamental Introductory Computing Concept Knowledge in a Language Independent Manner. ," PhD Dissertation, 2010

[23] Tew, A. E., "Developing a validated assessment of fundamental CS1 concepts," in *SIGCSE'10*, Milwaukee, Wisconsin, USA, 2010, 97-101

[24] Venables, A., G. Tan, and R. Lister, "A closer look at tracing, explaining and code writing skills in the novice programmer. ," in *The fifth International Computing Education Research Workshop (ICER 2009)*, Berkeley, California, USA, 2009

[25] Whalley, J., R. Lister, E. Thompson, T. Clear, P. Robbins, P. K. A. Kumar, and C. Prasad, "An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies," in *Eighth Australasian Computing Education conference (ACE2006)*, Hobart, Australia, 2006, 243-252