Code Classification as a Learning and Assessment Exercise for Novice Programmers

Errol Thompson
Department of Information Systems
Massey University, Wellington
New Zealand
e.l.thompson@massey.ac.nz

Jacqueline Whalley
Computer and Information Sciences
Auckland University of Technology
New Zealand
jacqueline.whalley@aut.ac.nz

Raymond Lister
Faculty of Information Technology
University of Technology, Sydney
Australia
raymond@it.uts.ac.au

Beth Simon
Computer Science and Engineering Department,
University of California, San Diego
USA
esimon@cs.ucsd.edu

Abstract

When students are given code that is very similar in structure or purpose, how well do they actually recognise the similarities and differences? As part of the BRACElet project, a multi-institutional investigation into reading and comprehension skills of novice programmers, students were asked to classify four code segments that found the minimum or maximum in an array of numbers. This paper reports on the analysis of responses to this question and draws conclusions about the students' ability to recognise the similarities and differences in example code. It then raises questions with respect to an approach to teaching that uses variations in code examples.

Keywords: Computing education, novice programming

1 Introduction

The first few months of learning to program can be a disconcerting time for students. It can be equally disconcerting for their teachers. Many teachers have had the experience of talking to a student about a program written by that student, only to discover that the student cannot explain the functioning of portions of that program. While tempting to attribute such an inability to plagiarism, recent research indicates that many novice programmers struggle to reason about code at any level of abstraction above the concrete code structures themselves.

The Leeds Working Group investigated the program reading skills of novice programmers (Lister *et. al*., 2004). The novices, who had completed approximately one semester of programming, were asked to answer a number of multiple-choice questions. The Leeds Group collected data from over 600 novices, at twelve institutions in seven countries, and found that one quarter of the students performed on the multiple choice questions at a level consistent with chance. Furthermore, the Leeds Group analyzed transcripts from approximately 30 students who were asked to "think out loud" as they attempted the multiple choice questions. Almost all of these students answered the questions by hand executing ("tracing") the code (Lister *et. al*., 2004; Fitzgerald, Simon, and Thomas, 2005). A follow-up study asked eight academics to solve a single multiple choice question from the Leeds study (Lister, Simon, Thompson, Whalley, and Prasad, 2006). In contrast to the students, seven of the eight academics used a higher level approach to answering the question. These seven academics analyzed the code to determine what computation was performed by the code. (The code counted the number of common elements stored in two integer arrays.) While they did some hand execution of the code, the academics stopped well short of completely hand executing the code (unlike many students in the Leeds group study).

Studies in many disciplines have consistently found that experts reason at higher abstract levels than novices (Chi, Glaser, and Farr, 1988). The classic study was of chess players (Chase & Simon, 1973). Chess novices tend to remember the position of each piece in isolation, whereas experts organize the pieces into higher level attacking and defensive combinations. In a study of programming that reflected the earlier chess studies, Adelson (1984) demonstrated that expert programmers reason at a more abstract level than novices.

In light of the research on the differences between novices and experts, one of our teaching aims should be to encourage students to reason about their code at a level above the concrete code. However, a number of the questions used by the Leeds group deliberately contained small bugs, which might have discouraged students from reasoning at a high level, and instead may have encouraged them to concretely hand execute code. While many of the students studied by the Leeds Group answered the questions by hand executing the code, it could be argued that this is a result of the type of question asked, not of the preferred thinking style of the students. This work explores a different kind of assessment

This quality assured paper appeared at the *19th Annual Conference of the National Advisory Committee on Computing Qualifications* (NACCQ 2006), Wellington, New Zealand. Samuel Mann and Noel Bridgeman (Eds).

question, which asks students to make explicit statements about their understanding of a code – by asking them to identify similarities and differences in a set of related codes. We explore whether students answering this type of question are more likely to identify higher level purposes in the codes.

1.1 Code Explanation: BRACElet Question 10

The multi-institutional BRACELet Group studied students who had completed one semester of programming (Whalley, *et al*., 2006). One of the aims of BRACElet was to explore the ability of students to reason at a higher level than merely hand executing code. BRACELet developed a question set within two pedagogical frameworks. These frameworks are the revised Bloom's taxonomy of educational objectives (Anderson *et al.*, 2001) and SOLO (Biggs, 1999, Biggs and Collis, 1982).

Of the 11 questions used in the BRACElet study, the first nine were multiple choice questions. Two of these nine questions were taken directly from the Leeds group study.

The tenth question in the BRACElet study required a short answer response. Students were given a piece of code and were asked to "explain in plain English" what the code did. The complete question is given in appendix 2. One type of response given to the question was an explanation of the computation performed by the code (e.g. "it checks to see if the array is sorted"). Only one third of students gave such a response. Many of the remaining students gave a line-by-line description of the code, without giving any indication of understanding what the code did as a whole. The results for this BRACElet "explain in plain English" question are consistent with the Leeds study – in both studies, many students did not manifest an ability to reason abstractly about code.

However, the "explain in plain English" question is open to the same criticism as the Leeds Group questions – the type of response given by students may be due to the form of the question, not the preferred thinking style of the students. The instruction "explain in plain English" is ambiguous. Students may have misunderstood the instruction, and thought a line-by-line explanation was required.

In this paper, we analyze student responses to the 11th question from the BRACElet study. This 11th question, like the 10th "explain in plain English" question, was intended to elicit a student response demonstrating higher level thinking. However, the 11th question is constructed differently, to avoid the ambiguity of "explain in plain English".

2 Code Classification: BRACElet Question 11

Question 11 focuses on the classification of four code segments. Students were asked to describe ways of separating the four code segments into groups – that is to suggest criteria that could be used to classify the code and indicate which codes fit each criteria (see Appendix 1 for details). Many classifications are possible. For example, a student might classify the segments according to whether the segments use a "for" loop or whether they use a "while" loop. A higher level response would be to classify the segments according to the computation performed by the segments – some of segments find the maximum element of the array, while others find the minimum. No limit was given on how many classifications a student should give, so a student was free to give many classifications at differing levels of abstraction (whereas the instruction "Explain in plain English" encourages a response at a single level of abstraction). A table with six rows in it was provided for answers, though some students added rows to the table as needed and many left some rows blank. However, students were specifically told that more than one classification was possible.

In terms of the revised Bloom's taxonomy, this classification question is targeted at the "cognitive process" level of "understand: classifying".

This classification question lends itself to several research questions. For example, when presented with code segments, do novice programmers recognise when two segments of code are performing the same computational task? Is it possible that the language constructs used in the code hide the similarity in purpose from many novices? When comparing the performance of students between the "explain in plain English" question and this classification question, is there a consistency in the level of abstraction chosen by an individual student?

3 Analysis approach: the SOLO taxonomy

The student responses to the classification question were analysed using the SOLO taxonomy (Biggs, 1999, Biggs and Collis, 1982). The SOLO categories devised for analyzing question 11 are presented in Table 1.

A similar set of categorisations were devised to analyse the answers to question 10 (Whalley et al., 2006; Lister et al., 2006). The same broad principles were used to define the categories for analysis of questions 10 and 11. However, the focus of question 10 was on eliciting a single response from the student, whereas question 11 was written so that many differences in the code segments could potentially be used for classification. This difference in the two questions leads to some differences in how the SOLO categories are defined for each question.

- Unclassified "X" student responses referred to things that had no relationship to programming (e.g. "bags on desk") or responses where the four authors could not identify the relationship between the classification and the code segments. The use of classifications unrelated to programming may have been a result of the example of classification provided by the supervisor of the assessment as these "X" responses all came from students at one institution.
- Prestructural responses were focussed on line counts or position on the page. Again this seems to reflect the example of classification given to the assessment group.
- Unistructural responses focus on a language construct or a single statement of the code. Examples would be

Table 1: SOLO categories for Question 11

SOLO category	Description
Relational [R]	Classification based on a precise summary of what the code does as a whole
Relational with Error [RE]	Classification has a relational description but code segment selection is not in agreement with description.
Advanced Multi-structural [AM]	Classification based on a summary of a fragment of the code.
Multistructural [M]	Classification based on two or more language constructs of the code
Unistructural [U]	Classification based on one language construct
Prestructural [P]	Classification is related to a code characteristic such as position on page or number of lines of code but not to an understanding of the code.
Unclassified [X]	Classification did not have any relationship to the code.
Blank [B]	No classification performed

contains a for loop or *i = 0*. This reflects a focus on the code as written with no interpretation.

- Multistructural responses were based on combining two unistructural responses. An example is *initialising i before loop*. This is focussed on two aspects related to the loop that appear in separate statements in the code rather than just the language construct.
- Advanced Multistructural responses showed an ability to recognise what a small segment of the code is doing, and thus recognise some of the details of the implementation. Such responses tended to focus on the operation of the loop with statements such as *test from the beginning of the array* or *from first to last (forward)*. These statements recognise the operation of the loop control constructs and that it is being applied to an array of values.
- The relational responses were either *find maximum* or *find minimum*. These statements show an ability to summarise the code segment in total. The student clearly recognised the purpose of the code segments.
- The relational with error responses were classifications that had a relational description but the application of the classification to the code segments did not agree with the description. The student seemed to recognise the purpose of the code but then confuse the detail when classifying. For example, a number of students correctly realized that some code segments found the maximum, and others the minimum, but they incorrectly identified which segments did each. This error can at least partly be attributable to answering the question under exam conditions. Others made statements at a relational level, but which were wrong such as stating the code counted the number of elements in the array.

4 Results

A total of 110 students from two institutions comprising three class groups are included in this analysis. All the students who completed the question sets were either completing their first programming course or beginning their second programming course at their respective institutions.

Question sets in a range of programming languages were prepared by institutions to match the programming languages being taught at those institutions (Java and Delphi).

The two institutions whose data is part of this study used the question set as part of an assessment exercise for students either completing or towards the end of their first programming paper or in the first week of their second semester of programming during 2005.

4.1 Could the students classify?

In examining four code segments, do the novice programmers know how to identify similarities and differences? An indicator of whether the students could apply a classification strategy could be based on the number of blank (B), unclassified (X) and prestructural responses (P).

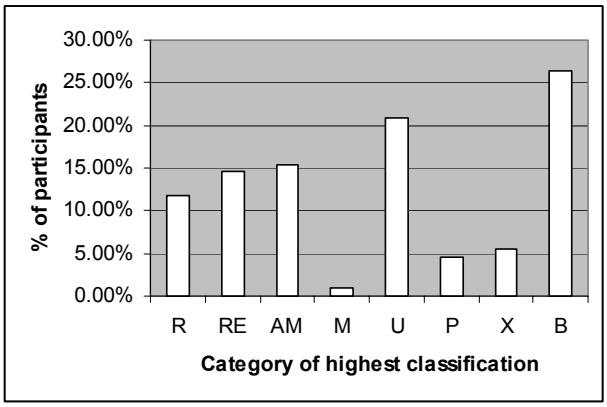

Figure 1: Distribution of question 11 responses by highest SOLO category (N=110)

The blank (B) responses (27% - see Figure 1) are the number of students who did not attempt to answer the classification question. This percentage is much higher than non-response rates for the nine multiple choice questions (< 2%) and question 10 (< 4%). Because the authors do not have access to think-out loud data for these students, it is difficult to determine why they made no attempt at the question. This low response rate may be because questions 11 was the last question in the paper and students ran out of time. Alternatively the lower

response rate may be an indicator of the level of difficulty that students felt with this question.

The proportion of the students within each quartile of performance on the MCQ scores (25% of students in each quartile based on their score in the MCQs – that is questions 1-9) indicates that 46% of the lowest quartile (Q4) students did not attempt question 11 (see Figure 2). For the other quartiles, this percentage is much lower, closer to 20%. This may indicate that those students who were struggling with the exam may have lacked the time to answer Q11.

It is interesting to consider the relationship between no-response and prestructural responses for questions 10 and 11. These responses for the "explain in plain English" question are more likely to produce no response (B) to the classification question (see Figure 3). However, only a small number of students gave such responses to Q10 (see Figure 7), so it is difficult to draw firm conclusions.

Whereas a non-response to Q11 may indicate either an inability to classify, an uncertainty of what to do, or simply a lack of time to complete the task, the prestructural response type (9% see Figure 1) and "X" response types (4% see Figure 1) are more readily attributable to difficulty with performing the task.

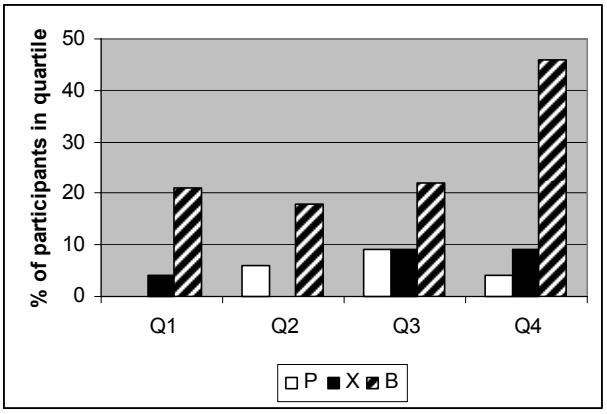

Figure 2: Percentage of prestructural (P), unclassified (X), and blank (B) responses in each quartile (N=110)

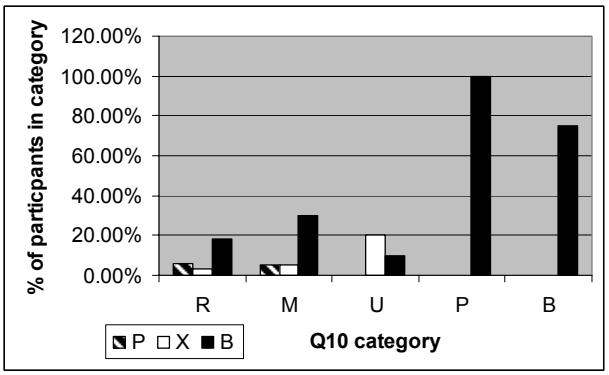

Figure 3: Percentage of P, X, and blank responses to question 11 for each response category for question 10

Prestructural (P) and unistructural (U) classifications are more common in the lower three quartiles (see Figure 2). However, only a small number of students gave such responses, so it is difficult to draw firm conclusions.

With 68% of the students providing non-blank, non-X, classifications, it would appear that classification is possible for novice programmers.

4.2 How did they classify?

The total number of classifications was 319 at an average of 3.94 classifications per student (N = 81, standard deviation 1.86). This excludes the students who did not attempt to classify (B). The maximum number of classifications for any student was ten classifications. Most of the supplied classifications were categorised as unistructural (51.7% see Figure 4).

A unistructural (U) classification is based on a language construct rather than the operation of more than one part or all of the code. As there are more language constructs in the code that there are relational descriptions of the code, we shouldn't be surprised to see this category dominant in the overall classifications for this question. However, 21% of the students had a unistructural classification (U) as their highest classification – that is they gave only unistructural or prestructural and unistructural responses. This category was only behind no response (B) as the highest classification category used (see Figure 1). Does this mean that this question tends to promote a unistructural response or is a unistructural classification easier to identify than a more complex multistructural (AM, M) or relational (R, RE)?

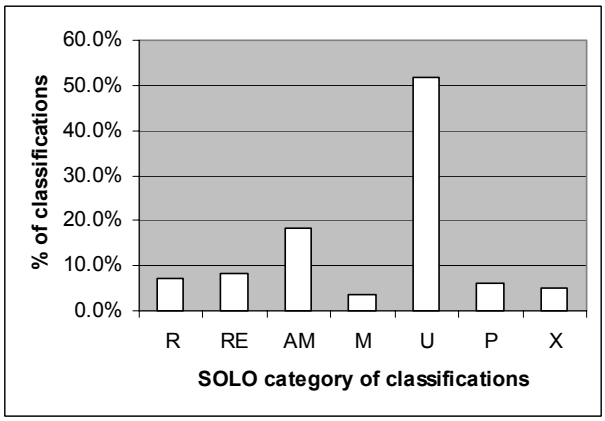

Figure 4: Distribution of SOLO Categories for Q11 (N=110)

A preferred response would be to see the relational (R) category dominating the highest classification used and the other categories (M, and U) represented strongly in the overall distribution. This would indicate an ability to both recognise the purpose of the code segment and to recognise the detail of the code. From the perspective of using code examples in teaching, this would mean that novice programmers recognised what made the code different even though they recognised that the code achieved the same objectives.

There is an indication that the students were moving toward this profile when the relational (R) and relational with error (RE) are combined. Together they represent 26% of the highest responses (see Figure 1).

The advanced multistructural (AM) category also has an element of a relational understanding. In this category, the students have recognised that the way that the loop is constructed impacts the direction in which the array is

processed. Students for whom AM was the highest category (15% - see Figure 1) failed to interpret the process carried out within the loop as finding the maximum or minimum.

Our results indicate that a significant number (63%) of the students could identify valid criteria to classify the code segments. Not all of these accurately assigned the code segments based on the identified criteria. Although 21% classified on the basis of a language construct, a reasonable proportion (26%) used or attempted a relational classification.

4.3 Depth, width and range of classifications

Another way of looking at a set of responses from a given student is to examine the highest SOLO response given by the student (depth), the number of classifications provided by a student (width) and the number of different classification categories (range).

The average width varied according to depth (see Table 2), with the lower classifications having a lower average width. However, the standard deviations are large, so there is not a clear trend in average width as a function of depth.

If a relational response is a demonstration of better understanding, there might be an expectation that the average range of categories per student would be higher for the relational category than for the other categories (see Figure 5). This would also reflect the fact that a lower highest category has fewer categories that are even lower. In Figure 5, the two relational categories have a slightly higher average range, and there is possibly a trend down to the unistructural category. However, it is difficult to see any major differences in the categories shown. It would appear that students, regardless of their highest classification category, work within a limited range of classification types. The average for all students was 1.8.

Table 2: Width of classifications. "M" omitted because of insufficient data.

Highest classification	Average	Min	Max	StdDev
R	4.46	2	6	1.08
RE	4.63	2	10	1.93
AM	3.65	1	7	1.78
U	4.04	2	9	1.94
P	2.60	1	4	1.20
X	2.17	1	5	1.46

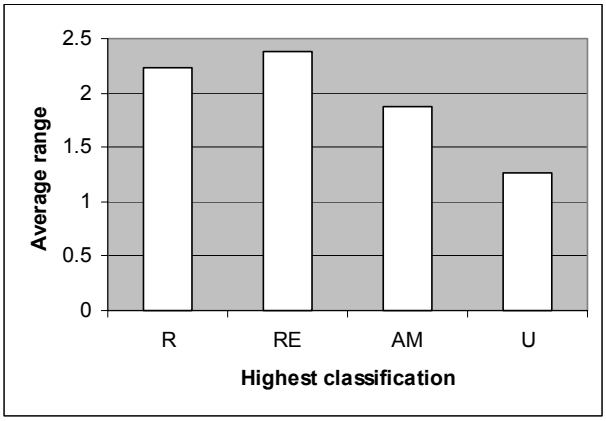

Figure 5: Average range of classifications

4.4 Recognising inverse classifications

For the particular code segments in Q11, every classification has an inverse classification – that is the classification that includes those code segments that are not part of the initial classification. For example, for the relational classification "finds minimum", the inverse is "finds maximum". For the classification "for loop", the inverse is "while loop". For the classification "incrementing", the inverse is "decrementing". Although the question clearly states "Select criteria where at least one of the code segments would be classified differently to the others", some students still used classifications that had no inverse or did not include the inverse classification. The average number of inverses is 1.5. In some cases, the student's classifications were all very closely related.

There does appear to be a trend of recognising the inverse classifications based on the highest classification used. The few students whose highest classification was prestructural (P – 4%) or unclassified (X – 9%) may make the reported averages meaningless for these particular categories.

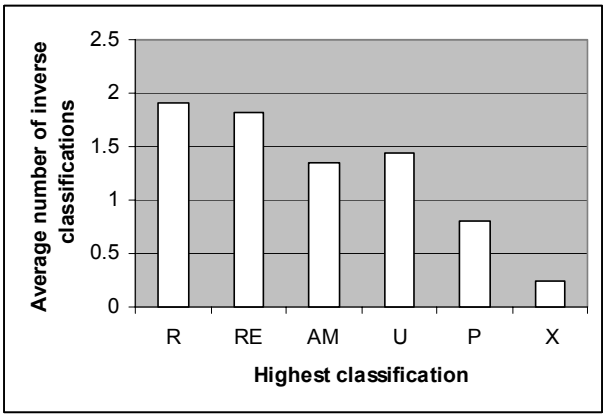

Figure 6: Average inverses by highest classification

4.5 How did this compare with question 10?

The most noticeable differences in the categories for Q10 and Q11 are observed for the multistructural (M) and unistructural (U) categories. There are fewer multistructural responses to Q11 (see Figure 4) than for Q10 (see Figure 7). To classify the code segments for

Q11, it isn't necessary to describe the whole code segment. Even a small feature of a code segment can be a point of difference.

In contrast it could be argued that the "explain in plain English" question required an explanation of each line of code thereby placing an emphasis on multi-structural or relational type descriptions. A multistructural answer for Q10 represents a set of unistructural descriptions for each line of the code. This grouping of unistructural answers into a multistructural response is not a requirement for Q11.

The difference in distribution of all classifications and the highest classification for Q11 and the distribution of categories for Q10 would appear to support this argument.

This difference in distribution of categories between Q10 and Q11 may also be a result of the difference in the cognitive skill level of the questions, as defined by the revised Bloom's taxonomy. The cognitive skill level of the Q10 is "Understand: Summarising", whereas the cognitive skill level for Q11 is "Understand: Classify". Classify is a lower level cognitive process than summarise within the revised Bloom's taxonomy. Students needed to both summarise and classify the code segments in order to reach higher SOLO categories. Identifying a unistructural (U) or prestructural (P) category avoided any summarisation of the code or identification of the purpose of the code.

4.6 Consistency in Relational Thinking

It is interesting to compare the performance of students on Questions 10 and 11, to see what percentage of students respond relationally to both questions. The results are summarized in Table 3.

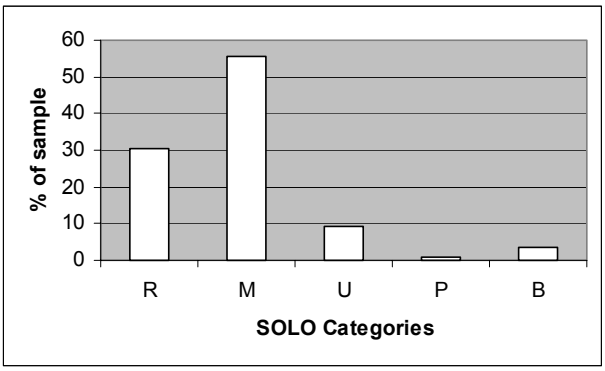

Figure 7: Distribution of SOLO categories for Q10

Table 3: Comparson of relational and non-relational answers for Q10 and Q11 (N=78)

Q10 Relational	Q11 Relational	Percentage Non-blank Responses
No	No	44%
Yes	Yes	14%
Yes	No	20%
No	Yes	22%

Of the students who provided a non-blank answer to both questions (N=78), 58% were consistent over both questions. That is, 58% of students either gave a relational answer to both questions, or a non-relational answer to both questions. Furthermore, 44% of the students gave non-relational answers to both questions, suggesting that these students consistently have difficulty in identifying the function of a piece of code.

Of the students who gave a non-relational answer to the "explain in plain English" question, only 22% gave a relational answer to Q11. On that basis, it would seem that the instruction "explain in plain English" may be ambiguous to a degree, but it is a reasonably reliable indicator of a student's ability to respond relationally.

Conversely, of the students who did not give a relational answer to Q11, only 20% gave a relational answer to the "plain English" question. On that basis, it would seem that both Questions 10 and 11 are reasonably reliable indicators of whether a student can extract meaning from a short piece of code.

5 Using variations in teaching

From a phenomenographic perspective, the space of learning is defined by the variations experienced by the learner (Marton, Runesson, and Tsui. 2003). If the learner does not experience variation in a dimension then they will not discern that dimension and consequently not learn what is required. Providing the appropriate variations in learning determines what is learnt and therefore the space of learning.

Variations in how to program a particular algorithm opens up an understanding of the efficiencies and differences in approaches to writing code. Using a classification question such as Q11 gives the educator the opportunity to see how the learners understand the differences in the code.

If learners are able to recognise different ways of implementing the same algorithm then the learner would use valid relational classifications. If these are provided along with advanced multistructural, multistructural, and unistructural responses, then the learner is also recognising the differences in the implementation of the algorithms. The educator may feel confident that these learners could comfortably learn from variations in sample code.

In contrast, a learner whose focus is on unistructural classifications is focussed on coding constructs and is less likely to recognise that different implementations of an algorithm are implementing the same algorithm using a different approach.

If the highest classification manifested by a student is multistructural then the learner is possibly beginning to see how coding structures work together.

6 Conclusion

This paper has analysed student responses to a classification question that was presented as part of the BRACElet multi-institutional investigation into reading and comprehension skills of novice programmers. The results have shown that most students were able provide classifications for code segments and were able, with

some degree of success, to apply the classification to the code segments. However, more than one quarter of the students studied did not attempt this question, thus raising the open question as to whether these students had the skills to answer this type of question.

When the classification question is compared with the results of the "explain in plain English" question there is a difference in the number of multistructural responses and the unistructural responses. This may be more of an indication of the question structure rather than of the students' ability to perform at these levels. With the degree of consistency in relational responses by students between Questions 10 and 11, it appears that both questions types are reasonably reliable indicators of whether a student can extract meaning from a short piece of code.

The analysis presented in this paper does highlight a number of interesting questions for future study, including:

1) Why did so many students not attempt to answer the classification question? Did they not understand what was required or is it a task that they could not perform? Was it merely an issue of time?
2) Would the percentage of students who attempted the classification question increase if the students had been more familiar with performing this type of task?
3) Can a novice programmer's ability to recognise patterns be fostered through the use of variations in code and classification type questions?
4) Would providing alternative code samples help build the students ability to classify code segments relationally?
5) How would developing the ability to recognise similarities and differences in code help a student's understanding and writing of programs?
6) Would a student who is able to identify the purpose of a code segment (a relational classification) in one language also recognise a code segment in a different language that has the same logic structure and is achieving the same task?

7 References

Adelson, B. (1984) When novices surpass experts: The difficulty of a task may increase with expertise. Journal of Experimental Psychology: Learning, Memory, and Cognition, 10, 483-495.

Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Raths, J. & Wittrock, M. C. (Eds.) (2001) *A taxonomy for learning and teaching and assessing: A revision of Bloom's taxonomy of educational objectives*, Addison Wesley Longman.

Biggs, J. B. (1999) *Teaching for quality learning at University,* Buckingham, Open University Press.

Biggs, J. B. & Collis, K. F. (1982) *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome),* New York, Academic Press.

Chase, W. C., & Simon, H. A. (1973). Perception in chess. In W. G. Chase (Ed.), Cognitive Psychology, 4, 55-81.

Chi, M. T. H., Glaser, R. & Farr, M. J. (Eds.) (1988) *The nature of expertise,* Hillsdale, NJ, Lawrence Erlbaum Associates.

Fitzgerald, S., Simon, B., Thomas, L. (2005) Strategies that Students Use to Trace Code: An Analysis Based in Grounded Theory. 1st International Workshop on Computing Education Research. Seattle, WA USA, 2005.

Lister, R., Adams E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppällä, O., Simon, B. and Thomas, L. (2004): A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *SIGSCE Bulletin*, 36(4):119-150.

Lister, R., Simon, B., Thompson, E., Whalley, J. & Prasad, C. (2006) Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. *Innovation and Technology in Computer Science Education (ITiCSE 2006)*. Bolonga, Italy.

Marton, F., Runesson, U. & Tsui, A. B. M. (2003) The space of learning. In Marton, F. & Tsui, A. B. M. (Eds.) *Classroom discourse and the space of learning.* Mahwah, NJ London, Lawrence Erlbaum Associates, Publishers.

Whalley, J., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, A. & Prasard, C. (2006) An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. In Tolhurst, D. & Mann, S. (Eds.) *Eighth Australasian Computing Education Conference (ACE2006)*. Hobart, Tasmania, Australia, Australian Computer Society Inc, CRIPT, **52**, 243-252.

8 Acknowledgements

The data used in the analysis for this paper was collected by a number of dedicated programming educators in a number of institutions. Thanks to Gordon Grimsey, Christine Prasad, Ajith Kumar and Phil Robbins for their contribution to the data collection phase.

We also wish to acknowledge the contributions of the entire BRACElet team in the development of the study instrument as well as their open sharing of ideas and many helpful discussions.

9 Appendix 1

This is question 11 in full from the Java version of the question set. The layout of the question varied based between versions of the question set. In one version, the four code segments where presented in two columns. The layout here is to fit the journal format.

Question 11:

Consider the following 4 code segments:

A

```
int[] a = {1,5,6,2,3,9};
int m = a[0];

for(int i = 0; i < a.length; i++)
{
      if(m < a[i])
         m = a[i];
}
```

B

```
int[] a = {1,5,6,2,3,9};
int m = a[0];
int i = 0;

while(i < a.length){
   if(m > a[i])
      m = a[i];
   i++;
}
```

C

```
int[] a = {1, 5, 6, 2, 3, 9};
int m = a[a.length-1];

for(int i = a.length-1; i >= 0;  i--)
{
      if(m < a[i])
           m = a[i];
}
```

D

```
int[] a = {1,5,6,2,3,9};
int m = a[a.length-1];
int i = a.length-1;

while(i >= 0)
{
      if(m <= a[i])
          m = a[i];
      i--;
}
```

There are a number of ways in which these code fragments can be classified. Suggest different criteria that could be used as a basis for classification, and for each criterion tick the fragments that you believe fall into that classification criterion.

10 Appendix 2

This is question 10 in full taken from the Java version of the question set.

Problem 10

Look at this section of code then describe in plain English what it does.

x is an array of integers.

```
bool bValid = true;

for (int i=0; i < (x.Length - 1); i++)
{
      if(x[i] > x[i+1])
        bValid = false;
}
```