

One Small Step Toward a Culture of Peer Review and Multi-Institutional Sharing of Educational Resources: A Multiple Choice Exam for First Semester Programming Students

Raymond Lister

Faculty of Information Technology
University of Technology, Sydney
PO Box 123, Broadway, NSW 2007, Australia

raymond@it.uts.edu.au

Abstract

This paper presents a multiple choice question exam, used to test students completing their first semester of programming. Assumptions in the design of the exam are identified. A detailed analysis is performed on how students performed on the questions. The intent behind this exercise is to begin a community process of identifying the criteria that define an effective multiple-choice exam for testing novice programmers. The long term aim is to develop consensus on peer review criteria of such exams. This consensus is seen as a necessary precondition for any future public domain library of such multiple-choice questions

Keywords: Introductory programming, CS1, Java, digital library, peer review, multiple-choice question.

1 Introduction

With the advent of the Internet and World-Wide Web there has grown a widespread belief that educational resources – lectures notes, lab exercises, assessment items – might be authored at one institution, then made available for reuse at other institutions. The technology exists to do this, and such learning repositories (or digital libraries) have been implemented (e.g. Bell and Schauder, 2003). However, there has been very little sharing of educational materials through these facilities.

Technological feasibility is just one factor in the widespread adoption of a software product. The author of this paper believes that learning repositories have not gained widespread use because we lack the necessary cultural framework. In parallel with the development of the technology, we also need to develop a culture of inter-institutional peer review of educational materials, which in turn requires agreed conceptual frameworks, and a shared vocabulary.

There have been several attempts to develop formal peer review approaches for educational materials in our discipline (Grissom, *et al.*, 1998; Joyce, *et al.*, 1997; Knox, *et al.*, 1999). These attempts did not fail. Rather,

these were the ground-breaking attempts in what will be a long process of cultural change. These early attempts approached the problem in a top down fashion. That is, the focus was on the identification of general criteria for evaluating educational materials, based upon the participants' years of experience of developing materials, without a strong reference to specific educational materials. A top down approach is not wrong, but there is a complimentary bottom up approach, where specific educational materials are discussed and criteria emerge from that discussion. That bottom up process has been largely neglected.

For most practising teachers, much of their teaching is based on implicit assumptions ("a fish is not aware of water" – Chinese proverb). If we are to engage in constructive discourse on reusable educational resources, across institutions, then we will need to learn how to make our assumptions explicit. The bottom up approach is particularly useful for eliciting implicit assumptions. When experienced and capable academics find they differ in their overall subjective assessment of a specific educational item, then the opportunity arises to identify the differing educational assumptions that led to their difference of opinion.

In this paper, we expose an assessment item to the bottom up process of developing evaluating criteria. The item is a multiple-choice exam for first semester Java programming. Space limitations do not allow the complete exam to be included, instead a representative set of questions are presented.

If assessment items are to be stored in repositories, then we should also store historical data on how students have performed on those assessment items. We will therefore need to develop agreed conceptual frameworks for evaluating this sort of historical performance data. This paper will explore such a conceptual framework for multiple-choice exams.

2 Identified Assumptions

In debating educational materials, academics need to distinguish explicitly between two types of debate. In one debate, it is the assumptions behind the educational material that are at issue. In the other debate, the assumptions are accepted and the debate is about whether the material is a valid rendering within the conceptual framework defined by those assumptions. From the author's experience, the least productive debate is when the real difference of opinion is about the assumptions, but the rhetoric concentrates on specifics of the

educational material, and when the assumptions are made explicit, they are articulated as self-evident truths.

2.1 Objects-Early versus Objects-Etcetera

This exam is for a subject called “Object-Oriented Programming”, and objects are introduced in the first lecture. However, this subject reflects a world-wide tension in our community, as many academics argue for the retention of traditional 3GL syllabus objectives. In addition to introducing basic object-oriented concepts, the subject also introduces standard, introductory, iterative processes on arrays, including the basic sorting and searching algorithms.

2.2 Mastery versus Development

In teaching, we sometimes aim to ensure that all students who pass have attained a minimum acceptable level of competence. At other times, we aim to give students the opportunity to perform to the very limit of their ability. The former aim is concerned with “mastery”, and the latter aim is concerned with “development” (Linn and Gronlund, 1995).

The exam in this subject is aimed at testing mastery. No question in the exam is aimed at development. Other assessment tasks in the subject are aimed at development. The overall assessment regime has been described in an earlier paper (Lister and Leaney, 2003).

2.3 Reading versus Writing

A 2001 ITiCSE working group assessed the programming ability of students across several countries (McCracken, 2001). Students were tested on their ability to write working code for a common set of programming problems. The majority of students performed poorly. Most students did not get close to solving the set task. Whereas a similar report from an author at a single institution might be dismissed because of poor teaching at that institution, it is difficult to dismiss a multinational study. Therefore, in designing this subject and its exam, the author assumes that an appropriate mastery test should not test the ability of students to write code, but instead focus upon their ability to read code. Fincher (1999) described this as a “literacy” approach to teaching programming.

2.4 Criterion Referencing and Objectives

Given the issues discussed in the above three subsections, the exam objective is that any student who passes will have demonstrated that they can:

- (a) Comprehend basic OO programming concepts of classes, instances, events, and methods.
- (b) Comprehend basic program control constructs of sequence, selection, and iteration.
- (c) Comprehend code that implements the basic sorting and searching algorithms on arrays.

Setting objectives that passing students should meet is known as criterion referencing.

2.4.1 Three Types of Questions

The author believes that each of the above three exam objectives requires its own type of multiple-choice

question. However, the second and third objectives are closely related, and so questions for those objectives can be similar in style.

There are 26 questions in the exam. The first thirteen are “object-oriented” questions, to test the first exam objective. The remaining thirteen “array questions” test the second and third exam objectives. Of the thirteen array questions, the first five concentrate on the second exam objective. These five questions test the students on “unseen array” code. That is, students are tested on code they have never seen before, where the code performs some sort of iterative process on arrays. The final eight questions test students on “seen array” code. That is, students are tested on code they have seen in class before, which implements basic sorting and searching algorithms on arrays.

2.5 Pass Mark

The author does not believe that the traditional 50% pass mark is appropriate for a multiple-choice exam that emphasises mastery. A student who can only answer half the questions correctly does not have sufficient grasp of the material to cope with the next subject. Therefore, the pass mark for the exam is set at 70%, which is 18 correct answers out of the 26 questions.

2.6 Norm- versus Criterion-Referencing

At the author’s institution, the informal faculty policy is that no more than 30% of students should fail a subject. Specifying a target figure for the percentage of students who should achieve a certain grade (including the failing grade) is known as norm referencing.

Strictly speaking, in subject design we should choose to either grade by norm referencing or by criterion-referencing. The two approaches are formally incompatible. In practise, academics are routinely instructed to meet both types of criteria – whether they do so is as much a question of good luck as good judgement.

This exam did not meet its norm-referencing criterion, as just over half the class scored less than 18 out of 26. To meet the norm-referenced target failure rate, the pass mark would need to be reduced to the more traditional 50% mark (i.e. 13 out of 26). Therefore, analysis of the exam will entertain both the 70% and 50% pass mark.

2.7 Student Preparedness for the Exam

Students sitting this exam had access to three, similar exam papers from earlier semesters. Indeed, throughout semester many tutorial questions were taken from these past exam papers. Therefore, the exam was designed on the assumption that students were very familiar with this type of multiple-choice exam.

2.8 Memorization Requires Abstraction

For decades, academics who teach novice programmers have despaired at the many students who cannot even reproduce, in an exam, short pieces of code that were taught during semester. A common theory is that these students did not apply themselves to their study. There is merit to that theory, but we also know that failure rates in programming are higher than in the other subjects

students study simultaneously. It seems unlikely that students value a pass in programming less than a pass in other subjects.

The author believes that most academics teaching programming fail to appreciate the high cognitive load in reading and memorizing code for novice programmers. Academics barely notice the syntactic minutia, individual lexical symbols, or even the exact coding of a test condition in loop, because we read, understand, and memorize code at a more abstract level. We see the intent of the code, the plan, or “schema”. When called upon to reproduce such code, we reconstruct the code from the abstraction.

Novice programmers, on the other hand, particularly those novices who are struggling in their first semester, do not see intent, plans, or schema. They see only the concrete code. At that concrete level, memorizing code is a prodigious task.

The author believes that we need to place greater emphasis on teaching students to read code in the more abstract way that academics do. In this context, testing students on their ability to recall code is legitimate. If students have been shown a great deal of code during semester, then memorizing it at the concrete level becomes near impossible. A student can only recall that code in the exam if they have learnt it at the more abstract level, and reconstruct it. In the terms of educational psychology, the conceptual framework that led to this exam paper distinguishes between meaningful reception learning (i.e. what is advocated here) and rote learning (Lefrancois, 1999, pp. 213-219).

3 Overview of the Exam and its Analysis

This section reviews the concepts and techniques used to analyse multiple-choice questions in later sections.

3.1 Overall Class Performance

Central to the multi-institutional sharing of an assessment item will be the question, “How hard is this assessment item for my students?” An effective learning repository will need to include historical information about prior use of assessment materials.

Figure 1 shows the performance of the whole class on each of the 26 multiple-choice questions. There is greater variation in class performance on the first 13 object-oriented questions than the array questions, and in general, the class did better on the array questions than the object-oriented questions. Within the array questions, the class did a little better on the “unseen” questions than the “seen” questions.

3.1.1 Quartile Analysis

In the class of 336 students, marks on this exam ranged from the maximum possible 26 down to a mark of 4. An analysis of aggregate class performance, as shown in Figure 1, groups students of very different capabilities.

There is a well-developed literature on the analysis of multiple-choice exams (Ebel & Frisbie, 1986; Linn & Gronlund, 1995). Much of that literature focuses on dividing students into quartiles, based on their performance on the whole exam. Table 1 shows the

quartile boundaries for this exam. However, much of that literature on multiple-choice questions is intended for norm-referencing scenarios. The exam under study in this paper is intended to test mastery, so it is not clear how much of the standard multiple-choice literature should be applied to this exam.

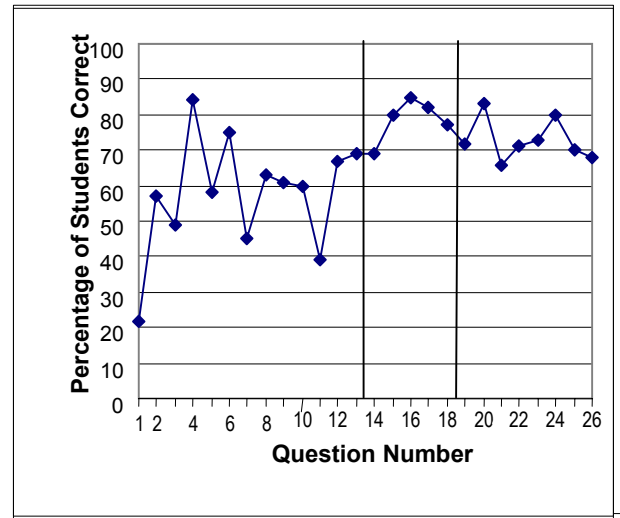


Figure 1: Percentage of class (N=336) who answered each question correctly. Questions to the left of the first vertical line are object-oriented questions. Questions between the vertical lines are “unseen array” questions. Questions to the right of the second line are the “seen array” questions.

Quartile	Score Range	No. of Students	Percent of Students
1st “top”	22 – 26	81	24%
2nd	19 – 21	82	24%
3rd	14 – 18	81	24%
4th “bottom”	4 – 13	92	27%

Table 1: The quartile boundaries for students on the 26 multiple-choice questions (N=336).

When an exam is intended to test mastery, perhaps the quartile information of most interest is the performance of the upper two quartiles. In Figure 2, it can be seen that the top quartile consistently performed above 90% on the array questions, and the second quartile consistently performed above 80% on the same questions. While not committed to the specific figures of 90% and 80%, the author believes that the high performance of these two quartiles is justification that the array questions are a reasonable test of mastery. Both of the upper two quartiles performed less consistently on the object-oriented questions, suggesting that this portion of the exam is a less reasonable test of mastery. (Not that the author realistically expects all questions of any exam to be good mastery questions; if they were then the pass threshold should be higher than 70%.)

3.1.2 Bottom Passing and Fifty-fifty Students

In a criterion-referenced mastery exam, the author proposes that students of particular interest are the

students who just “scrape” a pass, so it needs to be established that they have learnt enough to be ready for the next subject. To have a reasonable sample size, the author proposes to study at least 30 such students. There were 24 students who scored exactly 18, and another 18 students who scored 19, so this combined set of 42 “bottom-passing” students will be studied.

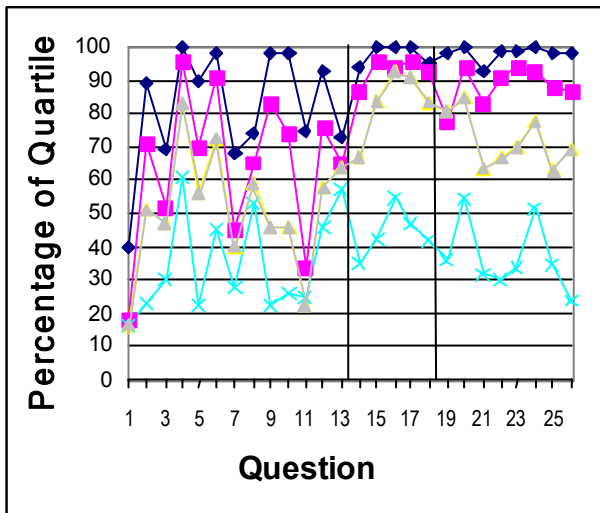


Figure 2: Percentage of students who answered each question correctly, by quartile. Diamonds, squares, triangles, and crosses represent the quartiles, from top to bottom, respectively.

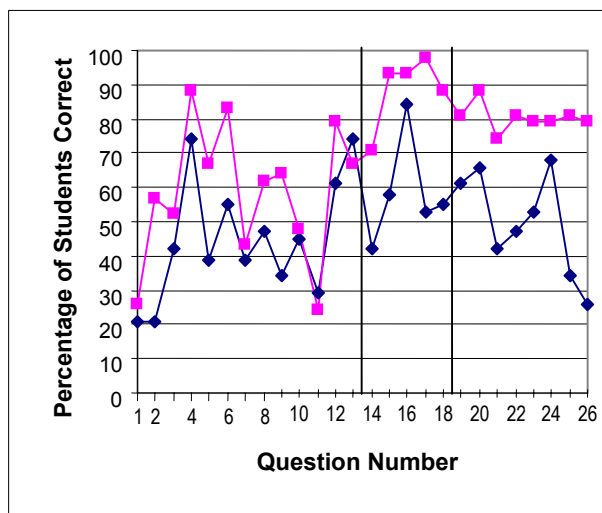


Figure 3: Percentage of bottom passing students (mark of 18-19), represented by squares, and fifty-fifty students (mark of 12-14), represented by diamonds, who answered each question correctly.

As mentioned earlier, informal faculty policy is that no more than 30% of students should fail a subject, but to meet that criterion, the pass mark would need to be lowered to 13 (i.e. 50%). Therefore, students who scored around that mark are of interest. They are also of interest because a 50% pass mark is traditional in academia, and a design assumption for the exam was that a 50% pass mark is too low to test mastery. The author elected to study the 38 “fifty-fifty” students who scored 12-to-14 in the exam.

Figure 3 shows the performance of bottom passing and fifty-fifty students on each multiple choice question.

4 Questions 1 to 7: The Account Class

The first seven questions of the exam all tested basic object oriented concepts within the context of a given class, called “Account”. Students had not seen “Account” before the exam. The code for the class was given in the exam paper, with some lines missing,:

```
public class Account {
    private double balance;
    private static double rate;
    private int accountNumber;
    private xxx1xxx lastAccountNumber;

    public Account() {
        this(100.00);
    }

    private Account(double b) {
        balance = b;
        lastAccountNumber++;
        accountNumber = lastAccountNumber;
    }

    xxx2xxx getAccountNumber() {
        return accountNumber;
    }

    xxx3xxx deposit(double amount) {
        balance = balance + amount;
    }

    xxx4xxx getBalance() {
        return balance;
    }

    xxx5xxx setRate(double newRate) {
        rate = newRate;
    }
}
```

In the remainder of the paper, whenever an individual question is analysed, it will appear under a sub-heading, and the question itself will be given immediately after that sub-heading. Commentary on the question will follow the question itself.

4.1 Questions 1 and 2: Constructors

The questions tested constructors. Question 1 proved problematic, so it will be discussed after Question 2.

4.1.1 Question 2

Consider the following line of code:

```
Account a = new Account();
```

If the missing parts of “Account” were filled in correctly, and “Account” had been compiled, then the above line of code would:

- (a) generate errors when it is compiled.
- (b) compile but will generate a run time error when it is executed.
- (c) compile and run, producing an instance of Account with a zero balance.
- (d) compile and run, producing an instance of Account with a one hundred dollar balance.

Only 57% of the whole class correctly chose option d. The same percentage of bottom passing students answered correctly, but only 21% of the fifty-fifty students answered correctly. However, 90% of the top quartile correctly chose option d. Most students who answered incorrectly chose distractor c.

4.1.2 Question 1

Question 1 was almost identical to Question 2, except in Question 1 the given line of code was:

```
Account a = new Account(100.00);
```

The four options for Question 1 were the same as for Question 2. Only 22% of the whole class chose option a, which is correct because the constructor with a parameter is declared “private”. Instead, most students chose distractor d. Only 40% of the top quartile answered correctly. This was the only question in the exam where a minority of top quartile students answered incorrectly. It could be argued that this question depended upon a detail too subtle for students to detect under exam conditions. However, it should also be noted that 21% of the fifty-fifty students answered correctly, exactly the same percentage as for Question 2. Therefore, the primary difficulty for fifty-fifty students was the concept of a constructor, not the use of the “private” keyword. (Also, see comments about “public” and “private” in the discussion of Question 5.)

5 Question 3: Static Data Members

The purpose of “lastAccountNumber” is to generate a new value for “accountNumber” each time an instance of “Account” is created. Therefore, the “xxx1xxx” in the declaration of “lastAccountNumber” should be replaced by:

- (a) int
- (b) static int
- (c) void
- (d) static void

While the “static” keyword received considerable attention during semester, this particular use of the keyword, to generate a unique identification number for each of instance of a class, had not been given as an example. Approximately 70% of the top quartile correctly chose option b, as did 52% of bottom passing students, and 42% of fifty-fifty students. However, any student with some grasp of the question could have eliminated

distractors c and d, as the use of “void” to declare a variable is nonsensical (and distractor a was by far the most popular distractor). Therefore, for many students, the question may have come down to a random choice between two options.

5.1 Questions 4 to 7: Accessors and Mutators

In any introduction to object-oriented programming, students are drilled extensively in technique of declaring data members private and accessing or mutating them via public methods. In this particular subject, this common principle is taught to students as the “golden rule”.

5.1.1 Question 4

If we follow the “golden rule”, the “xxx2xxx” in the getAccountNumber method should be replaced by:

- (a) public void
- (b) private void
- (c) public static void
- (d) public int
- (e) private int

This proved to be the easiest of the thirteen object-oriented questions, with 84% of the whole class correctly choosing option d. Almost all (88%) bottom passing students answered correctly, and 74% of fifty-fifty students also answered correctly. Even 60% of the bottom quartile answered this question correctly, which may in fact be a source of concern – given the commonality of “get” methods in object-oriented programming, it is possible that students were merely choosing the most familiar option, without understanding why it is correct. The weaker students performed much worse on other “get” and “set” questions in the exam (see below), evidence that these weaker students do not genuinely grasp the concepts behind this question.

5.1.2 Question 5

If we follow the “golden rule”, the “xxx3xxx” in the deposit method would be replaced by:

- (a) public void
- (b) private void
- (c) public static void
- (d) public int
- (e) private int

Only 58% of the whole class answered this question correctly. Of the top quartile, 90% correctly chose option a, while 67% of bottom passing students also chose correctly, but a mere 39% of fifty-fifty students answered correctly. Distractor b was most popular by a considerable margin, indicating that students who chose it have a weak grasp of the significance of the “public” and “private” keywords.

(The popularity of distractor b in this question also suggests that, in Question 1, students did not simply fail to notice that the constructor was “private”, but instead they did not understand the significance of “private”.)

Given the success students had in correctly answering the previous and next questions, which are closely related to

this question, it is disappointing that the class did much more poorly on this question. Part of the explanation may be the name of the method, “deposit”. Perhaps students did not see the connection between the three questions because the name of the method in this question does not begin with “set”, as mutator methods frequently did in examples throughout semester.

5.1.3 Question 6

If we follow the “golden rule”, the “xxx4xxx” in the `getBalance` method would be replaced by:

- (a) `public static void`
- (b) `public double`
- (c) `private double`
- (d) `public int`
- (e) `private int`

This question is similar to Question 4, and like that earlier question proved to be relatively easy, with 75% of the whole class correctly choosing option b. However, only 55% of fifty-fifty students answered correctly, as opposed to 74% of fifty-fifty students on Question 4. (With 38 students in that fifty-fifty group, the difference in these percentages represents 7 students). Students may have found this question harder than question 4 because this question involves a data member declared as “double”, whereas question 4 involved a data member declared as the more familiar “int”.

5.1.4 Question 7

If we follow the “golden rule”, the “xxx5xxx” in the `setRate` method above would be replaced by:

- (a) `public static void`
- (b) `public void`
- (c) `private void`
- (d) `public int`
- (e) `private int`

In this question, students once again demonstrate a weak grasp of the ramifications of the “static” keyword. Since data member “rate” is declared “static”, then its mutator method must also be declared “static”. Only 70% of the top quartile correctly chose option a. Bottom passing and fifty-fifty students performed poorly and similarly, 43% and 39% respectively. Option b was the most popular distractor.

6 Question 8: Strings and Object References

Consider the following code:

```
String s1 = "one";
String s2 = "two";
s1 = s2;
s1 = "three";
s2 = "four";
```

Immediately after the execution of the above lines, the two Boolean expressions

“`s1 == s2`” and “`s1.equals(“three”)`”

respectively evaluate to be:

- a) `false` and `true`
- b) `true` and `false`
- c) `false` and `false`
- d) `true` and `true`

This was one of the easier object-oriented questions, with 63% of all students correctly choosing option a. However, the performance gap between the four quartiles is narrow, compared with the gap on most other object-oriented questions. This question was one of the poorer object-oriented questions for the two higher quartiles (74% and 65% correct), but one of the better questions for the lower two quartiles (59% and 53%). Such a narrowing in the performance of the quartiles suggests that students chose an answer not from knowledge, but by guesswork. Perhaps many students reasoned shrewdly it was unlikely that the value of both Boolean expressions would be the same, narrowing the choice to only two options. Having made that guess, a student need only then be able to deduce the value of one of the two Boolean expressions.

Distractor b was chosen by 15% of students, distractors c and d were each chosen by 11% of students.

7 Questions 9 and 10: The Node Class

Questions 9 and 10 related to the class “Node”, which had been discussed in lectures during semester. At the preceding conference in this series, the author argued that linked lists should be introduced before arrays in any “objects early” introduction to programming (Lister, 2004). We therefore decided to try a tentative experiment with linked lists in this most recent semester. We taught linked lists, but then assessed it “gently” in this exam.

The class “Node”, as it was taught to the students, has two private data members:

```
private int theNumber;
private Node next;
```

A linked list of instances of class Node store integers, with integers in ascending order in the list. The insertion of a new integer into the list is done using the “insert” method of class Node, which takes a single integer parameter “num”. The body of that method is:

```
if ( theNumber == num ) return;

if ( theNumber < num ) {
    if ( next != null )
        next.insert(num); // xx1xx
    else
        next = new Node (num); // xx2xx
}
else {
    if ( next != null )
        next.insert(theNumber); // xx3xx
    else
        next= new Node(theNumber); // xx4xx
    theNumber = num;
}
```

Students were told that “Node” would be in the exam. Furthermore, they were told that any questions would

require them to nominate one or more of the four lines of code shown above in bold (i.e. in the exam, students were given all the above code, except the four lines in bold, which were replaced with the respective character sequences shown as comments on those four lines.)

The exam paper contained two questions on “Node”. The statistics for both questions are similar, so only one of these two questions is analysed here.

7.1 Question 10

The skeleton code of class “Node” contains:

xx4xx

The correct completion of this line is:

- (a) `next.insert(num);`
- (b) `next.insert(theNumber);`
- (c) `next = new Node (num);`
- (d) `next = new Node (theNumber);`

In nominating, prior to the exam, the four lines of “Node” that were examinable, the author practically begged the students to rote learn the answers. Perhaps the top quartile did exactly that, with 98% of them correctly choosing option d. The second quartile also fared well on this question, with 83% of them choosing correctly. However, bottom passing students and fifty-fifty students performed similarly and poorly, 48% and 45% respectively choosing the correct answer.

The relatively poor performance of bottom-passing and fifty-fifty students could be attributed to two possible explanations. The first explanation is that these students were so profoundly overwhelmed by the material in this subject that they could not even rote learn “Node”. This seems an unlikely explanation, as (even though the “Node” class was not discussed until late in the semester), all the concepts involved in class “Node” had been taught by week 4 of semester, and these students demonstrated mastery of those concepts in other questions. The second explanation is that these students simply did not study “Node” thoroughly before the exam. Further evidence for this explanation is the performance of these students on the “seen” array questions, which will be discussed later in the paper.

8 Question 11: Call by Reference vs. Value

Consider the following code:

```
public static void main(String[] args) {
    int[] x = {1, 2, 3, 4};
    int a = 1;
    increment1(a);
    increment2(x, a);
    for ( int i=0 ; i<x.length ; ++i )
        System.out.print(x[i] + " ");
    System.out.println();
}

private static void increment1(int i) {
    ++i;
}

private static void increment2(
    int[] a, int i) {
```

```
    a[i] = a[i] + 1;
}
```

If the “main” method of this class is executed, the outputted values are:

- a) 1 2 3 4
- b) 1 3 4 4
- c) 1 3 3 4
- d) 1 4 3 4

The purpose of this question was to test student understanding of call-by-reference and call-by-value. This is not an object-oriented concept, and in fact, the question involves an array, but the question is placed with the object-oriented questions because it tests the students on their conceptual knowledge.

The correct option is c. A student who mistakenly believes that arrays are passed by value would choose distractor a. A student who mistakenly believes that primitive integers are passed by reference would choose distractor b. A weakness of this question is that a student who believes that array subscripts start at one, and that primitive integers are passed by reference, would choose the correct answer.

The entire class performed poorly on this question, with only 39% correctly choosing option c. Only 75% of the upper quartile chose correctly. Bottom passing students and fifty-fifty students performed similarly and poorly, 29% and 24% respectively. Distractor a was the most popular, with 34% of the class choosing it (and 26% of upper quartile students). Distractor b was also popular with 20% of the class choosing it (but only 6% of upper quartile students).

9 Question 12: Inheritance and Interfaces

Consider the following incomplete statement: *The xxx1xxx reserved word means that an applet inherits public methods from its superclass xxx2xxx.* Which one of the following is correct:

- (a) The statement is true if xxx1xxx is replaced with “implements”, and xxx2xxx is replaced with “ActionListener”.
- (b) The statement is true if xxx1xxx is replaced with “implements”, and xxx2xxx is replaced with “Applet”.
- (c) The statement is true if xxx1xxx is replaced with “extends”, and xxx2xxx is replaced with “ActionListener”.
- (d) The statement is true if xxx1xxx is replaced with “extends”, and xxx2xxx is replaced with “Applet”.

This was an easier object-oriented question, with 67% of the class correctly choosing option d, and over 90% of the top quartile choosing correctly. Approximately 80% of bottom passing students chose correctly, but only approximately 60% of fifty-fifty students chose correctly. The three distractors were equally popular.

10 Question 13: GUIs and Events

In lectures and labs, we studied the “BMIApplet”. In that applet, a method called “addActionListener” was defined in:

- (a) the instances of TextField “heightInput” and “weightInput”.
- (b) the instances of Label “heightLabel” and “weightLabel”.
- (c) the instance of Button “computeButton”.
- (d) the BMIApplet class itself.

Like question 8, this question proved to be one of the easier object-oriented questions for the class, with 69% of all students answering it correctly. However, like question 8, the performance gap between the four quartiles is relatively narrow. This question was one of the poorer object-oriented questions for the two higher quartiles (73% and 65% correct), but one of the better questions for the lower two quartiles (64% and 57% correct). Such a narrowing in the performance of the quartiles suggests that students chose an answer not from knowledge, but by guesswork. Very few students chose distractors a or b, thus narrowing the choice to only two options. Option c is the correct choice. However, for many students the choice between options c and d may have rested in their appreciation of the exact meaning of the word “defined” in the question stem. The method name “addActionListener” appeared in the BMIApplet class, and the code within the standard class “Button” was never directly examined in class, so students may have chosen distractor d on that basis.

11 Questions 14 to 18: Unseen Arrays

As discussed earlier, the intent of these unseen questions is two-fold. First, these questions are to test a student’s skill in manually executing code (a “trace”, or “desk check”). That skill presupposes that the student is familiar with the various array and control constructs used in the question. Second, these questions serve as a check on whether students are merely rote learning the seen array questions (discussed later) without any understanding of the underlying array and control constructs – if a student did well on the seen questions but poorly on these unseen questions, then the student is likely to be rote learning.

The five unseen array questions are all very similar in concept. Furthermore, these types of questions were subjected to a very thorough analysis by a recent, large international project (Lister *et al.*, 2004). Therefore, it is sufficient in this paper to examine only one such question. Any reader wishing to look at these types of questions more closely should see the paper by the international project.

11.1.1 Question 18

Consider the following code fragment:

```
int[] x = {1, 2, 2, 2, 1, 1};
int count = 0;
int i = 0;
int j = x.length/2;

while ( j < x.length ) {
    if ( x[i] == x[j] ) ++count;
    else
        if ( x[i] < x[j] ) --i;
    else
```

```
        if ( x[i] > x[j] ) --j;
        ++i;
        ++j;
    }
```

After this code is executed, the variable “count” contains what value? You may use the table below to help you calculate the answer to the question.

- a) 4
- b) 3
- c) 2
- d) 1

Code	Comment	i	j	count
		0		0
... the table in the exam paper contained 30 blank rows, like the two rows above...				

This question was answered correctly by 77% of the entire class. More than 90% of students in the upper two quartiles correctly chose option c. Almost 90% of bottom passing students answered correctly. However, only 55% of fifty-fifty students answered correctly.

Distractor d was approximately twice as popular as the other two distractors.

11.1.2 Discussion of Unseen Array Questions

The author believes that students do better on the unseen array questions because they involve fewer concepts. If a student has the knowledge to answer one of these questions, then the student has the skills to answer all five questions. By comparison, most object-oriented questions involve at least one different concept from the other object-oriented questions.

From past exam papers, students knew there would be several unseen array questions in this exam. Any student who did study for this exam, not comprehensively, but shrewdly investing their limited study time, would do better on these five unseen array questions than on the rest of the exam paper. Figure 3 shows that bottom passing students did markedly better on these unseen array questions than other question types. That same figure shows that fifty-fifty students did not do markedly better on the unseen array questions than other question types. Therefore, the author concludes that the fifty-fifty students did not study effectively for this exam paper. The remedy for fifty-fifty students is not the development of more effective learning materials – they are unlikely to use those learning materials effectively. Improving the performance of the fifty-fifty students should first focus on changing their approach to study.

12 Previously Seen Array Questions

The final eight questions in the exam all concerned code that had been studied during semester. Furthermore, the students had also seen the multiple-choice questions. Weeks prior to the exam, the students were provided with

thirty questions, from which the eight questions were drawn for the exam paper.

Students were not required to memorize the algorithms. In the exam, students were provided with all the diagrams from lecture notes that explained each algorithm. There were 101 such diagrams provided in the exam! Any non- novice programmer who had never encountered these algorithms before would have been able to deduce the answers to the multiple-choice questions from these detailed diagrams.

Given all that was provided to students, how could they possibly not answer these questions correctly? These eight questions are a test of an assumption made explicit early in the paper – memorization requires abstraction. According to that assumption, if students do worse on these “seen” questions than they did on the unseen array questions, it is because they have not developed the capacity to memorize the algorithms at an abstract level, and reconstruct the code from the abstraction. (Indeed, since students had the diagrams, it would seem they do not even understand the diagrammatic abstraction.)

12.1.1 Question 19: Selection Sort

Below is “skeleton” code for the Selection Sort algorithm, where “xxxxxxx” replaces some code. The completed code is supposed to grow the sorted region from the “left” of the array “a” (i.e. from the end of the array with the lowest subscript):

```
for (int i=0; i<a.length-1; i++) {
    int chosen = xxxxxxx;
    swap(a, i, chosen);
}
```

Which is the correct replacement for “xxxxxxx”:

- (a) maxPos(a, 0, i);
- (b) maxPos(a, i, a.length-1);
- (c) minPos(a, i, a.length-1);
- (d) minPos(a, i, a.length-1);
- (e) minPos(a, 0, i);

Methods “maxPos”, “minPos” were presented during semester. These two methods search an array of integers (i.e. the first parameter) between the positions specified by the second and third parameters. The former returns the position of the maximum integer, the latter returns the position of the minimum. Method “swap” was also presented during semester. It swaps the array elements in the positions specified by the second and third parameters. The specifications of these three methods were given, as a reminder, in the exam paper.

The class did reasonably well on this question, with 72% correctly selecting option c. The upper two quartiles did well, indicating that the question is a reasonable test of mastery. Approximately 80% of bottom passing students answered correctly, but only 60% of fifty-fifty students answered correctly.

12.1.2 Question 20: Bubble Sort

Students were required to choose the correct implementation of Bubblesort from five possibilities. The distractors were similar to the correct answer, but contained bugs (e.g. a loop would terminate one iteration too early or too late). This proved to be one of the easiest

of the “seen” array questions. Even 55% of the fourth quartile students answered correctly.

12.1.3 Questions 21 to 23: Insertion Sort

Due to space limitations, these questions will not be discussed in detail in this paper. The complete Java code for this algorithm contains only nine lines. The two top quartiles performed very well on these three questions, while bottom passing students performed markedly better than fifty-fifty students.

12.1.4 Questions 24 to 26: AddElementToSet

A number of algorithms studied during semester related to storing the elements of a set in an array. The elements, all positive integers, are stored in ascending order in an array, with the end of the set indicated by the “sentinel” value of zero (declared as a constant). For example, an array declared as:

```
int s[] = {2, 4, 6, 0, 1, 7};
```

contains the set {2, 4, 6}, with the last two positions in the array not forming part of the set. The following code was studied during semester. It attempts to add an element “e” to the set “s”, returning false if there is not room for the new element:

```
// Find where "e" is, or should be
int pos = 0;
while((s[pos]<e)&&(s[pos]!=sentinel))
    pos++;

// if "e" is already in the set ...
if ( s[pos] == e ) return true;

// Find the position of the sentinel
int last = pos;
while ( s[last] != sentinel ) last++;

// Is there room for new element?
if ( last == s.length-1 )
    return false;

// Push up all elements > "e"
for (int i=last ; i >= pos ; i--)
    s[i+1] = s[i];

// Put new element into place
s[pos] = e;
return true;
```

The code shown in bold was omitted from the version given to students in the exam. The comments in the above code are abbreviated from the comments provided in the exam version. The final three questions in the exam required students to identify some of the above missing code. Only question 25 is discussed here.

12.1.5 Question 25

In the line after the comment “Is there room for new element?”, the correct completion of that line is:

- (a) if (s[last] == s[s.length-1])
- (b) if (last == s.length-1)
- (c) if (last == s.length)
- (d) if (s[last] == s[s.length])

Seventy percent of the class correctly chose option b. The upper two quartiles did well, indicating that the question is a reasonable test of mastery. Approximately 80% of bottom passing students answered correctly, but a very disappointing 35% of fifty-fifty students answered correctly. Distractor a was most popular, indicating some confusion between a position in an array and the contents of that position.

The two top quartiles performed very well on all three questions about the above code, indicating that they are reasonable tests of mastery. About 80% of bottom passing students correctly answered each of the three questions. The fifty-fifty students performed very poorly on the latter two of these three questions.

12.1.6 Discussion of Seen Array Questions

In considering the performance of students on these “seen” questions, we must remember that the students had been given a great deal of help – they had even been given these 8 questions in a pool of 30 prior to the exam! The fifty-fifty students performed far more poorly on these questions than the “unseen” array questions, indicating that they have not developed the capacity to memorize the algorithms at an abstract level, then reconstruct the code from the abstraction.

13 Conclusion

Academics in most universities believe that their first semester programming class is not working as well as it should. This belief leads to reluctance by academics to talk about the first semester programming class outside their own institution. Before public domain learning repositories become a reality, we will need to talk frankly, empirically, and quantitatively about what our novice programmers are actually capable of doing. The aim of this paper was to set the example. It is not important whether readers agree with the assumptions identified in this paper, but it is important that we learn to debate such assumptions.

If we can develop a cross-institutional culture of peer review, we will then be able to share educational resources, via repositories, thus removing a great burden from our individual shoulders.

Such a repository need not be made secure from the prying eyes of students. The evidence in this paper shows that a pool of well constructed multiple choice questions remain useful, even when students have already seen them, as memorization of large quantities of code requires abstraction. Indeed, the possibility arises of organising our teaching around the repository, making the questions available to students throughout semester.

References

Bell, J and Schauder, D (2003): The WEBWORKFORCE – a learning repository to support educators, trainers, and Information Technology courses, *Fifth Australasian Computing Education Conference (ACE2003)*, Adelaide, Australia. pp. 239-245

Ebel, R. and Frisbie, D. (1986): *Essentials of Educational Measurement*. Prentice Hall, Englewood Cliffs, NJ.

Fincher, S., (Nov. 1999): What are We Doing When We Teach Programming? *Frontiers in Education '99*, IEEE Press, 12a4-1 to 12a4-5.

Grissom, S, Knox, D, Copperman, E, Dann, W, Goldweber, M, Hartman, J, Kuittinen, M, Mutchler, D, Parlante, N (1998): Developing a digital library of computer science teaching resources. *SIGCSE Bulletin*, **30**(4), pp. 5-17.

Joyce, D, Knox, D, Gerhardt-Powals, J, Koffman, E, Kreuzer, W, Cary Laxer, C, Loose, K, Sutinen, E, and Whitehurst R A. (1997): Developing laboratories for the SIGCSE Computing Laboratory Repository: guidelines, recommendations, and sample labs; *The supplemental proceedings of the conference on Integrating technology into computer science education*, Uppsala, Sweden. pp 1-12.

Knox, D, Fincher, S, Dale, N, Adams, E, Goelman, D, Hightower, J, Loose, K, and Springsteel, F. (1999): The Peer Review Process of Teaching Materials *SIGCSE Bulletin*, **31**(4):87-100.

Lefrancois, G. (1999): *Psychology for Teaching*, Wadsworth / Thomson Learning. 10th edition.

Linn, R. and Gronlund, N. (1995): *Measurement and Assessment in Teaching*. Prentice Hall, Upper Saddle River, NJ.

Lister, R and Leaney, J (2003): First Year Programming: Let All the Flowers Bloom, *Fifth Australasian Computing Education Conference (ACE2003)*, Adelaide, Australia. pp. 221-229

Lister, R. (2004): Teaching Java First: Experiments with a Pigs-Early Pedagogy, *Sixth Australasian Computing Education Conference (ACE2004)*. *Conferences in Research and Practice in Information Technology*, pp. 177-183. Dunedin, New Zealand.

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

McCracken, M., V. Almstrum, D. Diaz, M. Guzdial, D. Hagen, Y. Kolikant, C. Laxer, L. Thomas, I. Utting, T. Wilusz, (2001): A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-year CS Students, *SIGCSE Bulletin*, **33**(4):125-140.