1. INTRODUCTION
Despite our best efforts, many students are still challenged by programming. A 2001 ITiCSE working group (the “McCracken group”) assessed the programming ability of a large population of students from eight universities, in the United States and four other countries [McCracken, 2001]. Each author tested his or her own students from a common set of programming problems. The students at each institution were required to write a program to solve one of the problems. Most students at all the institutions performed much more poorly than expected. Most students did not even get close to finishing the task. The results are compelling, given the multi-institutional, multi-national nature of the collaboration. Whereas a similar report from an author at a single institution might be dismissed as the result of poor teaching skills, that multi-national project suggests that there is something fundamentally wrong with the way we, as an entire discipline, teach programming.

While the work of the “McCracken group” has highlighted the extent of the problem, considerable further work is required to diagnose the cause of the problem. The nature of “McCracken group” study does not distinguish between two broad explanations of the cause of the problem: do students struggle with programming because they do not understand the programming language (the “language problem”), or are they comfortable with the language but lack training in the correct thinking processes required to construct a non-trivial program (the “design problem”)? It is difficult to distinguish between these two explanations in any scenario where students are required to write code. One way to distinguish between these two broad explanations is to ask students to demonstrate their comprehension of existing code. If a student cannot consistently demonstrate their understanding of existing code, then the student probably suffers from the language problem. If another student can demonstrate such an understanding, but struggles to write programs, then that student probably suffers from the design problem. The collaborators in this project believe that only when we have valid and reliable means of establishing that a particular student does not suffer from the language problem will we then be able to use such a student as a subject in any study of the design problem. This working group will therefore study the language problem.

2. BACKGROUND AND MOTIVATION
The four names shown as authors on this document are the people who submitted the original proposal to the ITiCSE conference organizers. All four authors are active in computer-science-education research. All four have experience teaching introductory programming in a variety of institutions, in three countries.

Two of the four authors, Lister and Thomas, have used multiple-choice questions (MCQs) as an assessment tool in their programming classes for several years (at different institutions, in different countries). The MCQs require the students to either (1) choose the correct result from the execution of a code fragment, or to (2) choose the correct missing line of code from a fragment.
Neither type of MCQ requires the students to synthesize code, certainly not in the strong five-step “problem solving” sense as defined by McCracken et al. (p. 126). Nevertheless, both Lister and Thomas have found that students perform badly on these tests. Lister has reported on his experiences with MCQs elsewhere [Lister, 2001; Lister and Leaney, 2003]. With a 70% passing threshold, he has found over several semesters that between one-third and two-thirds of his students fail such MCQ exams, despite preparation with similar practice questions. Such consistently poor performance suggests that many students suffer from the “language problem”.

Even more perplexing than the poor performance on these exams is the evidence for suspect student strategies for attacking these questions. Thomas provides “scratch” paper to her students to help them solve these MCQs. This scratch paper is collected at the end of the exam, but students make little effective use of it. On one occasion, only 36% of the sheets were returned with any kind of ‘working out’ on them. By contrast, when the same MCQs are presented to experienced colleagues, they draw profusely to determine the answers.

Thomas et al. (2004) report on a controlled experiment where they gave partially completed object diagrams to one group of students, and withheld the diagrams from a control group. The diagrams were intended to help students answer MCQs about the behavior of a related piece of Java code. The researchers found that there was no statistically significant difference in the performance of the two groups. In a follow-up experiment some weeks later, where students from both groups were asked similar questions but were not given the diagrams, the students who had prior experience with the diagrams showed little inclination to draw their own diagrams as an aid to answering the questions. In contrast, the teachers of these students do spontaneously draw such diagrams to answer those questions. Whatever technique these students were using to answer those questions, not only is it different from what their teachers use, but it is different from what the teachers believe a good programmer should be doing.

After discussing their independent teaching experiences, Lister and Thomas arrived at the following conjecture: weaker students cannot systematically create code, or even trace code, because they do not thoroughly understand the function of the code elements they are manipulating – the language problem. Some students who may at first glance appear to be suffering from the design problem are in fact manifesting the language problem: they bolt together some coding elements that they have frequently seen but do not completely understand, using heuristics that they hope will be right often enough to get them a passing grade. Thus, such students can create buggy code, but they lack the understanding to debug or even trace through that code. Because these students have constructed code, they may be misdiagnosed as suffering from the design problem.

The purpose of this working group is two-fold. First, the working group will seek to establish that the student behaviors observed by Lister and Thomas are also exhibited by students of other working group members. This first phase of the project is known as “Performance Benchmarking”. Second, the working group will further investigate the reasoning processes being used by the students. This will be achieved by having students answer these same MCQs while “thinking out loud”.

3. PERFORMANCE BENCHMARKING

In the first phase of the project, each working group member will test as many students at their own institution as is feasible, on a common set of MCQs. The purpose of this first part of the project is to gauge the general level of difficulty of each MCQ across all the participating institutions.

The working group members will test their students as either a normal part of the grading process, or as a test given to student volunteers. Administering the questions as a normal part of their grading process is the preferred method, as then there is little doubt about the commitment of students. However, each working group member will have to make the own determination on whether that is appropriate at their institution.

While the teaching of programming is similar in many institutions, the presentation order of topics varies considerably. Each working group member will need to make a judgment about the most appropriate students to study at their particular institution. Each working group member should choose students who have reached a stage in their degree program where normal grading practice assumes that the students are comfortable with the programming concepts used in the MCQs. At most institutions, the students of greatest interest to this study are most likely to be in their first or second semester of programming.

The students undergoing these MCQ tests should be given “scratch” paper. These pieces of paper should be collected from the students and will form part of the analysis of the working group.

6.1 The Core Set of MCQs

All working group members are expected to use the “core set” of MCQs. The exact set of MCQs to be used is not yet finalized, but it is expected that there will be about ten MCQs. This set of MCQs will focus on iterative processes on arrays.

The core set will use code fragments written in Java. However, participation in this working group is not restricted to those who teach Java. Working group members should have little difficulty in translating these questions into the language they are using at their own institution, even 3GL languages. (Indeed, the emphasis on iterative processes on arrays is precisely so that the working group is relevant to as many academics as possible.)

6.2 The Object-Java Set of MCQs
Some working group members have expressed an interest in studying student comprehension of Object-Oriented concepts. A set of MCQs is under preparation for this purpose. Working group members are free to participate in this optional part of the study.

These MCQs will use Java fragments. It may be difficult to adapt these MCQs to other Object-Oriented languages.

4. THINK OUT LOUD

In the second phase of the project, each working group member must interview at least three students. The purpose is to ascertain the thinking processes of students as they attempt these questions. In the interview, students will “think out loud” as they answer the core set of MCQs (and perhaps also the Object-Java Set of MCQs). The interviews are to be in English. The interviews are to be recorded and then transcribed verbatim, for subsequent analysis by the entire working group. Any scratch paper used by the students will also be collected for analysis.

With regard to the core set of MCQs, the students of primary interest to this study are likely to be those students who exhibit mediocre performance. Students who perform very well are also of interest, but students who perform very badly on the MCQs probably have mundane problems—they simply have not been working at their studies. When choosing interviewees for the core set, working group members should target students who are likely to fall in the middle of the performance range, say 40-80%.

If a working group member is using the MCQs as part of the grading process, the interviewees may have already done the MCQs as part of a class test. In such cases, working group members should recruit their interviewees prior to the class test, tell the recruits not to discuss the MCQs with other students prior to the interview, and conduct the interviews as soon as possible after the class test—ideally, immediately afterward.

Working group members not using the MCQs as part of grading may prefer to reverse phases one and two, and conduct the “think out loud” interviews prior to bringing a group of volunteer students together to collect the performance benchmarking data. In such cases, the interviews should be asked not to discuss the MCQs with the volunteers prior to the performance benchmarking.

5. WORKING GROUP MECHANICS

Until April 19, the meeting of this (or the other) working group at ITiCSE 2004 is not certain. Until that time, the ITiCSE organizers reserve the right to close down any working group, if they feel that insufficient people have joined that group. Therefore, all working group members should actively seek out new members. (We are optimistic, however, that the working group will proceed.)

By May 31, participants should have collected their data, and be in a position to share it (via email attachments) with other working group members. (Including the transcripts.) This will allow people to be familiar with the data before the working group convenes at ITiCSE.

The following "rules" may sound a little ominous, but I am not trying to be a dictator. We have found that successful working groups follow these suggestions and are prepared prior to and during the conference. Your successful proposal reflected such care and planning, and we know you will be successful in your group activities. The arrangements for groups will be similar to the last few years of ITiCSE. We will operate as follows:

0. Groups will convene as early as possible on the SATURDAY before the conference (June 26th).
1. Groups will work uninterrupted throughout the Saturday and Sunday. We will arrange some type of lunch get-together on Sunday to allow the exchange of ideas and experience.
2. Working groups will be expected to brief the conference on their progress and seek feedback from the conference attendees on their activities. This will occur on Monday (June 28)
3. It is expected that most time will be spent during the week (June 28th-30th) pursuing Group activity, although members will obviously seek to attend some conference presentations.
4. Groups will conclude their collective activity at the conference close (June 30th), by which time a draft report should exist for delivery to me.
5. Convenors will be responsible for forwarding a final report in publishable form to me by the end of July 30th. This should permit further electronic work and polishing as required, and obviate the need for rushed editorial work.
6. Final reports will be sent out for formal refereeing and, if deemed suitable, forwarded to SIGCSE for publication.
7. Participants will be advised to contact group convenors (ie. Raymond Lister)
6. ETHICAL CLEARANCE
It is a requirement of participation in the working group that participants complete the ethical clearance (“IRB”) processes at their institution. For most, this should not be onerous.

7. RELATED LITERATURE
A number of theories of program comprehension strategies have been developed in past work, and are summarized in a classic survey document by Deimel and Naveda (1990). However, most of this work studied experienced professional programmers rather than students. That literature on program comprehension begins by assuming that the expert is familiar with the programming language, which is not an assumption we make in this study. That literature also concentrates on the strategies used by experienced programmers to cope with the complexity of large programs, whereas large programs are not needed to study the "language problem" in students.

Working group members who are aware of papers relevant to this study should communicate the details of those papers to other working group members, as soon as possible.

8. FREQUENTLY ASKED QUESTIONS
Questions and answers will be added to this section as more questions are received from prospective working members.

FAQ-01: Why Multiple Choice Questions?
We use MCQs for two reasons. First, since this is a multi-institutional study, we wanted a way of scoring student performance that did not require judgment on the part of each working group member. Second, we were concerned that if students were instead required to explain the function of a piece of code, poor performance might be due to a lack of eloquence, not a lack of understanding of the programming language (especially where the student does not speak English as their first language).

FAQ-02: How will the common set of MCQs be generated?
Its likely that most of the MCQs will be questions already used by Lister in grading his classes. Prospective members of the working group should email him for a copy of one of those exam papers.

FAQ-03: Can I use my own MCQs?
Members of the working group might wish to generate another set of MCQs, in addition to the existing two sets of MCQs, to study some other issue. This may be possible, but will need to be discussed and agreed among the working group members. There is danger that the study could become too large and fragmented. Other sets of MCQs might be better left for a subsequent research project, when the design of that study will benefit from the hindsight of this project. All members of the working group must administer the MCQs in the core set.

FAQ-04: How many hours should data collection take?
With regard to performance benchmarking, if you are using the MCQs as part of your grading, then performance benchmarking should not add to your marking burden. If you are not using the MCQs as part of your grading, then you can collect your student volunteers into one room and administer the test simultaneously (indeed, this is recommended to eliminate students telling each other about the questions). Most students should complete the core set of MCQs in 0.5 to 1 hours.

Interviewing students should take no more than one hour per student. (We encourage working group members to do more interviews than the minimum of three.) The most time consuming aspect of data collection may be transcribing the interviews, unless you are a quick typist, or you can arrange to have a skilled typist transcribe the tape for you (recommended!).

9. REFERENCES
[3] Lister, R (2001), Objectives and Objective Assessment in CS1, 32nd Technical Symposium on Computer
