

The BRACElet 2009.1 (Wellington) Specification

Jacqueline L. Whalley

Computing and Mathematical Sciences
Auckland University of Technology
Auckland 1020, New Zealand
jacqueline.whalley@aut.ac.nz

Raymond Lister

Faculty of Engineering and Information Technology
University of Technology, Sydney
NSW 2007, Australia
raymond@it.uts.edu.au

Abstract

BRACElet is a multi-institutional computer education research study of novice programmers. The project is open to new members. The purpose of this paper is to: (1) provide potential new members with an overview of BRACElet, and (2) specify the common core for the next data collection cycle. In this paper, BRACElet is taking the unusual step of making its study design public before data is collected. We invite anyone to run their own study using our study design, and publish their findings, irrespective of whether they formally join BRACElet. We look forward to reading their paper.

Keywords: BRACElet, novice programming, multi-institutional collaboration.

1 Introduction

In any academic discipline, it takes years to become an expert. Research across several disciplines indicates that experts, in addition to knowing more, also organize their knowledge into more sophisticated and flexible forms than novices. This enables the expert to bring the most appropriate form of knowledge to bear on solving a specific problem. For example, when asked to memorize chess board positions, novices tend to remember the position of each piece in isolation, whereas experts recognise and remember the attacking and defensive combinations of the pieces (Chase and Simon 1973).

Results from many studies of novice and expert programmers (e.g. Adelson 1984) are consistent with the findings from other disciplines – novice programmers form concrete representations based on how the code functions whereas experts form abstract representations based upon the purpose of the code.

The Leeds Group (Lister et al. 2004) gave students short pieces of code and asked them to determine what the values in the variables would be after the code had been executed. The Leeds Group found that almost all the students – even the ‘top’ students who answered most questions correctly – solved the problems by hand executing (‘tracing’) the code. In contrast, a follow up study found that academics tended to use a more sophisticated strategy to solve similar problems (Lister et

al., 2006). Instead of tracing the code, the academics deduced the computation being performed by the code and then inferred the output directly from the input.

When teaching programming, lecturers frequently use diagrams to illustrate how a piece of code works. Expert programmers also frequently use diagrams to develop their understanding of an unfamiliar or buggy piece of code. In contrast, Thomas, Ratcliffe, and Thomasson (2004) found that many of their students were reluctant to use diagrams as an aid in tracing code, under circumstances that encouraged the students to use diagrams, even after students had been explicitly instructed in how to use the diagrams. Thomas, Ratcliffe, and Thomasson were led to conjecture that providing students with a specific diagrammatic abstraction of the code was not helpful because the self-development of such abstractions is intrinsic to developing an understanding of code.

If students cannot trace through code, how can those students write code? Traynor, Bergin and Gibson (2006) interviewed students who had completed exercises in both code tracing and code writing. One of the students who did relatively better on the writing tasks than the tracing tasks explained that the way in which writing tasks are graded helps students to gain a passing mark with only a weak grasp of what the code needs to do:

“... you usually get the marks for making the answer look correct. Like if it's a searching problem, you put down a loop, and you have an array and an if statement. That usually gets you the marks ... Not all of them, but definitely a pass”.

Student quoted in Traynor, Bergin and Gibson (2006)

The implication of the above student quote is that, when an academic grades code written by a student, there is a danger that the teacher will subscribe to the student a depth of understanding that the student does not have.

In our own teaching, we have noticed behaviours indicating that students do not understand their own code, such as:

- Attempting to debug code, sometimes for long periods of time, by ‘random mutation’.
- Introducing new bugs as they attempt a superficial and incorrect fix to an existing bug. Ginat (2008) noted a similar behaviour.
- Asking the teacher for help with a bug, after the student has worked hard to find the bug, and the teacher identifies the bug with a single reading of the code.

- Being unable to explain their own code to the teacher. Thomas, Ratcliffe, and Thomasson also relate such an anecdote.

Our own teaching, and the literature on novice programmers, has led the authors (and our collaborators in the BRACElet project) to the belief that there is hierarchy of skills associated with programming. At the bottom of the hierarchy is knowledge of basic programming constructs (e.g. what an “if” statement does). At the top of the hierarchy is the ability to write non-trivial, correct code using those programming constructs. The intermediate levels of the hierarchy are manifested in abilities such as:

- The ability to trace code, especially when there are too many variables to maintain in short term memory.
- The ability to understand/use diagrams and other abstractions of code.

More formally, our research goals are encapsulated in the following questions:

- Are there intermediate skills and knowledge in programming? If so, then ...
- Is it possible to assess explicitly a student’s grasp of these intermediate skills and knowledge?
- Is it possible to teach explicitly these intermediate skills and knowledge?

These are the research questions that have driven the BRACElet project. In the next section, we review past work on BRACElet. Subsequent sections of the paper then define the next iteration of work in the project. On the basis of this description of BRACElet, we invite others to join the project, or to at least conduct their own related studies.

2 The Brief History of BRACElet

The BRACElet project is a multi-institutional study of novice programmers. The project commenced in 2004 and to date 10 workshops have been held in Australasia. A useful summary of the first eight workshops has been provided by Clear et al. (2008c) and the reader is referred to that paper for a more detailed history of the project.

While BRACElet is a research project, the intention is that the project should remain close to education practice. Thus, most BRACElet data is collected via end-of-semester exams that students take at the participating educational institutions.

The following subsections summarise the evolution of the BRACElet project, through the workshops and in the papers that have emerged from the project. These subsections only offer a brief summary, and readers are referred to the earlier BRACElet papers for a comprehensive account.

2.1 The First BRACElet Paper

The first BRACElet workshop, in December 2004, began with a review of the results from the then recently published Leeds Group (Lister et al. 2004). The workshop participants felt that the Leeds Group study was not sufficiently based upon learning theories or educational models.

The workshop participants selected the revised version of Bloom’s taxonomy (Anderson et al., 2001) as an educational model for test question development, and then devised several questions that mapped to different parts of that taxonomy. None of the questions required the students to write code, but instead tested their ability to reason about code and also reason about associated abstractions, such as flow charts. These questions were then included in the end-of-first-semester exams that students attempted at some of the participating New Zealand institutions, in June 2005. Analysis of the data from those exams began at the second BRACElet workshop, in July 2005. The first BRACElet paper (Whalley et al. 2006) was a consequence of the analysis started at that second workshop.

One of the findings reported in that paper concerned the revised Bloom’s taxonomy. The results are summarised in Figure 1. Students tended to do best on questions from the *Understand* level of the taxonomy, which was the lowest level of taxonomy for which questions were designed, and which required the least abstract reasoning. Students tended to do less well on the more abstract *Apply* questions, and students did least well on *Analyse* questions, which was the highest level of the taxonomy for which BRACElet participants designed questions, and which required the most abstract reasoning (but did not require students to write code).

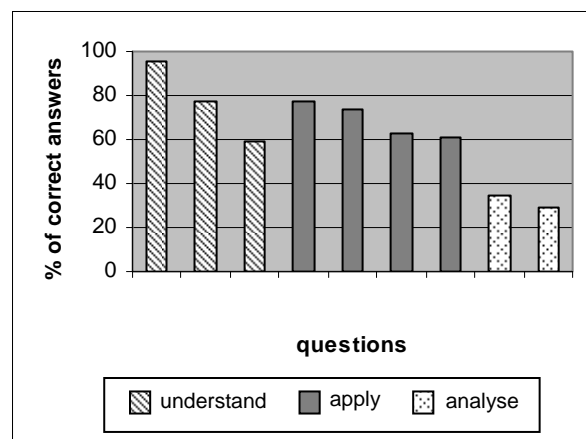


Figure 1: Novice performance on questions classified by Bloom’s revised category (Whalley et al., 2006).

The second workshop added another educational model to the project, the SOLO taxonomy (Biggs and Collis 1982). This taxonomy was introduced to analyse data for the question shown in Figure 2. The student answers to that question were placed into one of five categories, based on the SOLO taxonomy, as shown in Table 1.

In plain English, explain what the following segment of Java code does:

```

bool bValid = true;

for (int i = 0; i < iMAX-1; i++){
    if (iNumbers[i] > iNumbers[i+1])
        bValid = false;
}

```

Figure 2. An ‘explain in plain English’ question

Using the scores that the students achieved on all the BRACElet questions in the exam, except the ‘explain in plain English’ question, the students were broken into four quartiles. Figure 3 shows the types of SOLO answers that students in each of those quartiles gave to the ‘explain in plain English’ question. Students in the top two quartiles were far more likely to give a relational answer (i.e. an answer giving a summary of the purpose of the code) than students in the lower two quartiles.

SOLO category	Description of student’s answer
Relational	A summary of the purpose of the code. For example, “checks if the array is sorted”.
Multistructural	A line by line description of all the code. Some summarisation of individual statements may be included.
Unistructural	A description of one portion of the code (e.g. describes the <i>if</i> statement).
Prestructural	Shows substantial lack of knowledge of programming constructs or is unrelated to the question.
Blank	Question not answered.

Table 1: The SOLO Categories for the students’ answers to the ‘explain in plain English’ question

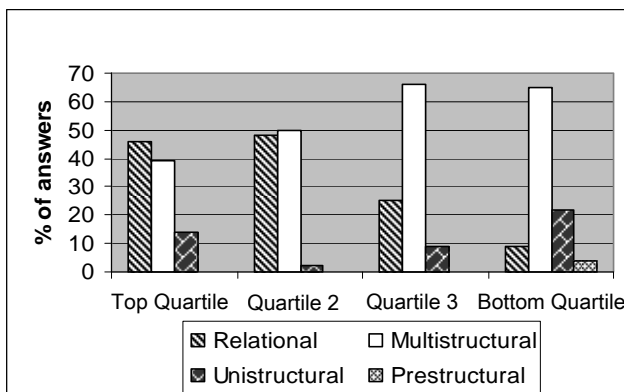


Figure 3. Performance of students on the ‘explain in plain English’ question (Whalley et al., 2006).

In the conclusion of that first BRACElet paper, the following observations were made:

It appears likely that programming educators may be systemically underestimating the cognitive difficulty in ... [exam questions] ... for assessing programming skills of novice programmers. ... The stages through which novice programmers develop ... and the time that this development process may take... may have been significantly underestimated ... Students who cannot read a short piece of code and ... [summarise the computation performed by that code]... are not well equipped intellectually to write code of their own.

2.1.1 Reflection: The Two-Task Approach

BRACElet has always been focussed on identifying aspects of novice programmers that transcend institutional boundaries. The most obvious potential difference between cohorts of novice programmers at different institutions is the ability of the ‘typical’ student at each institution. Suppose we are studying students at two different Institutions, I1 and I2, where I1 is a more prestigious educational institution than I2. If we give the same programming question, Q1, to students at both institutions, it is likely that, on average, students from I1 will do better on Q1 than students from I2. The Leeds Group faced the same problem and made the following observation:

There are trends in the data across institutions. In general, a question that is substantially more difficult for students at one institution is also more difficult for students at other institutions.

Lister et al. (2004, p. 126).

The BRACElet project has taken that observation and used it as a method of analysis, referred to as the two-task approach. Suppose students at I1 and I2 are given two questions Q1 and Q2. While students at I1 might do better on both Q1 and Q2 than students at I2, BRACElet participants look for patterns in the relative performance of students on the two questions. For example one pattern may be that students at both I1 and I2 do better, on average, at Q1 than Q2.

The two-task approach generalizes to N tasks. The first BRACElet paper (Whalley et al. 2006) used a three task approach when analyzing questions from different levels of Bloom’s taxonomy. Even though students at I1 might outperform students from I2 at each level of the revised Bloom’s taxonomy, students within each of those institutions will tend, on average, to perform better on *Understand* questions than *Apply* questions, and better on *Apply* questions than *Analyse* questions.

The two-task approach can also be used another way. One task (or one set of tasks) is used to place students from different institutions into two or more categories. The students in a single category are then considered to be equivalent, even though they come from different institutions. All students are also given a second task (or a second set of tasks). The statistical performance on the second task by students in one category is compared to the statistical performance of students in another category. This two task approach was used to perform the SOLO analysis of the above ‘explain in plain English’ question. Nine multiple choice questions were used as the task set 1. Students were placed into four quartiles (i.e. categories), based upon their score on those nine multiple choice questions. The analysis then focussed on differences between the quartiles in the frequencies of the various SOLO levels.

The two-task approach reduces the need to collect data describing the educational and institutional backgrounds of students. For example, if two students are both placed into the same category based on their performance on task 1, BRACElet participants are not concerned if one of those students had programmed prior to arriving at university, and the other had not; nor are BRACElet

participants concerned if one of those students is from an elite university and the other is not.

2.2 Follow up to the First Paper

A number of subsequent BRACElet papers elaborated upon the work of the first BRACElet paper. One of those papers (Whalley 2006) looked for any effect due to programming language. BRACElet participants did not teach the same programming language, so the questions were translated into the appropriate language for each institution. Whalley's analysis revealed no significant difference between the mean total score of students at one institution who completed the questions in the Delphi programming language and the students at another institution who completed the test in C++.

Another follow up study (Lister et al. 2006) asked academics to 'think out loud' as they solved problems from the Leeds Group study that were similar to problems from the first BRACElet paper. That follow up study found that academics actively seek to abstract beyond the concrete code, whereas the Leeds Group had found that novices did not seek to abstract.

2.3 Code Classification Questions

Thompson et al. (2006) explored a different kind of assessment question that asked students to make explicit statements about their understanding of code. In an exam question, they presented students with four segments of code. All four pieces of code contained a loop with an "if" statement inside the loop. Two of the segments used a 'for' loop and other two segments used a 'while' loop. The students were not told that two of the pieces of code found the minimum value in an array, and the other two pieces found the maximum. Students were asked to iteratively place the four pieces of code into two groups, so that code pieces inside a group were similar in some way and different from the code pieces in the other group. Thompson et al. found that many students grouped the code pieces based on syntactic features (e.g.; "uses a for loop vs. uses a while loop", but failed to identify the more abstract functional similarities that indicate an ability to abstract the purpose of code (e.g. "finds the minimum vs. finds the maximum").

Thompson et al. also compared student responses to the 'explain in plain English' questions with their performance on the new classification question. Thompson et al. found that if a student gave a relational response to an explain in plain English question they were more likely to provide responses that identified functional similarities when answering a classification question. 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 highlighted a number of interesting questions for future study, including:

- Can a novice programmer's ability to understand and write programs be fostered through the use of variations in code and through code classification questions?
- 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 achieves the same task?

2.4 Parson's Problems

A common criticism of having students solve pen-and-paper code writing problems in an exam is that it is a non authentic task and it is unreasonable to expect students to program without the support of a compiler.

Parson's puzzles are an alternative to code writing. In such puzzles, students are presented with jumbled lines of code which they are required to place into the correct order so that the code performs a prescribed task (Parsons and Haden 2006). The same type of puzzle was also developed independently for the "Head First" textbooks (Sierra and Bates 2005).

In more recent iterations of the BRACElet study (Whalley and Robbins 2007b, Lopez et al. 2008) a simple Parson's puzzle was used in an exam. The students performed much better on Parson's puzzles than code writing questions despite the fact that the students had never seen such a problem before. The authors suggested that Parson's puzzles might be a bridge between full code writing tasks and code reading tasks.

Denny, Luxton-Reilly and Simon (2008) furthered the study of student performance on Parson's puzzles and enlisted a student feedback loop in order to refine the way in which Parsons puzzles are presented in an examination. A notable correlation between Parson's scores and code writing scores was found. They recommend the use of Parson's puzzles in assessment:

"Parsons problems are easier and more reliable to mark than code writing, provide an opportunity to test student misconceptions more specifically than code writing, yet they appear to require the same set of skills (as analyzed by correlation in marks achieved). This makes them an excellent alternative to traditional code writing questions"

2.5 The Traffic Light Conjecture

Philpott, Robbins and Whalley (2007) analysed student responses to code tracing and 'explain in plain English' questions. A link was found between well developed tracing skills and the ability to think relationally about code. Conversely, students who were unable to arrive at a correct answer when tracing code were unable to reason about code segments, and instead focused on a single line of code within code segments. These findings led to the 'traffic light' conjecture – that a student's degree of mastery of code tracing tasks indicates their readiness or ability to be able to reason about code. It was proposed that a student has reached a 'green light' for relational thinking if they have mastered the ability to trace code. If they are able to trace reliably most of the time they have reached an 'orange light'. If students trace code correctly less than 50% of the time they are still at a 'red light' and they are only able to explain code in a unistructural way.

This study highlights the use of the two-task approach to study a hierarchy of programming skills. Here, tracing is task 1, and only students who demonstrate a particular level of competence in task 1 are capable of performing task 2.

2.6 The Relationship to Code Writing

From the outset of the BRACElet project, we intended to investigate the relationship between code reading and code writing. As educators, it was our intuition that code tracing is easier than code writing and that the ability to read code is a precursor to code writing, but there is little direct evidence to support that intuition.

Sheard et al. (2008) gave students three ‘explain in plain English’ questions in an end-of-first-semester exam, and also required the students to write code. The level of SOLO response to the ‘explain in plain English’ questions correlated positively with performance on writing tasks. However, correlation is not evidence of causality, and there remains much work to do before it can be said with confidence that the capacity to answer ‘explain in plain English’ questions involves skills that are a precursor to code writing.

2.7 The Path to Abstraction

Lopez et al. (2008) analysed student responses to an end-of-first-semester exam, looking for evidence of a hierarchy of programming skills. They looked for statistical relationships between the questions in the exam; specifically how well non-writing questions predicted student performance on a code writing question. A stepwise regression, with performance on code writing as the dependent variable, was used to construct a path diagram. The diagram suggests the possibility of a hierarchy of programming related tasks. At the bottom of their regression hierarchy were exam questions that required little more than knowledge of programming constructs. At intermediate levels of the regression hierarchy were “explain in English” questions, Parson’s puzzles, and the tracing of iterative code.

2.8 Doodles

McCartney et al. (2007) analysed student annotations on a test sheet (‘doodles’) and investigated the relationship between those doodles, the difficulty of the questions and student performance. They found that students who doodled performed better on the Leeds problem set.

Whalley, Prasad and Kumar (2007a) also looked at student annotations using a new problem set. They used the SOLO taxonomy to investigate the level of reasoning that students achieved when answering a short answer question and related this to the type and number of annotations they made. Whalley Prasad and Kumar found that despite encouraging students to doodle and teaching tracing, almost two thirds of the students turned in an exam paper on which there were no annotations. While higher achievers were more likely to doodle there seemed to be no relation between the use of annotations and the SOLO responses to ‘explain in plain English’ questions.

Whalley, Prasad and Kumar also found that certain types of questions are more likely to elicit certain doodle types. They argued that certain doodles actually arise because of the constructs in the code rather than the question type. For example, it could be argued that tracing would be a more appropriate strategy when the code fragment contains a loop.

This study raised several questions which have still not been investigated:

- Why in the majority of circumstances do students not annotate their scripts?
- When students do doodle what motivates them to do so?
- When students do annotate are they using doodles that are appropriate for the question type?

2.9 Reasoning with Diagrams

Lister (2007) used a two-test approach to study student performance on an end-of-first-semester exam. Task 1 comprised four multiple choice questions, in which students were asked to determine the value in a particular variable, after a given piece of code had finished executing.

In the second task, students were given eight multiple choice questions, but questions of a different nature from those in task 1. For each task 2 multiple choice question, students were given a set of diagrams that described an algorithm, and they were also given code that implemented that algorithm, with one or two lines missing. In each of the eight questions, students had to choose the appropriate line(s) from four options provided. Of the students who scored a perfect 4 on Task 1, approximately 20% scored 5 or less out of a possible 8 on Task 2. Those 20% were accomplished concrete reasoners about code – having scored a perfect 4 on task 1 – but those students did not manifest the more abstract skill of relating code and diagrams. This study is evidence that the ability to reason about algorithms diagrammatically is a skill that is higher on the hierarchy than the ability to trace code.

This study is consistent with the work of Thomas, Ratcliffe, and Thomasson (2004) that was discussed in the introduction of this paper. They found that many of their students were reluctant to use diagrams as an aid in tracing code – which is not surprising for any student who cannot reason about algorithms diagrammatically.

2.10 Developing the Research Methods

The BRACElet project has made extensive use of two taxonomies, the revised Bloom’s taxonomy and the SOLO taxonomy. However, in doing so, BRACElet participants have had to come to a mutual understanding of how these taxonomies apply to the work of novice programmers. This has not been easy – indeed BRACElet participants are yet to develop a strong consensus on how these taxonomies are best applied to novice programmers. Thompson, et al. (2008) represents our best attempt thus far to apply a consistent interpretation with concrete exemplars of the revised Bloom’s taxonomy, and Clear, et al. (2008b) does the same for the SOLO taxonomy.

3 The BRACElet Guiding Principles

A multi-institutional collaboration like BRACElet can only be successful if the participants have bought into the principles on which the project is founded. In this section we supply a brief summary of the guiding principles for the BRACElet paper. Many of our guiding principles

have their foundation in earlier multi-institutional projects (Fincher et al. 2005).

A detailed account of the practices and rules for BRACElet collaborators are in Whalley and Robbins (2007b), Whalley, Clear and Lister (2007c) and Clear et al. (2008a). Anyone who is contemplating joining BRACElet should read those documents.

3.1 Belief in a strong teaching–research nexus

BRACElet members believe in a strong teaching–research nexus. We see our work as an evidence-based and research-informed way of improving our teaching. We see BRACElet as:

“... the development of theory or understanding as a by-product of the improvement of real situations, rather than application as a by-product of advances in ‘pure’ theory.”

(Carr and Kemmis, 1986, p. 28).

As part of this commitment to research, BRACElet members debate the evidence, and not their folk-pedagogic intuitions.

3.2 Assessing the student always comes first

One of the core guiding principles is that assessing the students always comes first and the research must be conducted without compromising the course.

3.3 The Common Core

Placing the student first means that it is impossible to run exactly the same experiment (i.e. use exactly the same exam questions) in every participating institution. Instead, before each data gathering phase, BRACElet members agree upon a ‘common core’, which is a minimal set of properties that must be present in all their exams. For example, past common core definitions have specified conditions like ‘all exams will contain at least one explain in plain English question’. Such a specification leaves each institution with many degrees of freedom, such as what programming constructs will be in their choice of code, how long the code will be, and whether or not to give the students practice in this type of question prior to the exam. BRACElet participants were even free to use a different phrase than ‘explain in plain English’. For example, in one exam (Sheard et al. 2008) the students were asked to supply a meaningful name for a method.

The specification of the common core for the next phase of BRACElet data gathering – the 2009.1 (Wellington) specification – is given later in this paper.

3.4 Repetition for Robustness

Rather than weakening BRACElet, the common core makes our findings more robust. When non-identical tests are used at different institutions, but we see similar patterns in the results, we have a research outcome that has implications beyond the classrooms of the BRACElet collaborators. Fincher and Petre (2004) explained the importance of this type of repetition:

“... repetition can show how consistent the outcomes of a given study are across different related tasks,

across different environments, across different related contexts.”

3.5 The Rules of Engagement

What really drives BRACElet is a set of “Rules of Engagement” that cover the brainstorming or ideas phase and the post-brainstorming phase (Whalley, Clear and Lister, R. 2007c). The basic principle of these rules is that ideas are the easy part. BRACElet recognises that the hard graft is done gaining ethics clearance, drafting the actual final instruments, collecting the data, doing the analysis and writing the paper. When we meet we share ideas freely. All meeting participants are free to use those ideas, without any obligation, such as co-authorship, to the person who first uttered the idea.

3.6 The Publication Protocol

BRACElet members are not automatically authors of any BRACElet paper. In fact, most BRACElet papers are written by subgroups. Our policy has been that all co-authors must have made a substantial contribution to at least two of the following four activities:

- Conception or design
- Data collection and processing
- Analysis and interpretation of data
- Writing substantial sections of the paper (e.g. synthesising findings in a literature review or a findings/results section)

Also, everyone who is listed as an author should be able to defend the paper as a whole (although not necessarily all the technical details).

With the publication of this paper, there is a relaxation of one the above authoring requirements, which will apply as an experiment in the next data gathering phase, after which it will be considered as a permanent change to the rules. The relaxation is that anyone who contributes data that is collected in an exam from their students, but does not contribute on any of the other points, will be a co-author on at least one BRACElet paper that uses that data, probably the first paper. Note that contributing data implies meeting the institutional ethical clearance processes, adapting the questions to the specific exam, and providing the data in a form that is useful to other BRACElet participants.

Anyone who attended both days of the BRACElet workshop in Sydney, in September 2008, is deemed to have contributed to conception and design for the next iteration of the BRACElet research cycle.

3.7 Membership / Recruitment

BRACElet welcomes new members. Indeed, the need to repeatedly explain and justify our research to new members actually tests and strengthens our research. At every BRACElet meeting, there is little distinction between reviewing the project and bringing the new people up to speed – if we cannot explain our research to new people, what use is that research to anyone?

Joining BRACElet obliges the new member to abide by our guiding principles, especially the rules of engagement and the publication protocol. Also, members need to

attend some of the project meetings, but by no means all meetings. Almost all members begin their membership via attending their first meeting.

New BRACElet members *must* complete the appropriate human ethics clearance processes at their institution (known as ‘IRB’ to most Americans). For anyone who would like to join BRACElet, but who has no prior experience with getting such approval, we can help.

3.8 The Open Research Plan

Prior to the publication of this paper, BRACElet has traditionally kept each research plan private among its members until the data has been collected, analysed and a paper has been written. With this paper, BRACElet begins an experiment with a different approach, where a specification of the research plan is published prior to data collection, analysis and publication.

With the publication of this paper, our research plan is in the public domain. Anyone is free to collect their own data, analyse and publish it. In fact, we encourage people to do so. Neither the authors of this paper, nor existing members of BRACElet, have any claim to co-authorship with anyone who does so, nor do we have any claims to the data of those people. However, we do ask that people observe certain professional courtesies. They should:

- (1) Cite and summarise this paper in their paper(s).
- (2) Not claim to be part of the BRACElet project, unless they have our explicit approval to do so. Instead, they should refer to their work as being based upon the BRACElet 2009.1 (Wellington) specification. We reserve all naming rights to “BRACElet”.
- (3) Be explicit in their paper(s) about any alteration they made to the 2009.1 (Wellington) specification. They should NOT use a new version number to refer to their changes. We reserve naming rights to version numbering.

4 The 2009.1 (Wellington) Common Core

This section specifies the BRACElet 2009.1 (Wellington) common core, which comprises three parts:

- (1) Basic Knowledge & Skills
- (2) Reading / Understanding
- (3) Writing

These three parts are described in detail the next three subsections.

If students complete the different parts of the common core at different times, the elapsed time between doing the first part and doing the last part should be no more than 1 week. Also, record the relative times when each part was done (e.g. ‘part 2 was done 3 days after part 1’).

4.1 Basic Knowledge & Skills

The purpose of this part of the common core is two-fold. First, it establishes that students understand the programming constructs (e.g. how an “if” statement or a “while” loop works). Second, it establishes that students have mastered relatively concrete skills, such as tracking variable updates as the student traces through code.

Ideally, BRACElet participants should use several questions to test students on their basic knowledge and skills.

Thus far in BRACElet, some form of tracing question has always been used to establish that students understand the programming constructs. As a guiding (but non-mandatory) principal for tracing questions, there should be three types of questions:

- (1) Non-iterative (and non-recursive) tracing questions, where several programming constructs may be tested in the one question;
- (2a) Iterative tracing questions, without control logic within the loop, and with a very small number of variables, perhaps no more than a loop control variable plus one other variable. For example, a loop that sums the elements of an array meets those criteria (but it might be wise to avoid using a variable called “sum”, otherwise students might guess the answer). This type of question establishes whether students understand the iterative construct used.
- (2b) Iterative tracing questions that have control logic within the loop (e.g. the code in Figure 2) which establishes whether students can track variable updates as they trace that code. The code in at least one of these tracing questions should be of similar complexity to the code in one question from part 2 of the common core. Also, the code in at least one of these tracing questions should be of similar complexity to the code in one question from part 3 of the common core

Most BRACElet participants use multiple choice questions for tracing questions, but free response is also allowed.

These questions should not contain buggy code. That is, the code should not contain simple bugs, such as a loop that terminates either one iteration too early or too late. As a guiding principal, if an expert programmer was to attempt these questions hastily, they should not be tricked by a simple bug. Note, however, that nonsense code is permissible (i.e. code that has no purpose that an expert programmer could easily identify and describe).

BRACElet members are left free to decide whether the code will use meaningful variable names. Members who use meaningful names for parts 2a and 2b are warned not to use a piece of code for which a student might then guess the function of the code, and thus answer the question without tracing the code.

New BRACElet participants are encouraged to reuse questions published in past BRACElet papers – it is unlikely that any of your students are reading the BRACElet papers.

The common core does not specify on which programming constructs the knowledge of students should be tested – that is determined by the syllabus at each participating institution. The common core merely specifies that this part of the common core should test all the programming constructs used in the other two parts of the core.

4.1.1 Anticipated Analysis

The primary purpose of this first part of the common core is to act as task 1 in two-task analysis. However, we would also like to build upon the earlier BRACElet work on the doodles students make when tracing code (Whalley, Prasad and Kumar 2007a). Therefore, BRACElet participants should retain the exam scripts / answer booklets of their students, and be prepared to show those materials to other BRACElet members. This may be an issue for human ethics clearance. Students should be explicitly encouraged on the exam paper to show their working in a designated place (e.g. on the same page where a question occurs in the exam paper; perhaps in a box on that page if there is more than one question on the page).

When collecting data an established anonymous script scanning and data source identification protocol should be adhered to. All participants must follow this protocol, usually enforced by an institutions human ethics approval, which is created to safeguard the privacy of students and institutions.

4.2 Reading / Understanding

The purpose of this part of the common core is to establish whether students deduce the purpose of a piece of code from reading the code. BRACElet members should use at least one question based on any of the following three question types:

- (a) Explain in plain English. BRACElet members are free to use an instruction to students other than ‘explain in plain English’.
- (b) Parson’s problems.
- (c) Code Classification Questions, as in Thompson et al. (2006). However, if a BRACElet participant elects to use this type of question, they should use at least one of the other question types (as this type of question is the least evaluated and therefore the most risky of the three types).

Any code used for an explain in plain English question should not have been used in a part 1 tracing question. Otherwise, there is a danger that students might guess a SOLO relational response from the output of the code, when our aim is to see if they can produce a SOLO relational response by reading the code. It follows that at least some of the variables in any code should be left non-initialized; otherwise the student might trace the code. One way of doing this is to present the code as a method, with some of the variables declared as part of the method header.

The code used in at least one of these reading/understanding questions should be of similar complexity to at least one part 1 tracing task. Also, the code used in at least one of these reading/understanding questions should be of similar complexity to at least one part 3 writing task.

For repetition and robustness, new BRACElet participants are again encouraged to reuse questions published in past BRACElet papers.

4.2.1 Anticipated Analysis

BRACElet participants are encouraged to include several reading/understanding questions in their exam (of the same type, or of different types). With multiple questions we can examine the following issues:

- (a) For ‘explain in plain English’ questions, do students tend to provide responses at the same SOLO level for code of similar complexity? The work by Sheard et al. (2008) suggested that this was the case.
- (b) For two ‘explain in plain English’ questions of differing levels of complexity, do students tend to provide a response at a lower SOLO level for the more complex code? Again, the work by Sheard et al. (2008) suggested that this was the case.
- (c) For a given level of code complexity, is there a statistical relationship between the SOLO level of a student response to an ‘explain in plain English’ question, and a code classification question? The work by Thompson et al. (2006) suggested that this was the case.
- (d) For a given level of code complexity, can students who provide multistructural responses to ‘explain in plain English’ questions solve Parson’s Problems?

4.3 Common Core 3: Writing

Most programming exams already require students to write code, and BRACElet has only one additional requirement for such writing questions – at least one code writing task should be of similar complexity to one tracing task (i.e. from part 1 of the core), and at least one code writing task should be of similar complexity to one reading/understanding task (i.e. part 2 of the core).

4.3.1 Anticipated Analysis

BRACElet regards code writing ability as the dependent variable. That is, if programming is indeed a hierarchy of knowledge and skills, then code writing is at the top of the hierarchy, and our analysis will continue our earlier work (e.g. Lopez et al. 2008) of searching for statistical relationships between code writing and what we believe are precursor skills for code writing.

4.4 Other Data to be Collected

Where it is possible, collect the gender of every student from whom data is collected.

Where data is collected from volunteers, record the age of the volunteer. If data is collected via an exam given to the class, it is only necessary to be able to describe the age range of most students (e.g. “almost all students are between 18 and 21 years of age”).

Where data is collected from volunteers, record the stage they have reached in their formal study of programming at your institution (e.g. ‘eight weeks into their second semester of learning programming’). Also record whether the degree is undergraduate or postgraduate.

4.5 2009.1 (Wellington) Membership

To become a member of BRACElet, data may be collected either through an exam given to students, or as a paper-based test given to student volunteers, or through a ‘think out loud’ protocol with either students or

colleagues. The data collected is to be made available to all members of BRACElet.

In cases where data is gathered as part of assessment, or by volunteers completing a written test, data from at least 20 students is required, to enable statistical analysis.

Where it is not possible to collect data from 20 students, BRACElet membership can be obtained by conducting 'think out loud' sessions, with student volunteers or with colleagues.

5 Conclusion

Researchers who work within paradigms have agreed approaches and standards of evidence that allow them to work for long periods within their respective institutions. Peer reviewed papers are their primary means of communicating with researchers in other institutions. Computing education research has not yet developed its own paradigms. We do not have strong community agreement on research approaches and what constitutes evidence. Consequently, BRACElet participants have deliberately chosen to work more closely than is usual in research, across institutional boundaries, because BRACElet is as much about devising research approaches, and debating what constitutes valid evidence, as it is about studying novice programmers. We invite you to join the project. Even if you choose not to join BRACElet itself, you are welcome to execute the research plan we have published in this paper – our plan is now in the public domain, so anyone is free to use it. We look forward to reading about your results.

Acknowledgements

The authors thank their collaborators on the BRACElet project, especially our co-leader Tony Clear. Some of the past work on BRACElet has been funded by ACM SIGCSE Special Projects Grants awarded to Jacqueline Whalley and Tony Clear. Some of the current work on BRACElet is funded by an Associate Fellowship awarded to Raymond Lister and Jenny Edwards from the Australian Learning and Teaching Council (formerly the Carrick Institute).

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**(3): 483-495.
- Anderson, L.W., Krathwohl, D.R., Airasian, P.W., Cruikshank, K.A., Mayer, R.E., Pintrich, P.R., Raths, R. and Wittrock, M.C. (Eds) (2001): *A Taxonomy for Learning, Teaching and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives*. New York, Addison Wesley Longman, Inc.
- Biggs, J. B., and Collis, K. F. (1982): *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. New York: Academic Press.
- Carr, W., and Kemmis, S. (1986): *Becoming critical: education knowledge and action research*. Lewes: Falmer Press.
- Chase, W. C., and Simon, H. A. (1973): Perception in chess. *Cognitive Psychology*, **4**: 55-81.
- Clear, T., Edwards, J., Lister, R., Simon, B., Thompson, E. and Whalley, J. (2008a). The teaching of novice computer programmers: bringing the scholarly-research approach to Australia. *Proc. Tenth Australasian Computing Education Conference (ACE 2008)*, Wollongong, NSW, Australia. Simon and Hamilton, M., Eds., CRPIT, **78**:63-68, Australian Computer Society, Darlinghurst, Australia.
- Clear, T., Whalley, J., Lister, R., Carbone, A., Hu, M., Sheard, J., Simon, B., Thompson, E. (2008b): Reliably Classifying Novice Programmer Exam Responses using the SOLO Taxonomy. *Proc. 21st Annual Conference of the National Advisory Committee on Computing Qualifications (NACCCQ 2008)*, Auckland, New Zealand. Samuel Mann and Mike Lopez., Eds., 23-30.
- Clear, T., Philpott, A., Robbins, P., and Simon. (2008c): Report on the Eighth BRACElet Workshop (BRACElet Technical Report No. 01/08). Auckland: Auckland University of Technology.
- Denny, P., Luxton-Reilly, A. and Simon, B. (2008): Evaluating a New Exam Question: Parsons Problems. *Proc. of the 2008 International Workshop on Computing Education Research (ICER '08)*, Sydney, Australia. ACM Press, New York, NY.
- Fincher, S, and Petre, M. (2004): *Computer Science Education Research*, Taylor & Francis.
- Fincher, S, Lister, R, Clear, T, Robins, A, Tenenberg, J, and Petre, M. (2005): Multi-institutional, multi-national studies in CSEd Research: some design considerations and trade-offs. *Proc. International Computing Education Research Workshop*. Seattle, USA, 111-121.
- Ginat, D. (2008): Learning from wrong and creative algorithm design. *Proc. of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08)*, Portland, OR, USA, 26-30, ACM Press, New York, NY.
- Lister R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, E., Sanders, K., Seppälä, O., Simon, B., and Thomas, L. (2004): A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *SIGCSE Bulletin*, **36**(4):119-150.
- Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. (2006): Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *Proc. of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education (ITICSE '06)*, Bologna, Italy, 118-122 ACM Press, NY.
- Lister, R. (2007): The Neglected Middle Novice Programmer: Reading and Writing without Abstracting. *Proc. of the 20th Annual Conference of the National Advisory Committee on Computing Qualifications (NACCCQ'07)*, Port Nelson, New Zealand, Mann, S. and Bridgeman, N., Eds, 133-140.
- Lopez, M., Whalley, J., Robbins P. and Lister, R. (2008): Relationships between reading, tracing and writing

- skills in introductory programming. *Proc. of the 2008 International Workshop on Computing Education Research (ICER '08)*, Sydney, Australia. ACM Press, New York, NY.
- McCartney, R., Moström, J. E., Sanders, K. and Seppala O. (2004): Questions, Annotations, and Institutions: observations from a study of novice programmers. *Proc. Of the Fourth Finnish/Baltic Sea Conference on Computer Science Education*, Koli, Finland, 11-19.
- Parsons, D. and Haden, P. (2006): Parsons' programming puzzles: a fun and effective learning tool for first programming courses. *Proc of the 8th Australian Conference on Computing Education*, Hobart, Australia, D. Tolhurst and S. Mann, Eds., CRPIT, **78**: 157-163.
- Philpott, A, Robbins, P., and Whalley, J. (2007): Accessing The Steps on the Road to Relational Thinking. *Proc. 20th Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ 2007)*, Mann, S and Bridgeman, N., Eds, Nelson, NZ, 286.
- Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., and Whalley, J. (2008): Going SOLO to Assess Novice Programmers. *Proc. of the 13th Annual SIGCSE Conference on innovation and Technology in Computer Science Education (ITICSE '08)*, Madrid, Spain, ACM Press, New York, NY, 209-213.
- Sierra, K. and Bates, B. (2005): *Head First Java*, 2nd Edition, O'Reilly Media, Inc.; 2nd edition.
- Thomas, L., Ratcliffe, M., and Thomasson, B. (2004): Scaffolding with object diagrams in first year programming classes: some unexpected results. *SIGCSE Bulletin*, **36**(1):250-254.
- Thompson, E., Whalley, J., Lister, R. and Simon, B. (2006): Code Classification as a Learning and Assessment Exercise for Novice Programmers. *Proc. of the 19th Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ 2006)*, Wellington, New Zealand, 291-298.
- Thompson, E., Luxton-Reilly, A., Whalley, J., Hu, M., and Robbins, P. (2008): Blooms Taxonomy for CS Assessment. *Proc. Tenth Australasian Computing Education Conference (ACE2008)*, Wollongong, Australia, Simon & M. Hamilton, Eds., CRPIT, **78**: 155-161.
- Traynor, D., Bergin, S., and Gibson, J. P. (2006): Automated assessment in CS1. *Proc. of the 8th Australian Conference on Computing Education - Volume 52*, D. Tolhurst and S. Mann, Eds., Hobart, Australia, *ACM International Conference Proceeding Series*, **165**: 223-228. Australian Computer Society, Darlinghurst, Australia,
- Whalley, J., Lister, R., Thompson, E., Clear, T, Robbins, P., and Prasad, C. (2006): An Australasian Study of Reading and Comprehension Skills in Novice Programmers, using the Bloom and SOLO Taxonomies. *Proc. of the 8th Australian Conference on Computing Education - Volume 52*, D. Tolhurst and S. Mann, Eds., Hobart, Australia, *ACM International Conference Proceeding Series*, **165**: 243-252, Australian Computer Society, Darlinghurst, Australia.
- Whalley, J. (2006): CSEd research instrument design: the localization problem. *Proc. 19th Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ 2006)*, S. Mann & N. Bridgeman, Eds., Wellington, NZ, 307-312.
- Whalley, J., Prasad, C. and Kumar, P. K. A. (2007a): Decoding Doodles: Novice Programmers and Their Annotations. *Proc. Ninth Australasian Computing Education Conference (ACE 2007)*, Ballarat, Vic., Australia. Mann, S. and Simon, Eds., CRPIT, **66**: 171-178.
- Whalley, J. L., and Robbins, P. (2007b): Report on the Fourth BRACElet Workshop. *Bulletin of Applied Computing and IT*, **5**(1), Retrieved June 7, 2007 from http://www.naccq.co.nz/bacit/0501/2007Whalley_BRA CELET_Workshop.htm
- Whalley, J., Clear, T. and Lister, R. (2007c): The Many Ways of the BRACElet Project. *Bulletin of Applied Computing and IT*, **5**(1), Retrieved June 7, 2007 from http://www.naccq.co.nz/bacit/0501/2007Whalley_BRA CELET_Ways.htm