AUSTRALIAN LEARNING & TEACHING COUNCIL

Promoting excellence in higher education

Fellowship report

TEACHING NOVICE COMPUTER PROGRAMMERS: bringing the scholarly approach to Australia

A report on the BRACElet project

Dr Raymond Lister
Professor Jenny Edwards
University of Technology, Sydney

2010

UNIVERSITY OF TECHNOLOGY SYDNEY

The Australian Learning and Teaching Council is an initiative of the Australian Government Department of Education, Employment and Workplace Relations.

Support for this fellowship has been provided by the Australian Learning and Teaching Council, an initiative of the Australian Government Department of Education, Employment and Workplace Relations. The views expressed in this report do not necessarily reflect the views of the Australian Learning and Teaching Council Ltd.

ISBN: 1-921856-02-5

2010

Photo on the title page courtesy of Dr Tony Clear, Auckland University of Technology

Report Contents

Executive summary

The philosophical and motivational foundations of this project are well summed up by the following 25 year-old quotation from a British academic, Eric Ashby: *For many years I taught in universities. ... I marked thousands of examination scripts without examining what the scripts could teach me about my capacity as teacher and examiner.*

The academics involved in this fellowship, both the fellows and their collaborators, are from the discipline of information and communication technology. For the fellowship we have adopted an action research approach for systematically collecting evidence from end-of-semester exams, with the aim of subsequently improving the teaching of computer programming. As part of this process the project participants formulated ideas on where the problems lay for novice programmers, devised exam questions to test these ideas, collected and analysed the data from the end-of-semester exams − and then repeated the process.

Contrary to the intuitions of many computing academics, the project participants have found that students tend not to have problems with the low level 'nuts and bolts' of programming. Instead they have difficulties fitting the pieces together to see the larger picture − they can't see the forest for the trees. Many traditional exam questions, however, still test the novice programmer only on the lower level 'nuts and bolts'.

During the funding period of the fellowship, three workshops have been held within Australia. A total of 21 Australian academics, from 14 different Australian universities, have either attended these workshops or actively participated in the project electronically. Academics at six Australian universities (in addition to the University of Technology, Sydney) have used end-of-semester exam questions that were designed as part of this project. The project has also attracted international attention. Fourteen universities from seven countries have actively participated in data collection and analysis. During the ALTC Fellowship funding period, 26 project participants have (co) authored 16 published papers, further disseminating the outcomes of the project.

At the most recent project workshop, held at Queensland University of Technology in January 2010, the workshop attendees identified four new activities or 'threads' for the coming year. We therefore expect that the project – and its evidence-based, multi-institutional collaborative mindset – will continue to live beyond the period of funding provided by this ALTC Fellowship.

1. Introduction

1.1 Problem: lower ICT enrolments and programming

Over the last decade, across the western world, student numbers in the discipline of information and communication technology (ICT) plummeted, especially the numbers of female enrolments. Even the students who do enrol in introductory programming courses have poor retention into further ICT courses. In 2004 McGettrick et al. noted that: “educators cite failure in introductory programming courses and/or disenchantment with programming as major factors underlying poor student retention".

A student’s journey from novice to expert computer programmer is poorly understood by computing academics. Clearly, experts know more than novices, but psychology research indicates that experts also organise that knowledge into more sophisticated and flexible forms (Chi, Glaser & Farr, 1988; Ericsson & Smith, 1991). Studies show that expert programmers form abstract representations based upon what the code does whereas novices form concrete representations based on the code itself (see, for example, Adelson, 1984; Fix, Wiedenbeck & Scholtz, 1993).

Soloway et al. (1983) found that just 38 per cent of computer programming students could, after one semester, write code to input some numbers and output the average – a very elementary program, that most computing academics believe a student should be able to write after one semester. An entire volume of papers, called *Studying the novice programmer*, also documented the difficulties of learning to program (Spohrer and Soloway, 1989). An Information and Technology in Computer Science Education (ITiCSE) international working group (McCracken et al., 2001) studied the competency of novice programmers at four universities in two countries, and found that the great majority of the students performed far more poorly than expected.

1.2 Problem: the folk-pedagogic approach to teaching

Few computing academics are aware of the afore-mentioned literature. Instead computing academics tend to speak as if the difficulty many students face with programming is a new phenomenon and one which is peculiar to their own institution. Furthermore, computing academics tend to speak as if the difficulty is due to factors attributable and correctable entirely within their own institution (eg the quality of the lecturing, the choice of textbook or the choice of programming language). Many computing academics try to address the quality of lecturing and change textbooks frequently but the choice of programming language is often a department decision over which individuals may have only limited influence. This inward looking, institutional focus is a way of thinking, however, that is not well-suited to solving a problem endemic, world-wide.

Most academics lead a double life: the way in which they approach their teaching is markedly different from the way they approach their research. In their research lives, academics focus upon evidence and are part of a community that reaches beyond their own university. In contrast, the teaching life of most academics is not grounded in literature, and is relatively isolated. Ashby (1963) described the contradiction between the research and teaching lives of academics thus:

> … all over the country … scholars … who would not make a decision about the shape of a leaf or the derivation of a word or the author of a manuscript without painstakingly assembling the evidence, make decisions about … content of courses, and similar issues, based on dubious assumptions, scrappy data, and mere hunch.

Continuing along this line of thinking, Ashby (1984) subsequently wrote:

> For many years I taught in universities. Like most academics I assumed that the only

> qualification I needed was expertise in the discipline I taught. … I marked thousands of examination scripts without examining what the scripts could teach me about my capacity as teacher and examiner.

Echoing Ashby's observations, Bruner (1996) coined the term 'folk pedagogy' to describe "intuitive theories about how other minds work". Bruner added that such folk pedagogies "badly want some deconstructing if their implications are to be appreciated".

Bruner's use of the term 'folk pedagogy' reminds us of the common term 'folk medicine', which has been defined thus:

> Traditional medicine as practiced by non-professional healers or embodied in local custom or lore…
>
> from http://medical-dictionary.thefreedictionary.com/folk+medicine

If we replace 'medicine' with 'pedagogy' and 'healers' with 'teachers' then the above quote becomes a workable definition of folk pedagogy:

> Traditional pedagogy as practiced by non-professional teachers or embodied in local custom or lore…

The following description of folk medicine has been edited to provide a further description of folk pedagogy:

> Folk ~~medicine~~ [pedagogy] … is a category of informal knowledge distinct from 'scientific ~~medicine~~ [pedagogy]' … is usually unwritten and transmitted orally ... [and] … may be diffusely known by many ~~adults~~ [teachers] … [Folk ~~medicine~~/pedagogy is] … not necessarily integrated into a coherent system, and may be contradictory. Folk ~~medicine~~ [pedagogy] is sometimes associated with quackery … [but] … it may also preserve important knowledge and cultural tradition from the past.
>
> from
> http://en.wikibooks.org/wiki/Introduction_to_Sociology/Health_and_Medicine#Folk_Medicine

In expressing a similar sentiment to that of Bruner, Boyer (1990) argued that academic work could transcend the 'teaching versus research' dialectic, if we saw academic work as comprised of four equally important areas of scholarship, one of these areas being the 'scholarship of teaching'.

Academics need not lead double lives. There is no reason why academics cannot approach their teaching with the same mindset they apply to their research – we can focus upon evidence and be part of an education-related community that reaches beyond our own university.

Before their fellowship, both Jenny Edwards and Ray Lister had worked in such education-related communities. Edwards has chaired Australian Computer Science (ACS) accreditation committees and review committees at computing departments in a number of Australasian universities. Lister has been involved in multi-institutional computing education research projects.

The joint fellowship arose from these experiences; Lister's previous research into the teaching and learning of introductory programming; and Edwards' previous research on attracting and retaining more women in computing. In addition, we felt that Lister's activities and leadership in Australasian (ACE) and international computer science education groups – Special Interest Group on Computer Science Education (SIGCSE) and ITiCSE – and Edwards' broader activities and leadership in Computing Research and Education (CORE), the association of Australasian Computing academics, and Australian Computer Society accreditations, meant that the work of the fellowship would be more likely to have a wider impact than if either of us had worked separately.

2. Background: before the fellowship

In 2004, Lister led the ITiCSE Leeds working group (Lister et al., 2004). The group comprised 12 academics, all from different institutions, across seven different countries. The working group studied the program-reading skills of novice programmers by inserting a common set of questions into their end-of-semester exams.

The working group had to grapple with the fundamental problem of any multi-institutional study of student learning: how is it possible to compare students across institutions? Are there not too many differences between the educational environments (ie confounding factors) to prevent valid comparisons being made across institutions? One obvious potential difference is that an elite institution would probably claim to have better students than many other institutions. The working group addressed this particular problem by not looking at student performance in absolute terms (ie how many students overall, and at each institution, answered a particular question correctly), but instead in relative terms. For example, suppose students at institutions A and B are both given two questions, Q1 and Q2. While students at institution A might do better on both Q1 and Q2 than students at institution B, the working group found that if Q1 was substantially more difficult than Q2 for students at institution A, then it tended to be the case that Q1 was also substantially more difficult than Q2 for students at institution B. Using that general approach, other comparisons were made across institutions, and other regularities were found.

The power of such multi-institutional collaborations is that regularities found across institutions cannot be explained by factors that vary across the institutions. Therefore, the factors that folk pedagogues most commonly talk about (the quality of the lecturing, the choice of textbook, and the choice of programming language) are irrelevant. Factors more fundamental must be the cause. Thus, via a multi-institutional approach, issues in teaching are moved beyond folk pedagogy, and studied in a more systematic, scientific manner.

The BRACElet[1] project began in New Zealand, at the instigation of Tony Clear, after he heard Raymond Lister give a presentation on the results from the Leeds Working group. Clear invited Lister to the first BRACElet workshop, in December 2004.

The workshop participants then devised their own set of questions, using a framework based upon the revised version of Bloom's taxonomy (Anderson et al., 2001). These questions were then included in the end-of-first-semester exams that students attempted at the participating New Zealand institutions in June 2005. The 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. By December 2007, six workshops had been held in New Zealand. The seventh BRACElet workshop, which was the first workshop held as part of this ALTC Fellowship, was the first in Australia and was held in January 2008. Clear et al. (2009/2010) provide a summary of the first eight workshops.

[1] The "BRACE" in BRACElet stands for Building Research in Australasian Computing Education.

2.1 Action research

As the BRACElet project matured in New Zealand, the project participants began to see the project as being a type of action research. Action research has been described thus:

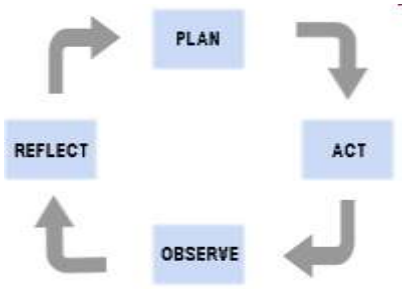
> … a reflective process of progressive problem solving led by individuals working with others in teams or as part of a 'community of practice' to improve the way they address issues and solve problems. ... As designers and stakeholders, researchers work with others to propose a new course of action to help their community improve its work practices. Kurt Lewin, then a professor at MIT, first coined the term 'action research' in about 1944. In his 1946 paper *Action research and minority problems* he described action research as … [using] … "a spiral of steps, each of which is composed of a circle of planning, action, and fact-finding about the result of the action".
>
> Wikipedia, April 2010

The particular action research cycle we used in this project consisted of four steps: plan, act, observe, and reflect. The activities within each of those steps are described below.

The action research cycle as used in this project

Plan: Done at the workshops. Activities included setting research questions for the coming cycle, and devising the general style and framework of exam questions to answer those research questions.

Act: Done after the workshops, in small groups, via email and voice-over-the-internet. Specific exam questions were written.



Observe: Conducted the examinations at each institution, collected and collated the data. Conducted preliminary analysis within each institution.

Reflect: Done at the workshops. Activities included in-depth analysis of the data, including cross-institutional comparisons, to confirm or deny the research questions of the current cycle. The writing of a paper for publication was a common vehicle for driving this process.

2.2 The SOLO taxonomy

Consistent with the principles of action research, we came to take a social constructivist perspective on our project, in which we saw "… 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 & Kemmis, 1986, p. 28).

Our aim was to stay close to educational practice, by focussing on the assessment of students via exam papers. In our view, detailed work on how to teach programming first requires that we have assessment practices that are valid and reliable − it is not clear that current examination practices for programming are either valid or reliable.

At the second BRACElet workshop, in July 2005, we first made the connection between what we saw in students' responses to exam questions and the Structure of Observed Learning Outcomes (SOLO) taxonomy (Biggs and Collis 1982). The taxonomy is concerned with describing the quality of a student's understanding of a topic (as elicited, in our case, via an exam question), from the lowest and least sophisticated 'pre-structural' level, up four levels to the highest 'extended abstract' level. Each level is an indication of how well the student has integrated the knowledge taught into a coherent, meaningful whole. The following table summarises the five levels.

Extended abstract	The student response goes beyond the coherent whole, to place the topic area into its broader context.
Relational	The student response describes the relationships between the facts in such as way as to articulate a coherent whole.
Multi-structural	The student response includes many connections among the facts, but major connections that render a coherent 'big picture' are not articulated.
Uni-structural	The student response includes simple, obvious connections made between the facts, but the significance of the connections is not articulated.
Pre-structural	The student responds with isolated facts.

How the SOLO taxonomy relates to programming is described in Appendix C: A technical summary of recent results from the project, for the ICT academic reader. To summarise our overall findings in terms that can be understood by someone outside the ICT discipline: we have found that when students are asked to explain what a piece of code does, they frequently respond at the multi-structural level, but struggle to respond at the relational level. In other words, students can frequently tell us what each line of a program does, but they cannot tell us what that program as a whole does. Furthermore, we have found that students who cannot articulate a relational response frequently struggle to write coherent programs of their own.

3. Fellowship workshops

As described in the previous section, the 'plan' and 'reflect' phases of the action research cycle happen at workshops. Most of our workshops were organised to happen in conjunction with a conference.

Each year in January the computing academics of Australia and New Zealand (and a growing number of internationals) hold Australasian Computer Science Week (ACSW). One of the constituent conferences, and the largest, is Australasian Computing Education (ACE). The 2008, 2009 and 2010 workshops were direct contributors to this fellowship. In addition, when International Computing Education Research (ICER) was held in Sydney in 2008, a two-day BRACElet/fellowship workshop was held at the end of that conference, with many additional international participants.

In between workshops, the fellows, but particularly Lister, gave one-on-one advice to participants about question development. Most frequently, this advice was communicated by email, or voice-over-the-internet, but sometimes 'site visits' were made to collaborators in Queensland and Victoria.

Note**:** throughout the time in which the following ALTC funded workshops were held, the New Zealanders were continuing to run their workshops, usually in association with the annual National Advisory Committee on Computing Qualifications (NACCQ) conference. Those New Zealand workshops continued to make large, valuable contributions to the overall BRACElet project, but as those workshops were not funded by the ALTC, those workshops are not described here. Several New Zealanders attended one or more of the following ALTC-funded workshops.

3.1 ACE '08 Workshop (Wollongong, January)

This was the first workshop funded under the ALTC Fellowship, but the seventh BRACElet workshop overall. It was a half-day event, held at the end of the ACE2008 conference, in Wollongong. Academics from over a dozen institutions attended the workshop.

As BRACElet was new to most Australian academics, this workshop was primarily about introducing them to the project, what it did, what we had found thus far, and how we went about collaborating. Our aim was to get the attending academics to try some BRACElet-style questions in their end-of-semester exams in June. This aim was more about getting the Australian academics to replicate earlier BRACElet work, rather than extend the existing work.

As briefly described earlier, what we had found up to that point in BRACElet was that when students are asked to explain what a piece of a program does, they frequently respond at the SOLO multi-structural level, but struggle to respond at the relational level − they articulate what each line of a program does, but not what the program does as whole. Much of the discussion focused around this issue. The discussion was first about explaining the SOLO taxonomy. As the discussion continued, two objections were raised:

1) Perhaps students thought that a line-by-line, multi-structural answer was what they were being asked to provide? Prior to this workshop, when we had asked students to explain what a piece of code did, our instruction to students had been the ambiguous "In plain English, explain what the following segment of code does".
2) Perhaps reading/explaining code is a separate intellectual activity from writing code?

The pedagogical significance of this issue is discussed in Appendix C: A technical summary of recent results from the project, for the ICT Academic Reader. For the non-ICT reader, we summarise the argument as follows: the traditional approach to teaching programming places very heavy emphasis on students writing code, whereas the BRACElet results might suggest that there should greater emphasis on having students read code.

Many of today's ICT academics were novices when programming was done with punched cards and overnight batch runs. In such an environment, teachers did not need to encourage students explicitly to think carefully about their code before attempting their next compile-and-run. If students did otherwise, a careless error could waste a whole day. For today's novices, however, who are learning in an era where the next compile-and-run is only a mouse-click away, it is tantalisingly easy, but ultimately futile, for a novice to pursue a strategy of trying to get the computer to do the thinking for them. Today's educator needs to place greater explicit pedagogical emphasis on the importance, and practice of, the tracing and explaining skills that lead to systematic code writing.

In addressing questions such as those above, the aim of the BRACElet project is to avoid lapsing back into folk pedagogy. Instead, our aim is to devise scholarly, evidence-based ways to answer the questions.

Two scholarly, evidence-based ways to addressing the first question emerged from further discussion. One approach was simply not to confront the students with this type of question for the first time in the end-of-semester exam. Instead, this type of question would be discussed with them before the exam. The type of answer expected would be made clear to them. The other approach was to experiment with other ways of presenting the question to the students. Both of these approaches were pursued during the fellowship.

The second of the above questions could be investigated directly. A cohort of students could be asked both to explain code and to write code. Subsequent analysis would look for a statistical correlation between explaining code and writing code. This approach was also pursued within the fellowship.

3.2 ICER '08 Workshop (Sydney, September)

This workshop was a one-and-one-half-day event, held immediately after the ICER 2008 conference, in Sydney. As ICER is an international conference, the workshop attracted a number of people who had not been part of any previous workshop. The first half-day therefore covered similar ground to the ACE'08 workshop.

Before the fellowship workshop, a BRACElet paper (Lopez et al., 2008) had been presented at ICER. That paper contained results showing a statistical relationship between explaining code and writing code. Thus the discussion of a relationship, if any, between reading/explaining code and writing code took its first step away from a folk-pedagogic debate, toward an evidence-based debate. The discussion began to move beyond whether there was any relationship between reading/explaining code and writing code, to a discussion of the nature of the relationship – was it a causal relationship, or was it a manifestation of a common, underlying deeper skill? That discussion is ongoing.

Of course, one paper, written from one data set, collected at one institution, could not establish beyond doubt that there was a relationship between reading/explaining and writing code. As a consequence of the 'reflect' stage at this workshop, analysis began on other data, from exams conducted in June 2008, with the aim of replicating the type of analysis described in the Lopez et al. study. The first of those replications would eventually be published in 2009.

3.3 ACE '09 Workshop (Wellington, NZ, January)

This was a half-day event, held at the end of the ACE2009 conference, in Wellington. Seventeen academics attended from 12 different institutions, including 6 different Australian universities – University of Technology, Sydney (UTS); University of Southern Queensland (USQ); Monash University; RMIT; University of Newcastle; and Queensland University of Technology (QUT).

By this stage of the BRACElet project, our methods of collecting and analysing data had matured to the point where we could begin to document it. An opportunity to do that presented itself when Whalley and Lister were invited to present a paper on the BRACElet project at ACE2009. The purpose of this paper was to provide potential new participants with an overview of BRACElet, and to specify the type of data that would be collected in the next action research cycle. In publishing that specification, we were taking the unusual step – some might say the unwise step − of making our study design public before doing the actual study. Such an approach works in the BRACElet project because of its multi-institutional nature. It would not matter if someone outside BRACElet took that design and published their own results for it. From our perspective, that would be an excellent mechanism of dissemination. Another advantage of the invited paper, which was presented as part of the ACE2009 conference and before the actual BRACElet workshop, was that it removed the time-consuming burden of beginning the workshop with yet another introduction of the project to any newcomers. Those introductions wasted the time of established participants, and were beginning to try their patience.

At this workshop, most of the discussion was organised around a reading of a draft BRACElet paper that was eventually published mid-year (Lister, Fidge and Teague, 2009). That aim of that paper was to replicate the Lopez et al. (2008) study presented at ICER 2008, using data collected from a different exam paper, conducted at a different educational institution. Also, this study used a different statistical approach from Lopez et al. (2008). The results presented at the workshop were consistent with Lopez et al. (2008). Furthermore, the similarities and differences between this paper and the Lopez et al. paper led to a good discussion on what is essential, and what is not essential, to such a BRACElet study.

3.4 ITiCSE '09 Working Group (Paris, July)

This working group was the first systematic attempt to expand BRACElet beyond New Zealand and Australia. This group eventually published a paper (Lister et al., 2010) with 12 authors from 11 different institutions, in seven countries. This paper included multiple replications of earlier BRACElet analysis, using a far broader pool of data, a refinement of the SOLO taxonomy for code-explaining questions, an extension of the SOLO taxonomy to code-writing questions, an extension of some earlier studies on student annotations ('doodles') and an exploration of a theoretical basis for BRACElet work (see the subsection below).

The replication work was a comprehensive answer to the questions first raised at the ACE2008 workshop, as to whether there was any relationship between code explaining and code writing. In doing so, the BRACElet project settled an issue with evidence and scholarship. The question remains, however, about the nature of the relationship.

Relationship to mathematics education research

It was at this workshop where project participants began to connect their BRACElet work with the SOLO taxonomy to the abundant research in mathematics education. One might say that we had reached a point where we could manifest an understanding of the SOLO taxonomy at the highest SOLO level, the extended abstract level. For example, Sfard (1988 & 1991) dichotomised mathematical

knowledge as follows:

- Operational or process understanding considers how something can be computed; the concept is regarded as an algorithm for manipulating things. Such knowledge allows a student to apply a concept to some data. For example, an operational view of the function $y = 3x^2$ is the process of squaring a number, represented by x, and multiplying the result by 3. We see that operational view as being a manifestation of the multi-structural level, or lower, in the SOLO taxonomy.
- Structural or object understanding describes a concept by its properties, treating it as a single entity. Such knowledge allows a student to reason about the concept, by treating the concept itself as data. For example, one structural view of $y = 3x^2$ is as a parabola in the Cartesian plane. We see that structural view as being a manifestation of the relational level in the SOLO taxonomy.

Sfard describes a three-phase process of concept formation, from process to object understanding:

- *Interiorisation*: the student becomes familiar with applying the process to data.
- *Condensation*: the student abstracts the process into more manageable chunks. Ultimately the student may abstract the process into its input-output behaviour (in computing terms), but will still view the concept as an algorithmic process.
- *Reification*: the student views the concept as a unified entity. The concept is understood by its characteristics, and can be manipulated as a primitive object in other concepts.

Once a concept is reified, Sfard claimed it can be used as a primitive in the formation of an even higher-level concept. Similarly, Biggs and Collis (1982) described a spiral approach to the SOLO taxonomy, where a relational level of understanding on one turn of the spiral became a unistructural understanding at the next turn of the spiral.

The work of connecting BRACElet to a wider set of literature was only begun by this ITiCSE working group, and it is likely to be ongoing.

3.5 ACE '10 Workshop (Brisbane, January)

This was a half-day event, held at the end of the ACE2010 conference, in Brisbane. A total of 23 academics attended, including 14 Australians. The Australians came from 11 different institutions: CQUniversity; Edith Cowan University; La Trobe University; Monash University; The University of Newcastle; QUT; Swinburne University of Technology; USQ; UTS; Victoria University; and the University of Wollongong. Apart from representatives of three New Zealand universities, there were also three participants from Finland, Indonesia and the USA.

A feature of this workshop was the way the leadership of BRACElet was broadening, with some participants who had not previously taken on a leadership role in BRACElet stepping up to describe where they wanted to take the project. These people outlined their plans for BRACElet activities (now known as 'threads') in the coming year. All of these people extended to the other workshop attendees an invitation to collaborate on these activities, in the BRACElet tradition. The threads, and their leaders, are as follows:

- Michael de Raadt (USQ) demonstrated a prototype of his web-based tool, with which he plans to perform a repetition of Wiedenbeck (1985).

- Judy Sheard (Monash), Angela Carbone (Monash) Jacquie Whalley (Auckland University of Technology, NZ), Mikko Laakso (Finland, on sabbatical at Monash) and Donald Chinn (University of Washington, USA) intend to carry out a survey/analysis of exam papers used to test novice programmers, looking at (for example) the types of exam questions used. They also plan to interview academics about how they set exam papers.
- Jacquie Whalley (Auckland University of Technology, NZ) described the longitudinal assessment data she has for students at her institution, across various courses from CS1 to capstone. She intends to analyse this data.
- Mikko Laakso (Finland, on sabbatical at Monash) demonstrated ViLLE (Kaila, Rajala, Laakso and Salakoski, 2010), his web-based program visualisation tool, which he proposed could be used as a platform for multi-institutional, multi-national BRACElet studies.

Also, Raymond Lister described the past BRACElet work, and plans for the continuation of that style of work. He distributed drafts of the appendices F and G in this report.

Michael de Raadt (USA) agreed to set up a wiki to support BRACElet, which is at http://community.usq.edu.au/course/view.php?id=71 (login required). This login is available to current and prospective BRACElet participants. The material on the wiki forms part of the documentation for each cycle of the BRACElet action research project.

4. Problems encountered

Despite the good progress made during the fellowship, some problems were encountered.

4.1 Ethics

The planned activities of the fellowship received ethics clearance at UTS with no difficulty. The application was submitted as a multi-institution project and all the then known participants and their institutions were named on the application. It was around this time that new guidelines for Australian University ethics clearance were released. One of the welcome clauses was that if ethics had been obtained at one institution for a multi-institution project, that clearance was valid for all the other institutions. Our application and clearance letter were supplied to all participants. Nevertheless, some of the other institutions still required individual participants to obtain ethics clearance at their own universities.

A more serious issue, and one we raised at ALTC Fellows workshops, is the necessity for ethics clearance at all in some circumstances. In the simplest version of our research, participants tried certain types of exam questions and written assessments and then analysed the results. This involved no contact with the students or any information from them. Yet almost all universities required an ethics clearance just for this step. We would argue that as academics, this is something that ought not to be happening. If we try a new approach to presenting material, we analyse the approach for its efficacy and either adopt it or try something else. We do not require ethics clearance for this. Why then do we require ethics clearance for examining exam questions? Indeed, who would know in many instances?

An unforeseen extension of this problem occurred when we tried to make multi-institution comparisons. The project plan was for participants to set the same questions in their own settings. Then, depending on the particular style of question being examined, each participant would provide us with either marks, an analysis of the marks or the raw answers just for those particular questions (not the whole exam paper). In all cases, the data would be anonymised, both for the individuals concerned and the institution. We could then analyse the data and, where appropriate, classify the responses according to the SOLO taxonomy. A number of the participants' institutions refused to allow even anonymised data to be submitted to the project. Their reasons were that student assessments were being used without their consent or, despite all assurances and explicit statements to the contrary, they were concerned that the data would be used for comparison rather than pooling purposes.

To address the former, students were asked for their consent. Many students were reluctant to allow their answers to some exam questions to be used for the research, even though nothing was required of them.

Those who were willing to participate were coded and the consent form also asked for their gender and their English language background because these were important to our analysis.

The existing practices of ethical clearance pose a threat to the routine use of a scholarly, evidence-based approach to teaching.

4.2 Gender

As discussed in the introduction, both academics and the IT industry have serious concerns over the dwindling number of females entering or remaining in IT courses and hence, entering the profession. Previous studies had shown that the teaching of first year programming was a major factor in the attrition of females from IT courses. By putting considerable thought into the assessment and related teaching and

learning of first year programming, we hoped to alleviate this issue somewhat.

Perhaps naïvely, we based this aspect of our fellowship nomination on analyses we were able to perform at UTS and had used in previous studies relating to women in computing. In many institutions, participants in this program were unable or not permitted to obtain gender information to attach to results, even where the information was anonymised or where student administration could link student numbers to gender and then give the result an anonymous code.

Thus we were unable to do the planned gender analyses of experiments. Nevertheless, for those institutions where we were able to obtain this information, either centrally or through consent forms, the pleasing result was that for introductory programming students in both undergraduate and postgraduate courses, there were no discernible gender differences in results. Furthermore, even for the institutions for which we had no data, anecdotal evidence again was that female and English as second language students had responded well to the changed style of assessment and related learning. This contrasts greatly to other studies on the learning and teaching of introductory programming.

4.3 Two other problems

Some institutions are now demanding that 90 per cent of students achieve a mark of pass or higher. This meant that there was tighter control over assessment, exam questions and marking. Even though the participants were keen to be part of the project and actively contributed to the development of ideas, experiments and theory, we were not always able to use their results.

As will be seen from the description of dissemination below and the papers in Appendix B, many BRACElet participants, not just the fellows, have worked and are actively working to improve the assessment, learning and teaching of introductory programming and to promulgate this as widely as possible. Perhaps somewhat unfortunately, this dissemination is often at computing education conferences, where it could sometimes be regarded as preaching to the converted. Nevertheless, many of our colleagues in our universities still have no interest in a scholarly approach to teaching. Also, despite clear evidence of the effectiveness of the work being carried out in this fellowship and related projects, nor do they wish to change their often decades old approaches to the teaching and assessment of introductory programming. There is a long way to go, but BRACElet has made a good start.

5. Dissemination during the fellowship

The open, multi-institutional nature of BRACElet, and its action research orientation, makes dissemination intrinsic to the BRACElet model. Specific evidence for dissemination during the period of the fellowship (September 2007 – February 2010) includes:

- The conduct of five workshops, as described above, with over 40 different attendees, some attending multiple workshops (see Appendix A).
- The ongoing use of BRACElet-style exam questions at six Australian universities for periods ranging from one to four years. The numbers of students engaging with these exam questions each semester range from about 100 to 800 at each institution in both Australian, and where relevant, offshore campuses:
    - Victoria University, Melbourne: used in two programming courses, in two semesters.
    - USQ: used in the first programming course, for several semesters.
    - Monash University: used in two courses, for several semesters. One of the courses is run across four Australian campuses (Clayton, Caulfield, Berwick and Gippsland) and two at international campuses (South Africa and Malaysia).
    - The University of Western Australia: currently being tried for the first time, in the first programming course.
    - The University of Newcastle: used in a programming course, for several semesters. The course is run at two Australian and at overseas campuses.
    - QUT: used in a programming course for several semesters and used with a change in programming language.
    - Note that *BRACElet questions are also used at participating New Zealand institutions* on an ongoing basis.
- The publication of 16 peer-reviewed or invited papers, as listed in Appendix B, with a total of 26 authors.
    - Participation, as authors, of 14 Australian academics from five different Australian universities (UTS, Monash, QUT, Victoria, and Newcastle).
    - Participation, as authors, in the ITiCSE 2009 working group of 12 authors from 11 different institutions, in 7 countries.

There have been some spinoff activities and workshops. For example, in one Australian university (Monash), introductory programming is taught in three different departments using three different programming languages. There was at least one active participant from each of these departments in our project. They were able to conduct a side project just comparing the common exam questions in the different programming languages in their own institution.

6. Outcomes of the fellowship

A specified in our fellowship proposal, our desired outcomes were:

- **[Achieved]** The adoption of an action research, evidence-based approach to the design of assessment strategies for novice programmers.
- **[Ongoing]** A culture/community of scholarly teaching, built across institutions and some national boundaries, with a discourse based in evidence rather than anecdote, this community would be particularly useful to isolated IT academics.
- **[Achieved]** The design of assessment strategies for novice programmers that are more:
  - valid
  - reliable
  - gender neutral
  - consistent with the results from computer education research.
- **[Achieved]** The creation of a bank of appropriate material for teaching and assessing novice programmers.
- **[Ongoing]** The better learning of both reading and writing code by novice programmers.
- **[Ongoing]** The detection and elimination of any gender differences using the scholarly approach.
- **[Achieved]** The dissemination of results via workshops, ongoing activities, journal and conference papers.
- **[Achieved]** A wiki to act as a repository of banks of questions, reports, discussions, papers, etc.(login required) http://community.usq.edu.au/login/index.php?id=71
- **[Ongoing]** The eventual attraction and retention of more students in IT.

  Numbers of IT students in Australia have increased slightly in 2010. We cannot claim this is a result of our fellowship. However, we are sure that the work of the fellowship and its promulgation through ACE, BRACElet, publications, etc are raising the profile of teaching and learning in computing in Australasia. We hope that that is being reflected in computing departments and will eventually help with attraction and retention of both male and female students.
- **[Achieved]** A framework and ‘how to’ manual for running similar multiinstitutional collaborative projects in any discipline.

7. Acknowledgements

We would like to thank the ALTC for the opportunities provided by this fellowship. Not just for enabling our project activities but also for the opportunity to benefit from the friendship, the wisdom and the expertise of the band of fellows and to broaden our own educational horizons by participating in some of the many workshops, etc offered by the ALTC.

We also thank our collaborators on the BRACElet project, especially our New Zealand co-leaders, Tony Clear and Jacqui Whalley. Not only did BRACElet begin in New Zealand, but throughout the duration of our ALTC Fellowship our New Zealand collaborators continued to hold workshops, which made large and valuable contributions to the overall BRACElet project.

The fellowship activities and the ongoing legacy of the fellowship are possible only through the ongoing enthusiasm of our computing academic colleagues both in Australasia and internationally, who are prepared to put considerable effort into improving computing education, especially of first year programming, The ultimate aim is to provide a better and more attractive computing education for all students, male, female and those from a variety of cultural and educational backgrounds. Without the hard work of our colleagues, the fellowship aims would not have been achieved.

8. References (non-fellowship)

Papers written as part of the ALTC Fellowship are listed separately, in Appendix B.

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), pp. 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.

Ashby, E (1963), 'Introduction: decision making in the academic world', In Halmos, P (ed), *Sociological studies in British university education*, University of Keele. pp. 5-13.

Ashby, E (1984) from the 'Foreword' in Brewer, I. M., *Learning more and teaching less.* Guildford, Guildford: Society for Research into Higher Education.

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

Boyer, E. *Scholarship reconsidered: priorities of the professoriate.* Princeton, New Jersey: Princeton University Press: The Carnegie Foundation for the Advancement of Teaching, 1990.

Bruner, J (1996) *The culture of education*, Harvard University Press.

Carr, W., Kemmis, S. (1986) *Becoming critical: education knowledge and action research*. Lewes: Falmer Press.

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

Ericsson K, and Smith, J. (Eds) (1991) *Toward a General Theory of Expertise : Prospects and Limits*. Cambridge University Press, England.

Fix, V., Wiedenbeck, S., and Scholtz, J. (1993). 'Mental representations of programs by novices and experts'. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems* (Amsterdam, The Netherlands, April 24 - 29,1993). CHI '93. ACM, New York, NY, 74-79. DOI= http://doi.acm.org/10.1145/169059.169088

Kaila, E., Rajala, T., Laakso, M. and Salakoski, T. (2010) Effects of Course-Long Use of a Program Visualization Tool, 12th Australasian Computing Education Conference (ACE 2010), Brisbane, Australia pp. 97-106.

Lister R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, E., Sanders, K., Seppälä, O., Simon, B., Thomas, L., (2004) 'A multi-national study of reading and tracing skills in novice programmers'. *SIGCSE Bulletin*, Volume 36, Issue 4 (December), pp. 119-150.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagen, D., Kolikant, Y., Laxer, K., Thomas, L., Utting, I., Wilusz, T. (2001): 'A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-year CS Students'. *SIGCSE Bulletin*, Volume 33, Number 4, pp. 125-140.

McGettrick, A, Boyle, R, Ibbett, R, Lloyd, J, Lovegrove, L, and Mander, K. (2005) 'Grand challenges in computing: education – a summary'. *The Computer Journal.* The British Computer Society. Volume 48, Number 1, pp 42-48.

Sfard, A. (1988). Operational vs. structural method of teaching mathematics – case study. Twelfth Conference for the Psychology of Mathematics Education 2, 560-567.

Vesprèm, Hungary: Ferenc Genzwein, OOK.

Sfard, A. (1991). ‘On the dual nature of mathematical conceptions: reflections on processes and objects as different sides of the same coin’. *Educational Studies in Mathematics* 22, 1-36.

Soloway, E. and Ehrlich, K, Bonar, J., and Greenspan, J. (1983) ‘What do novices know about programming?’ In Shneiderman, B. and Badre, A. (Eds), *Directions in Human-Computer Interactions*, Ablex, Inc., Norwood, NJ, pp. 27–53.

Spohrer, J. and Soloway, E. (1989) (Eds) *Studying the novice programmer*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Wiedenbeck (1985) ‘Novice/expert differences in programming skills’. *International Journal of Man-Machine Studies* 23(4): 383-390.

Appendix A: Active attendees at workshops, and participants in various iterations of the action research project

1	Peter Andreae	Victoria University of Wellington	New Zealand
2	Steven Bird	The University of Melbourne	Australia
3	Jonas Boustedt	Uppsala University	Sweden
4	Ilona Box	University of Technology,Sydney	Australia
5	Angela Carbone	Monash University	Australia
6	Paul Carter	The University of British Columbia	Canada
7	Donald Chinn	University of Washington	USA
8	Tony Clear	Auckland University of Technology	New Zealand
9	Matt Daniels	Uppsala University	Sweden
10	Paul Denny	The University of Auckland	New Zealand
11	Michael De Raadt	University of Southern Queensland	Australia
12	Chris d'Souza	Australian Catholic University	Australia
13	Anna Eckerdal	Uppsala University	Sweden
14	Jenny Edwards	University of Technology, Sydney	Australia
15	Colin Fidge	Queensland University of Technology	Australia
16	Mark Hall	University of Wisconsin, Marathon County	USA
17	John Hamer	The University of Auckland	New Zealand
18	Margaret Hamilton	RMIT University	Australia
19	Chris Johnson	The Australian National University	Australia
20	Judy Kay	The University of Sydney	Australia
21	Alanah Kazlauskas	Australian Catholic University	Australia
22	Angel Kennedy	The University of Western Australia	Australia
23	Cat Kutay	The University of New South Wales	Australia
24	Tracy Lewis	Radford University	USA
25	Raymond Lister	University of Technology, Sydney	Australia
26	Mike Lopez	Manukau Institute of Technology	New Zealand
27	Jan Lönnberg	Helsinki University of Technology (Now the School of Science and Technology at Aalto University)	Finland
28	Andrew Luxton-Reilly	The University of Auckland	New Zealand
29	Rachel Or-Bach	Max Stern Academic College Of Emek Yezreel	Israel
30	Dale Parsons	Otago Polytechnic	New Zealand
31	Anne Philpott	Auckland University of Technology	New Zealand
32	Alex Potanin	Victoria University of Wellington	New Zealand
33	Phil Robbins	Auckland University of Technology	New Zealand
34	Judy Sheard	Monash University	Australia
35	Shuhaida Shuhidan	RMIT University	Australia
36	Simon	The University of Newcastle	Australia
37	Beth Simon	University of California, San Diego	USA
38	Wanda Smith	Virginia Tech	USA
39	Peter Strazdins	The Australian National University	Australia
40	Ken Sutton	Southern Institute of Technology	New Zealand
41	Donna Teague	Queensland University of Technology	Australia
42	Errol Thompson	Kiwi-ET Consulting	New Zealand
43	Jodi Tutty	Charles Darwin University	Australia
44	Anne Venables	Victoria University	Australia
45	Jacqui Whalley	Auckland University of Technology	New Zealand
46	Alison Young	Unitec	New Zealand

Appendix B: The BRACElet papers: written and published during the period of the fellowship (September 2007 – February 2010)

1) **Clear, T., Edwards, J., Lister, R., Simon, B., Thompson, E. and Whalley, J.** (2008). 'The teaching of novice computer programmers: bringing the scholarly-research approach to Australia'. In *Proc. Tenth Australasian Computing Education Conference* (ACE 2008), Wollongong, NSW, Australia. CRPIT, 78. Simon and Hamilton, M., Eds., ACS. 63-68. http://crpit.com/Vol78.html

Abstract: BRACElet is a multi-institutional multi-national research study of how novice programmers comprehend and write computer programs. This paper reviews the first action research cycle of the BRACElet project and, in the process, charts a path for the upcoming second cycle. The project remains close to educational practice, with much of the data being either data collected directly from exams sat by novices, or data from 'think-out-loud' protocols where the task undertaken by a novice or an expert is modelled on an exam question. The first action research cycle analysed data in terms of the SOLO taxonomy. From 'think-aloud' responses, the authors found that educators tended to manifest a SOLO relational response on small reading problems, whereas students tended to manifest a multi-structural response. Furthermore, those students who manifested a relational response tended to do better overall in the exam than students who manifested a multi-structural response. The second action research cycle will explore the relationship between the ability to read code and the ability to write code. Apart from reporting on the BRACElet project itself, this paper serves as an invitation for institutions and individuals to join the second action research cycle of the BRACElet project.

2) **Lister, R.** (2008). 'After the gold rush: toward sustainable scholarship in computing'. In *Proc. Tenth Australasian Computing Education Conference* (ACE 2008), Wollongong, NSW, Australia. CRPIT, 78. Simon and Hamilton, M., Eds., ACS. 3-18. http://crpit.com/Vol78.html

This is the paper written for the conference keynote talk.

Abstract: In just 30 years, we have gone from punched cards to Second Life. But, as the American National Science Foundation recently noted, "undergraduate computing education today often looks much as it did several decades ago". Consequently, today's 'Nintendo Generation' have voted with their feet. We bore them. The contrast between the changes wrought via computer research over the last 30 years, and the failure of computing education to adapt to those changes, is because computing academics lead a double life. In our research lives we see ourselves as part of a community that reaches beyond our own university. We read literature, we attend conferences, we publish, and the cycle repeats, with community members building upon each other's work. But in our teaching lives we rarely discuss teaching beyond our own university, we are not guided by any teaching literature; instead we simply follow our instincts. Academics in computing, or in any other discipline, can approach their teaching as research into how novices become experts. Several recent multi-institutional research collaborations have studied the development of novice programmers. This paper describes some of the results from those collaborations. The separation of our teaching and research lives diminishes not just our teaching but also our research. The modern practice of stripping away all 'distractions' to maximise research output is like the practice of stripping away rainforest to grow beef - both practices appear to work, for a little while, but not indefinitely. Twenty-first century academia needs to bring teaching and research together, to form a scholarship of computing that is an integrated, sustainable, ecological whole.

3) **Thompson, E., Luxton-Reilly, A., Whalley, J., Hu, M., & Robbins, P.** (2008). 'Bloom's Taxonomy for CS Assessment'. In Simon & M. Hamilton (Eds.), *Tenth Australasian Computing Education Conference* (ACE2008) (Vol. 78). Wollongong, Australia: Australian Computer Society Inc. http://crpit.com/Vol78.html

Abstract: Bloom's Taxonomy is difficult to apply consistently to assessment tasks in introductory programming courses. The Bloom Taxonomy is a valuable tool that could enable analysis and discussion of programming assessment if it could be interpreted consistently. We discuss each of the Bloom classification categories and provide a consistent interpretation with concrete exemplars that will allow computer science educators to utilise Bloom's Taxonomy for programming assessment. Using Bloom's Taxonomy to help design examinations could greatly improve the quality of assessment in introductory programming courses.

4) **Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., Whalley, J.** (2008) 'Going SOLO to assess novice programmers'. In *Proceedings of the 13th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. (Madrid, Spain, June 30 – July 2, 2008). ITICSE '08. ACM Press, New York, NY. pp. 209-213. http://doi.acm.org/10.1145/1384271.1384328

Abstract: This paper explores the programming knowledge of novices using Biggs' SOLO taxonomy. It builds on previous work of Lister et al. (2006) and addresses some of the criticisms of that work. The research was conducted by studying the exam scripts for 120 introductory programming students, in which three specific questions were analysed using the SOLO taxonomy. The study reports the following four findings: when the instruction to students used by Lister et al. - "In plain English, explain what the following segment of Java code does" - is replaced with a less ambiguous instruction, many students still provide multi-structural responses; students are relatively consistent in the SOLO level of their answers; student responses on SOLO reading tasks correlate positively with performance on writing tasks; postgraduates students manifest a higher level of thinking than undergraduates.

5) **Clear, T., Whalley, J., Lister, R., Carbone, A., Hu, M., Sheard, J., Simon, B., Thompson, E.** (2008) *Reliably classifying novice programmer exam responses using the SOLO taxonomy*. 21st Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ 2008), Auckland, New Zealand. Samuel Mann and Mike Lopez (Eds). http://www.naccq.ac.nz/conferences/2008/23.pdf

Abstract: Past papers of the BRACElet project have described an approach to teaching and assessing students where the students are presented with short pieces of code, and are instructed to explain, in plain English, what the code does. The student responses to these types of questions can be analysed according to the SOLO taxonomy. Some students display an understanding of the code as a single, functional whole, while other students cannot 'see the forest for the trees'. However, classifying student responses into the taxonomy is not always straightforward. This paper analyses the reliability of the SOLO taxonomy as a means of categorising student responses. The paper derives an augmented set of SOLO categories for application to the programming domain, and proposes a set of guidelines for researchers to use.

6) **Denny, P., Luxton-Reilly, A. and Simon, B.** (2008) 'Evaluating a new exam question: Parsons problems'. In *Proceedings of the 2008 International Workshop on Computing Education Research* (Sydney, Australia,

September 06 - 07, 2008). ICER '08. ACM, New York, NY. http://doi.acm.org/10.1145/1404520.1404532

Abstract: Common exam practice centres around two question types: code tracing (reading) and code writing. It is commonly believed that code tracing is easier than code writing, but it seems obvious that different skills are needed for each. These problems also differ in their value on an exam. Pedagogically, code tracing on paper is an authentic task whereas code writing on paper is less so. Yet, few instructors are willing to forgo the code writing question on an exam. Is there another way, easier to grade, that captures the 'problem solving through code creation process' we wish to examine? In this paper we propose Parsons puzzle-style problems for this purpose. We explore their potential both qualitatively, through interviews, and quantitatively through a set of CS1 exams. We find notable correlation between Parsons scores and code writing scores. We find low correlation between code writing and tracing and between Parsons and tracing. We also make the case that marks from a Parsons problem make clear what students don't know (specifically, in both syntax and logic) much less ambiguously than marks from a code writing problem. We make recommendations on the design of Parsons problems for the exam setting, discuss their potential uses and urge further investigations of Parsons problems for assessment of CS1 students.

7) **Lopez, M., Whalley, J., Robbins P. and Lister, R.** (2008) 'Relationships between reading, tracing and writing skills in introductory programming'. In *Proceedings of the 2008 International Workshop on Computing Education Research* (Sydney, Australia, September 06 - 07, 2008). ICER '08. ACM, New York, NY. http://doi.acm.org/10.1145/1404520.1404531

Abstract: This study analysed student responses to an examination, after the students had completed one semester of instruction in programming. The performance of students on code tracing tasks correlated with their performance on code writing tasks. A correlation was also found between performance on "explain in plain English" tasks and code writing. 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. Knowledge of programming constructs forms the bottom of the hierarchy, with "explain in English", Parsons puzzles, and the tracing of iterative code forming one or more intermediate levels in the hierarchy.

8) **Simon, Lopez, M., Sutton, K. and Clear, T.** (2009). Surely We Must Learn to Read before We Learn to Write!. In Proc. Eleventh Australasian Computing Education Conference (ACE 2009), Wellington, New Zealand. CRPIT, 95. Hamilton, M. and Clear, T., Eds., ACS. 165-170. http://crpit.com/confpapers/CRPITV95Simon2.pdf

Abstract: While analysing students' marks in some comparable code-reading and code-writing questions on a beginners' programming exam, we observed that the weaker students appeared to be able to write code with markedly more success than they could read it. Examination of a second data set from a different institution failed to confirm the observation, and appropriate statistical analysis failed to find any evidence for the conclusion. We speculate on the reasons for the lack of a firm finding, and consider what we might need to do next in order to more thoroughly explore the possibility of a relationship between the code-reading and code-writing abilities of novice programming students.

9) **Whalley, J. and Lister, R.** (2009). The BRACElet 2009.1 (Wellington) Specification. In Proc. Eleventh Australasian Computing Education Conference (ACE 2009), Wellington, New Zealand. CRPIT, 95. Hamilton, M. and Clear, T., Eds., ACS. 9-18.

http://crpit.com/confpapers/CRPITV95Whalley.pdf

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.

10) **Philpott, A., Clear, T., and Whalley, J.** (2009). *Understanding student performance on an algorithm simulation task: implications for guided learning*. In Proceedings of the 40th ACM Technical Symposium on Computer Science Education (Chattanooga, TN, USA, March 04 - 07, 2009). SIGCSE '09. ACM, New York, NY, 408-412. http://doi.acm.org/10.1145/1508865.1509012

Abstract: This paper extends the work of the BRACElet project by assessing the program comprehension skills of intermediate level students. Student performance on a pathfinder algorithm simulation task is reviewed to assess the students' comprehension levels, as categorised according to the SOLO educational taxonomy. The paper describes the nature of student responses, and the variety of representations provided, illustrating the role of discovery in effective student learning.

11) **Lister, R., Fidge C. and Teague, D.** (2009) *Further evidence of a relationship between explaining, tracing and writing skills in introductory programming*. ITiCSE'09, July 6-9, 2009, Paris, France. http://doi.acm.org/10.1145/1562877.1562930

Abstract: This paper reports on a replication of earlier studies into a possible hierarchy of programming skills. In this study, the students from whom data was collected were at a university that had not provided data for earlier studies. Also, the students were taught the programming language Python, which had not been used in earlier studies. Thus this study serves as a test of whether the findings in the earlier studies were specific to certain institutions, student cohorts, and programming languages. Also, we used a non-parametric approach to the analysis, rather than the linear approach of earlier studies. Our results are consistent with the earlier studies. We found that students who cannot trace code usually cannot explain code, and also that students who tend to perform reasonably well at code writing tasks have also usually acquired the ability to both trace code and explain code.

12) **Venables, A., Tan, G. and Lister, R.** (2009) *A Closer Look at Tracing, Explaining and Code Writing Skills in the Novice Programmer*. The Fifth International Computing Education Research Workshop (ICER 2009), Berkeley, California, August 10-11, 2009. http://doi.acm.org/10.1145/1584322.1584336

Abstract: The way in which novice programmers learn to write code is of considerable interest to computing education researchers. One research approach to understanding how beginners acquire their programming abilities has been to look at student performance in exams. Lopez et al. (2008) analysed student responses to an end-of-first-semester exam. They found two types of questions accounted for 46 per cent of the variance on the code writing portion of the same exam. One of those types of question required students to trace iterative code, while the other type required students to explain what a piece of code did. In this paper, we investigate whether the results by Lopez et al. may be generally indicative of

something about novice programmers, or whether their results are just an artefact of their particular exam. We studied student responses to our own exam and our results are broadly consistent with Lopez et al. However, we did find that some aspects of their model are sensitive to the particular exam questions used. Specifically, we found that student performance on explaining code was hard to characterise, and the strength of the relationship between explaining and code writing is particularly sensitive to the specific questions asked. Additionally, we found Lopez et al.'s use of a Rasch model to be unnecessary, which will make it far easier for others to conduct similar research.

13) **Clear, T., Philpott, A., Robbins, P. & Simon.** (2009/2010, Dec/Jan). 'Report on the eighth BRACElet workshop: BRACElet Technical Report 01/08 AUT University, Auckland'. *Bulletin of Applied Computing and Information Technology* Vol. 7, Issue 1. ISSN 1176-4120. http://www.naccq.ac.nz/bacit/0701/2009Clear_BRACElet_Report.pdf

Introduction: This paper reports on the activities of the Eighth BRACElet workshop held 4 July 2008 concurrent with the NACCQ Conference at AUT University. The BRACElet project is a longitudinal multi-institutional multinational investigation into the code reading, code comprehension and code writing skills of novice programmers.

14) **Lister, R., Clear, T., Simon, Bouvier, D., Carter, P., Eckerdal, A., Jacková, J., Lopez, M., McCartney, R., Robbins, P., Seppälä, O., Thompson, E.** (2010) 'Naturally occurring data as research instrument: analyzing examination responses to study the novice programmer'. *SIGCSE Bulletin*. 41, 4 (Jan 2010), 156-173. http://doi.acm.org/10.1145/1709424.1709460

Abstract: In New Zealand and Australia, the BRACElet project has been investigating students' acquisition of programming skills in introductory programming courses. The project has explored students' skills in basic syntax, tracing code, understanding code, and writing code, seeking to establish the relationships between these skills. This ITiCSE working group report presents the most recent step in the BRACElet project, which includes replication of earlier analysis using a far broader pool of naturally occurring data, refinement of the SOLO taxonomy in code-explaining questions, extension of the taxonomy to code-writing questions, extension of some earlier studies on students' 'doodling' while answering exam questions, and exploration of a further theoretical basis for work that until now has been primarily empirical.

15) **Simon.** (2010) *A note on code-explaining examination questions*. 9th Koli Calling International Conference on Computing Education Research. Finland.

Abstract: The BRACElet project, which explores aspects of the way students learn to program, involves questions to assess the students' skills in tracing, reading, and writing code. In a recent examination based on the BRACElet specification, students' marks were significantly lower on the code-reading questions than on the other two types. A close examination of the students' answers leads to the conclusion that a marking scheme for code-reading questions should be proposed explicitly, and that work is still required to ensure that students fully understand what sort of answer will earn them full marks for such a question.

16) **Tan, G. and Venables, A**. (2010) 'Wearing the assessment "BRACElet"'. *Journal of Information Technology Education*, Volume 9, pages IIP 25-34. http://www.jite.org/documents/Vol9/JITEv9IIPp025-034Tan778.pdf

Executive Summary: There exists a wealth of computing education literature

devoted to interventions designed to overcome novices' difficulties in learning to write computer programs. However, various studies have shown that the majority of students at the end of a semester of instruction are still unable to write a simple computer program, despite the best efforts of their teachers (Lister et al., 2004; McCracken et al., 2001; Soloway, Bonar, & Ehrlich, 1983). In an effort to address this problem, a workshop titled Building Research in Australasian Computing Education (BRACE) was convened in 2004. BRACE brought together academics interested in learning and applying the techniques and methodologies of action research to the problem of poor student code-writing performance. At this workshop, and at those that followed, participants agreed to use end-of-semester assessments to try to pinpoint the key steps and difficulties beginners faced in learning introductory programming. Subsequently the group, which has come to be known as the BRACElet project, has continued to meet twice a year in an effort to better understanding of how students learn and grasp various programming concepts.

Over the past five years, the BRACElet project has fostered a multi-institutional community of practice amongst academics who teach novice computer programming. For participants, the BRACElet project provides the benefits of external moderation, quality assurance, and benchmarking of examinations. At a BRACElet meeting, attendees decide upon a small common set of question types for use within their own local examinations; it is the subsequent analysis and discussion of locally collected data from various institutions that comprises the BRACElet project's iterative series of action research. By examining, across institutions, common difficulties students may have on various question types, such as reading, tracing, and coding, the BRACElet project attempts to build theory on how learners acquire programming knowledge.

In 2008, academics from Victoria University joined the BRACElet group. After modifying our end of semester examination to contain the BRACElet core question types, the examination was run and data was collected across the cohort of 32 students. The analysis treated student performance on code writing questions as the dependent variable and looked at the relationship between code writing questions and non-code writing questions. Overall, our study found that the combination of tracing and explaining questions, more so than each skill independently correlates highly to code writing skills supporting the BRACElet notion of a hierarchical development of skills in learning to program. From Australasian beginnings, the BRACElet project has continued to attract new collaborators to become a large multi-national, multi-institutional research community that has published over 20 papers, with 28 academics as authors, from 20 tertiary institutions across seven countries. By joining the BRACElet collective, we have been nurtured in making informed decisions about our own assessment practices and our local research has made a small contribution to the dialogue about understanding how novices learn to program.

Author statistics, for BRACElet papers written and published during the period of the fellowship (September 2007 – February 2010).

26 different authors

From seven different countries:

- 10 authors from New Zealand
- nine authors from Australia
- three authors from USA
- one author each from Canada, Finland, Slovakia, and Sweden

From 14 different institutions:

From five different Australian institutions:

- two authors from University of Technology, Sydney

- two authors from Monash University
- two authors from Queensland University of Technology
- two authors from Victoria University, Melbourne
- one author from The University of Newcastle

From six different NZ institutions:

- four authors from Auckland University of Technology
- two authors from The University of Auckland
- one author from Massey University
- one author from Tairawhiti Polytechnic
- one author from Manukau Institute of Technology
- one author from Southern Institute of Technology

From three different US institutions:

- University of California, San Diego
- University of Connecticut
- Southern Illinois University

The remaining authors are from the following institutions:

- The University of British Columbia, Canada
- Uppsala University, Sweden
- University of Zilina, Slovakia
- Helsinki University of Technology, Finland

The number of papers (co)authored by each author during the period of the fellowship (September 2007 – February 2010)

9 **Lister**, UTS, Australia
7 **Whalley**, AUT, NZ
6 **Clear**, AUT, NZ
5 **Thompson**, Massey, NZ
5 **Simon**, UCSD, USA
4 **Robbins**, AUT, NZ
3 **Lopez**, Manukau Inst. of Tech., NZ
3 **Simon**, U. of Newcastle, Australia
2 **Luxton-Reilly**, U. Auckland, NZ
2 **Hu,** Tairawhiti Polytechnic, NZ
2 **Carbone**, Monash, Australia
2 **Sheard**, Monash, Australia
2 **Philpott**, AUT, NZ
2 **Tan**, Victoria U., Melbourne, Australia
2 **Venables**, Victoria U., Melbourne, Australia
1 **Bouvier**, Southern Illinois Univ. Edwardsville, USA
1 **Carter**, University of British Columbia, Canada
1 **Denny**, U. Auckland, NZ
1 **Eckerdal**, Uppsala University, Sweden
1 **Edwards**, UTS, Australia
1 **Fidge**, Queensland U. of Technology, Australia
1 **Jacková,** University of Zilina, Slovakia
1 **McCartney**, University of Connecticut, USA
1 **Seppälä**, Helsinki University of Technology, Finland
1 **Sutton**, Southern Institute of Technology, NZ.
1 **Teague**, Queensland U. of Technology, Australia

Appendix C: A technical summary of recent results from the project, for the ICT academic reader

The Common Core

As the project developed, we settled upon studying how students answered three types of questions. Whalley & Lister (2009) formally documented these types of questions, in detail. The three question types are referred to collectively, within the project, as 'the common core'. The three types of questions are as follows:

1) **Basic knowledge and skills:** Some form of tracing question is most commonly used, such as the tracing question in the box below:

Consider the following code fragment:

```
int[] x = {0, 1, 2, 3};
int temp;
int i = 0;
int j = x.length-1;

while (i < j)
{
      temp = x[i];
      x[i] = x[j];
      x[j] = 2*temp;
      i++;
      j--;
}
```

After this code is executed, array "x" contains the values:

a) {3, 2, 2, 0}
b) {0, 1, 2, 3}
c) {3, 2, 1, 0}
d) {0, 2, 4, 6}
e) {6, 4, 2, 0}

The purpose of this type of question is two-fold. First, it establishes that a student understands the programming constructs being tested (eg how a 'while' loop works). Second, it establishes that a student has mastered the concrete skill of tracking variable updates as the student traces through code.

2) **Reading / understanding:** The type of exam question used has been 'Explain in plain English' questions, where students are shown a small piece of code and are asked to explain what the code does. Note that the student is expected to describe what the code does as a whole; NOT provide a line-by-line description of that the code does. The question in the box below has been used in a number of our studies:

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

```
boolean bValid = true;

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

We categorise student responses to 'Explain in plain English' using the SOLO taxonomy (Biggs and Collis 1982). See the table below, which is adapted from Whalley et al. (2006):

Table 2 from Whalley et al. (2006): SOLO Categories

SOLO category	Description
Relational	The student provides a summary of what the code does in terms of the code's purpose. For example, for the code in the box above, a suitable relational response would be "It checks to see if the array is sorted".
Multistructural	The student provides a line by line description for all or most of the code.
Unistructural	The student provides a description for a portion of the code.
Prestructural	The student response demonstrates a lack of knowledge of programming constructs or the answer is unrelated to the question. Also, the question was not attempted.

3) Writing: These are questions where students are required to write code, when given a brief description of what the code should do. For example, a student might be asked "Given an array of integers called iNumbers, write code that checks to see if the array is sorted". While the sophistication of the question may vary from institutions to institution, this style of question is a common style of exam question.

The remainder of this appendix focuses on the project's most recent findings. Since this document is a report on the ALTC Fellowship, the findings presented will only be for the project activities in which the fellows have been directly involved. Descriptions of other work in the project can be found in the publications listed in Appendix B.

Lister, Fidge and Teague (2009)

Table 7 from Lister, Fidge and Teague (2009) is reproduced below. It summarises the relationships found in that study between student performance on writing exam questions and the combination of tracing and explanation questions. In that study, students were given three tracing questions. As indicated in the leftmost column of Table 7, a score of 1 out of 3 on the tracing questions was considered to be a low score, whereas a score of at least 2 out of 3 was considered a high score. The students were also given 4 'Explain in Plain English' questions. As indicated in the uppermost row of Table 7, a score of 1 or 2 out of 4 on these questions was considered to be a low score, whereas a score of 3 or 4 was considered a high score. The lower left cell of Table 7 (ie the cell containing '10%') shows the data for students who scored a low score on both tracing (ie 1 out 3) and explanation (ie 1 or 2 out of 4). That lower left cell shows that only 10 per cent of those students gave a good answer to the writing question. If we move vertically from that lower left cell of Table 7, to the upper left cell, then the score on explanation remains low (ie 1 or 2 out of a total of 4), but the tracing score increases to 2 or 3 out of 3. This cell shows that 46 per cent of these students scored well on the writing task. If we now move horizontally from that upper left cell to the upper right cell, where students did well on both the tracing and explanation questions, then the percentage of students who gave a good answer to the writing question rises to 67 per cent.

	Number of Good Explanations on the Four Explain in Plain English Questions		
Tracing Score	1 or 2		3 or 4
2 or 3	46%	← $\chi 2 = 3.8$ →	67%
	$\chi 2 = 4.0$ ↕		$\chi 2 = 9.4$ ↕
1	10%	← $\chi 2 = 0.5$ →	0%

Table 7 from Lister, Fidge and Teague (2009):
Percentage of good answers to the writing task for
combined scores on tracing and explanation questions (n=137).

Venables, Tan and Lister (2009)

Figure 12 (reproduced below) from the paper by Venables, Tan and Lister (2009) summarises the relationships found between student performance on writing exam questions and the combination of tracing and explanation questions. The figure shows a multiple regression, with the score on the code writing tasks as the dependent variable. The independent variables are the combined scores on nine tracing tasks and two explanation tasks. The line of regression in Figure 12, with R^2 of 0.66, is an excellent outcome, especially with only two explanation tasks available for inclusion in the regression. Furthermore, student scores on two of the separate writing tasks comprising the dependent variable only, showed an R^2 of 0.8, so the R^2 of 0.66 is very high (0.66 / 0.8 = 83 per cent).

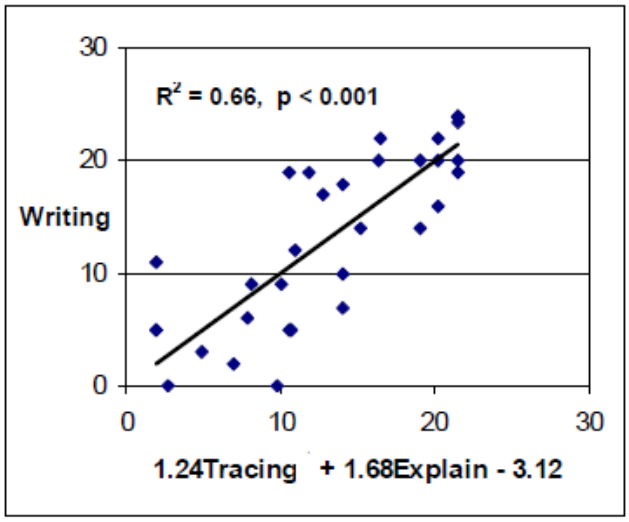


Figure 12 from Venables, Tan and Lister (2009):
A multiple regression, with score on code writing as the dependent variable, and the combination of scores on tracing and explaining as the independent variables.

A visually striking feature of Figure 12 is the absence of data points that are well removed from the line of regression – there are no points toward the upper left or the lower right of Figure 12. One interpretation of this relatively tight distribution around the line of regression is that the two independent variables (ie tracing and explaining) describe most of the factors leading to performance at code writing.

The linear statistical relationship between student score on the nine tracing questions and the combined writing questions is shown in Figure 9. The solid line in the figure is a line of regression through all data points, with an associated $R^2 = 0.50$ (shown in bold).

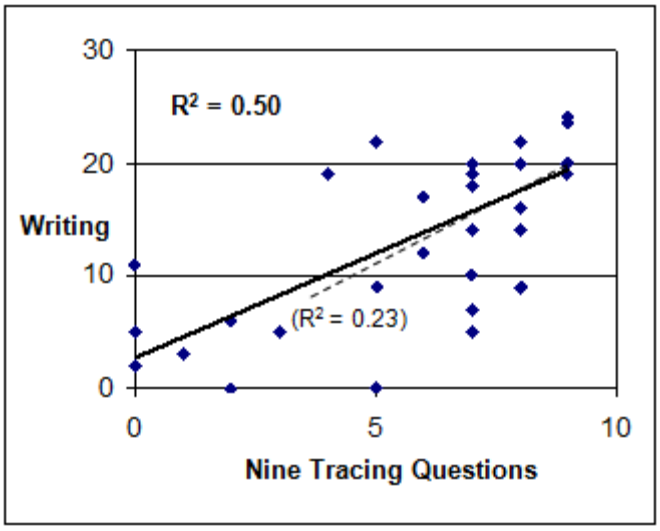


Figure 9 from Venables, Tan and Lister (2009):
A comparison of student scores on all nine tracing questions and the combined writing questions.

The dashed line in Figure 9 is a line of regression for the subset of tracing scores ≥4. That line only has an associated $R^2 = 0.23$. That is, Figure 9 illustrates that those students who score poorly on the tracing questions rarely score well on the code writing tasks, but there is no clear relationship with code writing for students who

scored well on tracing questions. This suggests a causal relationship, where a minimal level of skill at tracing is necessary for code writing, but that minimal skill at tracing is not sufficient by itself to enable code writing. It is the skills required for code explanation that, when combined with tracing skill, form a strong predictor of performance on code writing.

Discussion of the common core results, including pedagogical implications

The above two papers, and our earlier studies in this project upon which these papers build, are built upon the ongoing use of BRACElet type questions at a number of Australasian universities over several years with many hundreds of students participating each semester. From these, a picture is emerging of the early development of the novice programmer. First, the novice acquires the ability to trace code. As the capacity to trace becomes reliable, the ability to explain code develops. When students are reasonably capable of both tracing and explaining, the ability to systematically write code emerges.

While arguing for a hierarchical development of programming skills, we do NOT argue for a strict hierarchy, where the ability to trace iterative code, and explain code, precedes any ability to write code. We merely argue that that some minimal competence at tracing and explaining precedes some minimal competence at systematically writing code. Any novice who cannot trace and/or explain code can only thrash around, making desperate and ill-considered changes to their code − a student behaviour many computing educators have reported observing. Having written a small piece of code, it is important that a student be able to pause, inspect the code, and see that the code actually does what the student intended. Also, when programmers struggle to see a bug by such an inspection, but the output from running the code shows that a bug is present, they may resort to tracing the code.

Until students have acquired minimal competence in tracing and explaining, it may be counterproductive to have them write a great deal of code. We do not advocate that students be forced to demonstrate minimal competence in tracing and explaining before they write any code (indeed, we suspect that such an approach would lead to motivational problems in students). However, we do advocate that educators pay greater attention to tracing and explaining in the very early stages of introductory programming courses, and that educators discuss with their students how these skills are used to avoid a haphazard approach to writing code.

Appendix D: Tips on how to run a similar multi-institutional collaborative project in any discipline, based on the BRACElet experience

1. Getting started

A multi-institutional collaborative project will probably grow from a nucleus of people who have worked together already, perhaps people who have an affiliation via a professional organisation or regular conference. BRACElet commenced in New Zealand, after Dr. Tony Clear heard Raymond Lister's keynote address at the NACCQ 2004 conference. He organised and hosted the first BRACElet workshop at his university. Also, he had strong ties with the nation-wide NACCQ organisation, which was a natural network through which to spread word of that workshop. Later, the NACCQ conference hosted several of the subsequent BRACElet workshops.

A multi-institutional collaborative project probably needs to begin with a clear, well defined goal. Replicating an interesting piece of previously published work, perhaps with minor extensions, is such way to start. The previously published piece of work may be an interesting result derived from work at a single institution, so a multi-institutional replication is an interesting and very publishable initial goal. The first BRACElet workshop was conducted with that sort intention, to replicate a piece of work (Lister et al., 2004), although that work was already multi-institutional.

2. Recruitment

The initial meeting may throw its doors open, and invite anyone with an interest in the topic area. Subsequently, the project organisers may restrict some meetings to people who are known to have done work on the project prior to that meeting – perhaps to people who have collected data. Such a restriction reduces social loafing. However, BRACElet still held most of its meetings as being open to newcomers, and by that policy the project gradually accreted people with valuable skills.

A mix of experience levels is useful. BRACElet has always had a mix of people, some with strong track records in discipline-based education research, and others who were there to learn. BRACElet has always seen its role as two-fold: to pursue a particular research agenda, but also to train people who are new to discipline-based education research. Inevitably, the project has also lost participants, so regularly recruiting and training new participants maintains critical mass.

There have been many people who have come to one or two BRACElet workshops, but who have never truly joined the project. As we see it, you cannot tell who will definitely go on to join the project, so having people who come and go is an inevitable part of the recruitment process. Besides, those people who come and go can be seen as part of the dissemination process. There may be some concern that people who come to one or two workshops may 'steal' the work of the project, but BRACElet has never experienced that problem, which may be due to the emphasis on shared activities, shared artefacts and the publication protocol (discussed below).

3. Shared data / shared artefacts / common core

The participants must bond, and they do so through a shared experience. Without shared data, and shared artefacts, meetings are a gathering of people with a common interest but different projects. Such meetings are inevitably competitive.

In the first one or two action research cycles, it may be good if people worked in a tightly coupled fashion. An experiment kit (see Fincher et al., 2006) ensures tight coupling. Another useful idea is the common core – a common subset of activities, allowing participants to pursue their own interests as long as they complete the core

activity.

4. Being grounded in data / evidence

Academics (folk pedagogues) LOVE to gossip about their teaching. But when that is all they do, and two or more academics disagree, there is no resolving the disagreement without evidence. A BRACElet-style project must be grounded in data, otherwise it's a rabble. Disagreements should be seen as a research opportunity.

5. Agreed rules of engagement

To avoid conflict, BRACElet uses an explicit set of "Rules of Engagement". The philosophy behind the rules is that having ideas is quick and easy − the hard, time-consuming work is in less glamorous activities such as getting ethics clearance, drafting the actual final detailed data collection instruments, collecting the data, and doing the analysis. The rules are organised around the phases of action research:

Plan: Most of the workshops conduct sessions where project participants discuss the work done since the previous meeting, ideas for where the project will go next. In these sessions, ideas are given away. That is, if person A offers up an idea that is eventually used by person B then person A is not by default a co-author.

Act: Inevitably, not all project participants are interested in all project activities. Instead, project participants form groups that will conduct their own studies around specific, common instantiations of the components, for which all those "nucleus" team participants collect data. Such team nuclei may also, at their discretion, invite specific people from the broader framework to join their team (as coauthors) to contribute to data analysis and writing.

6. Publication protocol

It is important to develop and agree on a protocol for who can expect to be an author on a paper. Authoring is the most likely source of dispute. See Appendix E for a suggested protocol.

7. Protocols for data collection, management and sharing

To enable meaningful data analysis and comparisons to be made between data from different sources and institutions, good data management − common spreadsheet formats etc. – should be established as part of the planning stage.

8. Planned obsolescence

By about the third action research cycle, people who were new to this style of work will have become experienced collaborators who will have acquired, quite rightly, their own research interests and agenda. Do not frustrate that. Perhaps the existing project should end, so that new projects, with new leadership, can emerge. Or perhaps the project could have a 'sabbatical', where the project is suspended for a period of time, so people can explore their own directions. At least put in place a mechanism for a routine, amicable change to the leadership.

9. Further essential reading on running these projects

Whalley, J., Clear, T. & Lister, R. (2007), *The Many Ways of the BRACElet Project*. Bulletin of Applied Computing and Information Technology Vol. 5, Issue 1. ISSN 1176-4120.
http://www.naccq.co.nz/bacit/0501/2007Whalley_BRACELET_Ways.htm

Read section 6, which discusses the following issues:

It Works Better When There Are Agreed Rules of Engagement

It Works Better When There Is a Publication Protocol

It Works Better If Ethical Issues Are Addressed

It Works Better When Beliefs about the Teaching/Research Nexus Are Shared

It Works Better When There Are Protocols for Data Collection and Management

It Works Better When Participants Can Meet

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*. In Proceedings of the First International Workshop on Computing Education Research (Seattle, WA, USA, October 01 - 02, 2005). ICER '05. ACM, New York, NY, 111-121. http://doi.acm.org/10.1145/1089786.1089797

Despite being written for a computer science audience, much of the material in this paper is accessible to a broader audience. Section 3 is particularly relevant.

Appendix E: A suggested authoring / publication protocol

Authorship has the most potential to cause conflict in a large collaboration, especially a multi-institutional collaboration. This appendix provides a draft set of authorship guidelines for such collaboration. This draft was originally based upon the British Sociological Association's Authorship Guidelines for Academic Papers: http://www.britsoc.co.uk/Library/authorship_01.pdf accessed April 2010

It was first adapted by Sally Fincher, who kindly supplied it to us.

Agree early, Agree often: Authorship should be discussed between researchers at an early stage in any project and renegotiated as necessary. Where possible, there should be agreement on which papers will be written jointly (and who will first author each paper), and which will be single authored, with an agreed acknowledgement given to contributors. Many disputes can be avoided by a clear common understanding of standards for authorship (especially in multi-disciplinary groups). A record should be made of these discussions. Early drafts of papers should include authorship and other credits to help resolve any future disputes.

Legitimate authorship: Everyone who is listed as an author should have made a substantial direct academic contribution (i.e. intellectual responsibility and substantive work) to at least **two** of the four main components of a typical paper:

1. **Conception or design**. Note that this implies a significant amount of work; it is not just the contribution of an idea.
2. **Data collection and processing**. Note that the BRACElet project moved to a policy of having as an author anyone who had collected data from their students, as part of an exam. No other contribution had to be made. This applied to the first paper that used the data. This change was made because BRACElet needed to encourage more people to collect data.
3. **Analysis and interpretation of data**
4. **Writing substantial sections of the paper** (e.g. synthesising findings in a literature review or a findings/results section). This does NOT include offering comments on a draft.

Everyone who is listed as an author should have critically reviewed successive drafts of the paper and should approve the final version.

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

Non-legitimate authorship claims: The following, by themselves, do not justify authorship ... contribution of ideas, paying for the work, reviewing the work, editing the work, acquiring the funding, collecting the data, supervising the research group.

Order of authors: The person who has made the major contribution to the paper and/or taken the lead in writing is entitled to be the first author.

1. Those that have made a major contribution to analysis or writing (i.e. more than commenting in detail on successive drafts) are entitled to follow the first author immediately; where there is a clear difference in the size of these contributions, this should be reflected in the order of these authors.

2. All others who fulfill the criteria for authorship should complete the list in alphabetical order of their surnames.

3. If all the authors feel that they have contributed equally to the paper, this can be indicated in a footnote.

Acknowledgements: All those who make a substantial contribution to a paper without fulfilling the criteria for authorship should be acknowledged, usually in an acknowledgement section specifying their contributions. These might include interviewers, computing staff, clerical staff, statistical advisers, colleagues who have reviewed the paper, students who have undertaken some sessional work, the supervisor of a research team and someone who has provided assistance in obtaining funding.

Appendix F: 'Explain in plain English' questions: examples and advice

Background

In 'Explain in plain English' questions, students are shown a small piece of code and they are asked to explain what the code does. Note that the student is expected to describe what the code does as a whole; NOT provide a line-by-line description of that the code does (although this distinction has not always been made clear to students).

The SOLO Taxonomy

'Explain in plain English' questions are typically analysed via a series of categories based on the SOLO taxonomy (Biggs and Collis 1982). See Table 2 below, which is adapted from the Whalley et al. (2006) paper.

Table 2 from Whalley et al. (2006): SOLO Categories

SOLO category	Description
Relational	The student provides a summary of what the code does in terms of the code's purpose. For example, for the code in the box above, a suitable relational response would be "*It checks to see if the array is sorted*".
Multistructural	The student provides a line by line description for all or most of the code.
Unistructural	The student provides a provides a description for a portion of the code
Prestructural	The student response demonstrates a lack of knowledge of programming constructs or the answer is unrelated to the question. Also, the question was not attempted.

Note: The "extended abstract" SOLO category is not included in this table as it is not appropriate to an 'explain in plain English' question.

Previously Published 'Explain in Plain English' Questions ...

From Whalley et al. (2006)

This was the first BRACElet "Explain in Plain English" question and has been used in a number of subsequent versions.

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

```
boolean bValid = true;

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

A SOLO interpretation of student responses to the above question is as follows:

- **Unistructural Responses:** This type of description has a narrow focus with an emphasis on individual statements or part of a statement such as "number at index is less than or equal to the number at index plus one". This is a focus on the condition of the "if" statement and ignores the iteration through the array. An example of a unistructural comment, taken from the data collected, is "*If the index number is higher than the index number plus one then the code will be invalid*".
- **Multistructural Responses:** For the multistructural category, the description describes two or more statements but without showing any relationship between them other than the sequence in which they appear. Two examples of multistructural comments are:
  - "*Compare every element in the array with the one next to it. Return false if it is bigger than the one next to it*."
  - "*From the first element in the array to the second last element. Test to see if they are in order from the smallest to the largest. If not, return a 'false' to bValid otherwise bValid remains 'true*"'.

  The extreme multistructural response is when the student describes each line of the code. For example:
  - *"bValid is a Boolean. i value is 1. Perform the loop until i equals to the number of array which is deducted -1. If the value of i array is greater than the one of i+1 array, set bValid to "False"".*
- **Relational Responses:** In the relational category, the description draws together all of the code showing an increasing depth of understanding of the relationships between individual statements to achieve an overall result. These descriptions of the code segment become more abstract as they rely less on the actual operations specified in the code. Three examples of relational comments are:
  - *"Test an array of integers if their values are listed from smallest to largest return true else return false."*
  - *"This piece of code is used to find out if the values in an array are in ascending order"*

- *"To recognise all the elements of the array in increasing order. If two elements which are neighbours are the same number the program still thinks they are in increasing order"*

From Clear et al. (2008)

This paper described three explain in plain English questions, which began with the common preamble shown in the box below.

> For each of these sections of code, explain in plain English what it does.
>
> Note that more marks will be gained by correctly explaining the purpose of the code than by giving a description of what each line does.
>
> Variable names are deliberately not very meaningful so you will have to work out what the code does.

The three questions were as follows:

```
public double method10A(double[] aNumbers)
{
 double num = 0;

 for(int iLoop = 0; iLoop < aNumbers.length; iLoop++)
 {
  num += aNumbers[iLoop];
 }

 return num;
}
```

```
public void method10B(int iNum)
{
      for(int iX = 0; iX < iNum; iX++)
      {
            for(int iY = 0; iY < iNum; iY++)
            {
                  System.out.print("*");
            }

            System.out.println();
      }
}
```

```
public boolean method10C(int[] aiNum, int iValue)
{
     int iX = 0;
     int iY = aiNum.length - 1;
     int iZ,
     iTemp;
     boolean bSwitch = false;

     while (!bSwitch && (iX <= iY))
     {
          iZ = (iX + iY) /2;
          iTemp = aiNum[iZ];

          if ( iValue == iTemp )
          {
               bSwitch = true;
          }
          else if ( iValue < iTemp )
          {
               iY = iZ - 1;
          }
          else
          {
               iX = iZ + 1;
          }
     }

     return bSwitch;
}
```

From Lister, Fidge and Teague (2009)

This paper described four explain in plain English questions, written in Python. All four of these questions began with the following instruction:

> *Explain, in plain English, the purpose of the following Python function. (Do not say how the code works. Instead say what the function would be used for.)*

We recommend two of the four functions as particularly good questions:

-
```
def enigma(nums): # nums is assumed to be a list of numbers
  for index in range(len(nums) - 1):
    if nums[index] > nums[index + 1]:
      return False
  return True
```

  A suitable relational answer would be *It checks to see if the list is sorted*. This is the same question (rewritten into Python) as used in some earlier studies (Whalley et al., 2006; Sheard et al., 2008).

- ```
  def puzzle(nums): # nums is assumed to be a list of numbers
  answer = []
  for num in nums:
    if num >= 0:
     answer = answer + [num]
  return answer
  ```

  A suitable relational answer would be *It returns the given list of numbers with all the negative numbers removed*.

From Simon (2010)

This paper contained the following question, written in VisualBasic:

Explain what the following code does. You are not being asked to explain each line of the code; you are being asked to explain its overall purpose.

```
strOne = strTitle(0)
For i = 1 To strTitle.Length - 1
  If strTitle(i).Length > strOne.Length Then
   strOne = strTitle(i)
End If
Next
```

A suitable relational answer is: *It finds the longest title in the array of titles.*


```

From Sheard et al. (2008)

This paper described two new 'explain in plain English' questions:

Assume that a, b, c are declared as integers and have been initialized. Explain the purpose of the following segment of code.

```
a = b;
b = c;
c = a;
```

A suitable relational answer to the above question would be *It swaps the values in variables b and c.*

Suggest a name for method10 below that reflects its purpose.

```
public float method10(int[] aiNumbers)
{
 int iSum = 0;

 for (int iLoop=0; iLoop < aiNumbers.length; iLoop++)
 {
  iSum += aiNumbers[iLoop];
 }

 return iSum / aiNumbers.length;
}
```

A suitable relational answer to the above question would be *This method returns the average of the numbers in the array*.

From Venables, Tan, and Lister (2009)

This paper contained the following question, written in Java:

```
if ( a < b)
   if ( b < c)
        System.out.println (c);
   else
        System.out.println (b);
else if ( a < c)
        System.out.println (c);
else
    System.out.println (a);
```

A suitable relational answer is *It prints out the largest of the three values*.

Advice on Writing Good 'Explain in Plain English' Questions

1) A good explain question has a purpose that an expert programmer can easily deduce.
2) A good explain question has a purpose that a programmer might reasonably want to do.
3) A good explain question has a purpose that can be described in English in less length than the code itself.
4) A good explain question does not cognitively overload the reader. For example, the code does not have a lot of variables.
5) In a loop that involves an array, good explain questions often have each loop iteration access/modify more than one location of the array. For example, the classic BRACElet explain question (*checks to see if the array is sorted*) has code that looks at positions [i] and [i-1] at each loop iteration.
6) Good explain questions on loops often have the property that earlier passes through the loop impact on later passes through the loop. For example …
    a) Find the maximum/minimum. To understand that sort of code, you have to understand that the there is a variable that is holding the best value you've found thus far, in earlier passes through the loop.
    b) A "for" loop that scans across an entire array, looking for a specific value. An answer like *the code finds the position of the target value in the array* is not quite right. A better answer is something like *the code finds the highest position of the target value in the array*.

    As a counter example to this rule (i.e. a weak explain question), consider a loop that increments all values in an array. A student can read that loop declaratively, and doesn't need to understand how loops execute.

Unpublished 'Explain in Plain English' Questions

At the time this report was written, the following questions had not been used in any published study.

The first of the following questions is intended for use very early in an introductory programming course:

Qx. The purpose of the following code is to swap the values in variables a and b.

```
c = a;
a = b;
b = c;
```

<Then either …>
The following three lines reorder the above three lines. In one sentence, describe the purpose of these three lines of reordered code.

```
a = b;
b = c;
c = a;
```

<or … >
In one sentence, describe the purpose of the following three lines of code.

```
j = i;
i = k;
k = j;
```

Qx. Consider the following method called "mystery":

```
public int mystery(int numbers[])
{
 int maximum = 1;
 int count = 1;
 for ( int i=1 ; i<numbers.length ; ++i )
 {
  if ( numbers[i-1] == numbers[i] )
   ++count;
  else
   count = 1;
  if ( maximum < count ) maximum = count;
 }
 return (maximum);
}
```

Question continues on next page …

... question continues from previous page ...

Which of the following MOST ACCURATELY describes what "mystery" returns?

(a) the maximum value in the array
(b) the number of times the maximum value occurs in the array
(c) the number of times any two consecutive elements of the array are equal
(d) the number of times any two consecutive elements of the array both contain the maximum value
(e) the length of the longest consecutive set of numbers in the array that are equal
(f) the length of the longest consecutive set of numbers in the array that are the maximum value

Combining Tracing and Explain in Plain English Questions

This style of question establishes that a student understands all the constructs in a piece of code (via tracing) before asking the student to explain the code.

Qx. Consider the following code, where variables a, b and c are of type int:

```
if ( a < b) {
 if ( b < c)
     System.out.println (c);
 else
     System.out.println (b);
}
else if ( a < c)
   System.out.println (c);
  else
   System.out.println (a);
```

Which one of the following values for the variables will cause the value in variable b to be outputted?

(a) a = 1 b=2 c = 3
(b) a = 1 b=3 c = 2
(c) a = 2 b=1 c = 3
(d) a = 2 b=3 c = 1

In the box below, in a single sentence, describe what the value outputted represents, for any set of values stored in variables a, b and c.

Sample answer: *It outputs the largest value stored in a, b or c.*

Qx. The following code reads positive numbers from input until the person using the program enters a negative number. The program performs a computation on the positive numbers as those numbers are entered, and when the terminating negative number is entered the program outputs the result of that computation.

```
Scanner scan = new Scanner(System.in);

int x = scan.nextInt();
int y = scan.nextInt();
int count = 1;
int maximum = 0;

while ( y >= 0 )
{
 if ( x == y )
  ++count;
 else
  count = 1;

 if ( maximum < count ) maximum = count;

 x = y;
 y = scan.nextInt();
}

System.out.println("maximum = " + maximum);
```

Which one of the following sequences of input values will cause the code to output "maximum = 2"

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

In the box below, in a single sentence, describe what the "maximum" value outputted represents, for any input sequence of two or more positive numbers:

Sample answer: *The length of the longest sequence of identical numbers.*

There is no reason why code explanation need be restricted to describing complete methods and other pieces of code, as in the above questions. Explanation questions can also target a student's understanding of the role of a particular variable in a piece of code, such as in the following question on the next page …

Qx. With regard to the code in the previous question, which of the following MOST ACCURATELY describes the role played by the variable "x"?

(a) *Stepper:* It steps through a succession of values that can be predicted as soon as the succession starts.
(b) *Most-recent holder:* It holds the latest value encountered in going through a succession of values.
(c) *One-way flag:* It is a two-valued data entity that cannot get its initial value once its value has been changed.
(d) *Most-wanted holder:* It holds the best value encountered so far in going through a succession of values.
(e) *Gatherer:* It accumulates the effect of individual values in going through a succession of values.
(f) *Follower:* It gets its values by following another variable.
(g) *Temporary:* It holds some value for a very short time only.

For further explanation of the above question, see the description of "roles of variables" at …
http://cs.joensuu.fi/~saja/var_roles/

The following is version 1 of a question; version 2 follows immediately …

Qx. Consider the following method called "puzzle", which takes as a parameter an array of integers "x":

```
public void puzzle( int x[] )
{
 int i, j, k;

  i = 0;
  j = x.length - 1;

  while ( i < j )
  {
      k = x[i];
     x[i] = x[j];
     x[j] = k;

     ++i;
     --j;
  }
}
```

Which of the following MOST ACCURATELY describes what "puzzle" does?

(a) It swaps the first and last elements of the array.
(b) It swaps the first half of the array and the second half of the array.
(c) It copies the first half of the array to the second half.
(d) It copies the second half of the array to the first half.
(e) It reverses the order of the elements in the array.

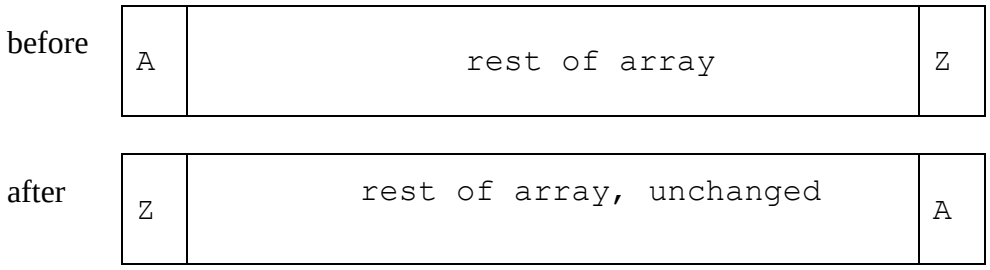
The following is version 2 of the previous question. This version has diagrams added, with the intention of making it more appropriate for students who read English as a second language …

Qx. Consider the following method called "puzzle", which takes as a parameter an array of integers "x":

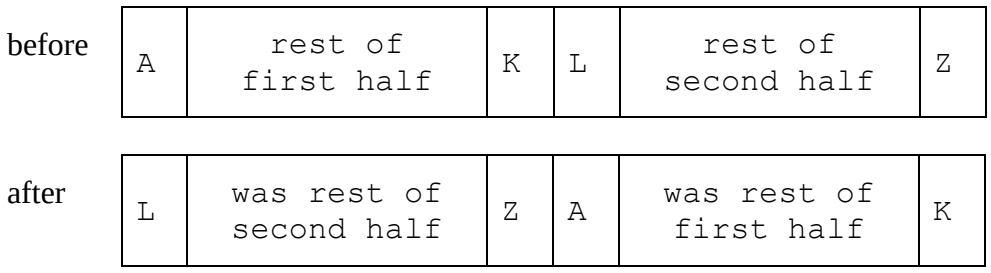
```
< SAME CODE AS IN VERSION 1>
```

Which of the following MOST ACCURATELY describes what "puzzle" does?

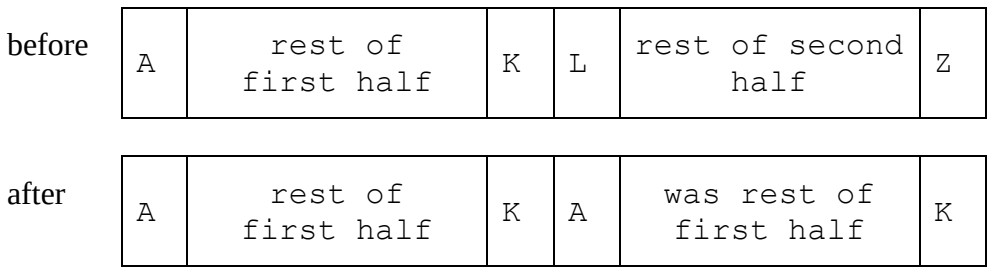
(a) It swaps the first and last elements of the array.



(b) It swaps the first half of the array and the second half of the array.



(c) It copies the first half of the array to the second half.



Options d and e are on the next page …

(d) It copies the second half of the array to the first half.

before	A	rest of first half	K	L	rest of second half	Z
after	L	was rest of second half	Z	L	rest of second half	Z

(e) It reverses the order of the elements in the array.

before	A	rest of array	Z
after	Z	rest of array, reversed	A

Qx. With regard to the method "puzzle" in the previous question, which of the following MOST ACCURATELY describes the role played by the variable "k"?

(a) *Most-wanted holder:* It holds the best value encountered so far in going through a succession of values.
(b) *Gatherer:* It accumulates the effect of individual values in going through a succession of values.
(c) *Follower:* It gets its values by following another variable.
(d) *Temporary:* It holds some value for a very short time only.

Qx. With regard to the method "puzzle" in the earlier question, which of the following MOST ACCURATELY describes the role of the variable "i"?

(a) *Stepper:* It steps through a succession of values that can be predicted as soon as the succession starts.
(b) *Most-recent holder:* It holds the latest value encountered in going through a succession of values.
(c) *One-way flag:* It is a two-valued data entity that cannot get its initial value once its value has been changed.
(d) *Most-wanted holder:* It holds the best value encountered so far in going through a succession of values.

The following is version 1 of a question; version 2 follows immediately …

Qx. Consider the following method called "mystery", which searches the array:

```
public int mystery(int data[], int x )
{
  int index;
  int found = -1;

  for ( index=0; index<data.length; index++ )
 {
  if( data[index] == x )
     found = index;
 }

  return found;
}
```

In the box below, in one or two sentences, describe what method "mystery" does. Do NOT give a line by line description of the code. Instead, provide a summary of what the method does.

Possible marking schemes for the above question:

If worth a total of one mark: Answer describes in some way that "mystery" searches the array for the value "x".

If worth a total of two marks: First mark as above. The second mark for noting that what is returned is the LAST location in the array that contains "x".

If worth a total of three marks: First mark as above. The second mark for noting that it is a position that is returned. A student's answer may be incomplete by talking about "the position", or the answer may even erroneously claim that it is the first position, or all positions that is returned. The third mark for noting that what is returned is the LAST location in the array that contains "x".

The following is version 2 of the previous question; a multiple choice version ...

Qx. Consider the following method called "mystery", which searches the array:

```
< SAME CODE AS IN VERSION 1>
```

Which of the following MOST ACCURATELY describes what "mystery" does?

(a) It returns true if the value in "x" is found in the array data and false if it is not found.
(b) It returns the position of the value "x" if that value is found in the array data and returns nothing if it is not found.
(c) It returns the first position of the value in "x" if that value is found in the array.
(d) It returns the last position of the value in "x" if that value is found in the array.
(e) It returns all the positions of the value in "x" if that value is found in the array.

Qx. With regard to the code in the previous question, which of the following MOST ACCURATELY describes the role played by the variable "found"?

(a) *Stepper:* It steps through a succession of values that can be predicted as soon as the succession starts.
(b) *Most-recent holder:* It holds the latest value encountered in going through a succession of values.
(c) *One-way flag:* It is a two-valued data entity that cannot get its initial value once its value has been changed.
(d) *Most-wanted holder:* It holds the best value encountered so far in going through a succession of values.
(e) *Gatherer:* It accumulates the effect of individual values in going through a succession of values.
(f) *Follower:* It gets its values by following another variable.
(g) *Temporary:* It holds some value for a very short time only.

Qx. Consider the following code, which performs operations on an array of integers "x". Prior to the code below being executed, the array "x" was declared and initialized.

```
int i, temp;

temp = x[0];

for ( i=0 ; i<x.length-1 ; ++i )
  x[i] = x[i+1];

x[x.length-1] = temp;
```

In the box below, in one or two sentences, describe what the code does to the array "x". Do NOT give a line by line description of the code. Instead, provide a summary of what the code does.

Sample answer: it moves all elements in the array one place to the left, with the leftmost element moving around to the rightmost position.

... if the name of the variable "temp" was changed to something more obscure, such as "p", a follow up question could be "which of the following MOST ACCURATELY describes the role played by the variable "p"? (The answer is "It temporarily holds the value from another variable.")

Appendix G: Emerging ideas for new types of questions, intended for ongoing work in the BRACelet project.

Many of these new styles of question are being piloted in participating institutions.

Higher Order Multiple Choice Question

Below is the incomplete code for a method "min" which is intended to return the smallest value in the array "x". The code is supposed to work by scanning across the array, using the variable "minsofar" to keep track of the smallest value seen thus far.

Each box contains two lines of code, labeled (a) and (b). For each box, draw a circle around the appropriate line of code so that the method will correctly return the smallest value in the array.

```
public int min( int x[] )
{
 int minsofar =
```

Either (a) 0
or (b) x[0]

```
 for ( int i=1 ; i<x.length ; ++i )
 {
  if ( x[i] <
```

Either (a) minsofar
or (b) x[minsofar]

```
     minsofar =
```

Either (a) i
or (b) x[i]

```
 }
 return
```

Either (a) minsofar
or (b) x[minsofar]

```
}
```

An attractive feature about this style of question is that neither of the options in each box is wrong, when considered in isolation from the other boxes. It is only when a choice has been made in one box that choices become right or wrong in other boxes. Thus there is a relational level of thinking involved in getting this question right.

How many loops/arrays does this problem need?

The following sample questions are separated by asterisks …

Suppose you want to write a program that will read in numbers provided by a user, and output the average of all those numbers. Your program will:

(a) not require either an array or a loop.
(b) require an array but not require a loop.
(c) require a loop but not require an array.
(d) require both a loop and an array.

***

Suppose you want to write another program. This program will read in numbers provided by a user, and output the smallest of all those numbers. Your program will:

(a) not require either an array or a loop.
(b) require an array but not require a loop.
(c) require a loop but not require an array.
(d) require both a loop and an array.

***

Suppose you want to modify a program, which already contains an array of numbers. The code you will add is supposed to look through that array to find whether some value N is stored in the array. The code will first look at the leftmost element of the array and then move right until it either finds N or looks at every location in the array. The code that you add will:

(a) not require either a second array or a loop.
(b) require a second array but not require a loop.
(c) require a loop but not require a second array.
(d) require both a second array and a loop.

***

Either …

Suppose you are asked to write a program to display some simple statistics for a set of positive integers supplied by a person. The person may provide as many positive integers as he/she likes. The person indicates that he/she has finished entering the positive integers by entering a negative integer. The program then outputs: (1) the largest positive number supplied by the person, (2) the smallest positive number supplied by the person, and (3) the average of all the positive number supplied by the person. The minimum number of loops required by your program is:

(a) Zero.
(b) One.
(c) Two.
(d) Three.
(e) Four.

… Or …

Suppose you are asked to write a program to display some simple statistics for a set of positive integers supplied by a person. The person may provide as many positive integers as he/she likes. The person indicates that he/she has finished entering the positive integers by entering a negative integer. The program then outputs: (1) the average of all the positive integers and (2) the number of positive integers that exceeded the average. Such a program would …

(a) Not require an array or a loop.
(b) Require an array but not a loop.
(c) Not require an array but would require a loop.
(d) Require an array and one loop.
(e) Require an array and at least two loops.

Suppose you are asked to write the following program … The person using the program may provide as input as many positive integers as he/she likes. The person indicates that he/she has finished entering the positive integers by entering a negative integer. The program then outputs all the positive numbers inputted, but in the reverse order from which the numbers were inputted. Such a program would …

(a) Not require an array or a loop.
(b) Require an array but not a loop.
(c) Not require an array but would require a loop.
(d) Require an array and one loop.
(e) Require an array and at least two loops.

***

Suppose you are required to write a function that will take three equal sized integer arrays and an integer variable set to the size of the arrays, and returns TRUE if all three arrays contain, in identical order, the same integers. You may assume that all elements in the equal sized arrays have been initialized. Assuming that using recursion is not an option, the minimum number of loops (and if statements where specified) required to produce a working function is:

a) No loops or if statements
b) No loops but multiple if statements
c) One loop
d) Two loops, one loop after the other
e) Two loops, one nested inside the other
f) Three loops

Skeleton Code with Multiple Completions

When the missing code is added, the following code prints out patterns of asterisks:

```
int n = 5;

for (int row=1; row<=n; row++)
{
 for (int column=1; column<=n; column++)
 {
  if ( *** missing code goes here *** )
    System.out.print ("* ");
  else
    System.out.print (" ");
 }

 System.out.println();
}
```

For each of the following patterns of asterisks, provide the appropriate missing code.

Reminder: The logical or operator is "||".
The logical and operator is "&&".
The modulus operator is "%". For example, 5 % 2 = 1 and 6 % 2 = 0

What is the output to the screen by the following code?

a) * missing code is:

```
* *
* * *
* * * *
* * * * *
```

Sample answer: `row > column`

b) * * * * * missing code is:

```
*   *
*   *
*   *
* * * * *
```

Sample answer:

```
row ==1 || row==n || column==1 || column==n
```

c) * * * missing code is:

```
* * *
* * *
* * *
* * *
```

Sample answer: `column % 2 ==1`

Reversing Code

In this type of question, students write code to undo what a given piece of code does. Here is an example of such a question.

The purpose of the following code is to move all elements of the array x one place to the left, with the leftmost element being moved to the rightmost position[†]:

```
int i, temp;

temp = x[0];

for ( i=0 ; i<x.length-1 ; ++i )
   x[i] = x[i+1];

x[x.length-1] = temp;
```

In the box below, write code that undoes the effect of the above code. That is, write code to move all elements of the array x one place to the right, with the rightmost element being moved to the leftmost position[‡].

[†] Alternately, the description of this code could be omitted, by using a sentence like "*Here is a piece of code*".

[‡] Likewise, the description of this code could be omitted, simply by omitting this sentence.

Combining Code

In this type of question, the student is given two pieces of code, C1 and C2. The student is then asked to write a piece of code, C3, which is shorter than the combined lengths of C1 and C2, but achieves the same effect as executing C1 followed by C2. Here is an example of such a question …

The purpose of the following code is swap the values in variables a and b[†].

```
temp = a;
a = b;
b = temp;
```

The purpose of the following code is swap the values in variables b and c[‡].

```
temp = b;
b = c;
c = temp;
```

In the box below, write a piece of code that is shorter than 6 lines (i.e. the combined length of the above two pieces of code) but which achieves the same effect as executing the first set of 3 lines, followed by the second set of 3 lines.

[†] Alternately, the description of this code could be omitted, by using a sentence like "*Here is a piece of code*".

[‡] Likewise, the description of this code could be omitted, by using a sentence like "*Here is another piece of code*".

AUSTRALIAN LEARNING & TEACHING COUNCIL

Promoting excellence in higher education

PO Box 2375 Strawberry Hills NSW 2012 Australia
Telephone 02 8667 8500 Facsimile 02 8667 8515
www.altc.edu.au
ABN 30 109 826 628