Journal of Information Technology Education: Innovations in Practice **Volume 9, 2010**

Wearing the Assessment 'BRACElet'

Grace Tan and Anne Venables
Victoria University, Melbourne, Victoria, Australia

Grace.Tan@vu.edu.au; Anne.Venables@vu.edu.au

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

Editor: John Beachboard

authors, from 20 tertiary institutions across 7 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.

Keywords: Assessment, computing education, introductory computer programming, novice programmers.

Introduction

There has been much written about the difficulties novices experience when learning to write programs and the various strategies to overcome these problems (Dunican, 2002; Miliszewska & Tan, 2007). Despite much reporting upon the plethora of differing teaching approaches, what is not often recognized in the literature is that many students after a semester of instruction are still unable to complete a simple program-writing task (Clear et al., 2008). In their early study, Soloway et al. (1983) reported that more than three-fifths of computer programming students failed when asked to write a simple program to calculate the average of a set of numbers. In a broader and more ambitious study in 2001, several computing student populations in different countries were tested on a common set of program-writing problems (McCracken et al., 2001). This study, along with its follow-up of 615 students across 7 countries in 12 institutions, showed that the majority of computing students were still unable at the end of semester to write a simple computer program, despite the best efforts of their teachers (Lister et al., 2004).

To begin to address this problem, a workshop titled Building Research in Australasian Computing Education (BRACE) was organised to run following the 2004 Australian Computing Education Conference. Funded by a SIGSCE award, the BRACE workshop brought together Australian and New Zealand computing academics interested in learning and applying the techniques and methodologies of computer science education action research to the problem of poor student code-writing performance. A brainstorming exercise was conducted to find possible effective approaches to improve on the current situation. Workshop attendees agreed to participate in a common research activity, and, at a follow-up meeting in 2005, participants organised to gather assessment data in their own classrooms. The goal was to use end-of-semester assessments to try to pinpoint the key steps and difficulties beginners faced in learning introductory programming. Each participant then committed to the addition of an agreed small common set of question types for use within their local examinations, thus repeating an earlier study by Lister et al. (2004) which investigated the relationship between novice programmers' ability to read code with code-writing performance. The subsequent analysis and discussion of the collected data of these commitments began an iterative series of action research, now known as the BRACElet project.

The BRACElet project community meets biannually at workshops which are timed to coincide with key computing education research conferences. The timing enhances the opportunities for interaction, discussion, and exchange between BRACElet participants and the research community as a whole. Each year there is a continual refinement of the action research approach to see a common assessment framework being adopted across all participating institutions. The aims of the BRACElet project have widened from researchers trying to understand novice difficulties in program-writing to an attempt in building theory on how learners acquire programming knowledge. This is achieved through annual incremental changes to the agreed set of common assessment tasks and their subsequent analyses. Additionally, for participants the BRACElet project provides the benefits of external moderation, quality assurance, and benchmarking of examinations

From Australasian beginnings, BRACElet has continued to attract new collaborators to become a large multi-national, multi-institutional research community. BRACElet has published over 20 papers, with 28 academics as authors, from 20 tertiary institutions across 7 countries. In 2008, academics from Victoria University joined the BRACElet group; this paper reports upon this collaborative experience and describes how it has informed local teaching practice. The following section expands upon the BRACElet project in detail, followed by a description of our local experience before commenting on both the local and the broader impacts of the BRACElet project.

The BRACElet Project

The BRACElet project is primarily concerned with the discovery and understanding of the developmental stages and cognitive processes which novice programmers take when learning to code. It aims to gain insight into a student's journey from novice to expert computer programmer through the analysis of a small, common set of assessment tasks conducted across various institutions. The BRACElet approach is based on the belief that valid, reliable assessment instruments aid academics in understanding how students learn. By using successive end-of-semester exams to help to improve assessment tools, it is possible to build learning theory from collected data, particularly when patterns are seen in the data from multiple institutions. Through an analysis of student performance on different types of test questions, it is hoped that results can offer insight into the generality of underlying relationships and possibly point to better teaching strategies to overcome student learning difficulties. These patterns of understanding seen across diverse locales are more indicative of genuine underlying relationships between various programming skill sets. Thus, one of the first tasks at each BRACElet meeting is to communally decide and commit to an examination framework for participants to use in their home institutions.

The composition of the examination framework is based upon a recognized set of essential programming skills. It has been well reported that students should be able to explain code, trace programs, and write code (Lister et al., 2004; McCracken et al., 2001). More importantly, it appears that these three components form a hierarchy of skills (Lopez, Whalley, Robbins, & Lister, 2008; Philpott, Robbins, & Whalley, 2007). At the bottom of the hierarchy is the knowledge and understanding of elementary programming constructs including basic definitions and data types, selections, and iterations. In the intermediate level is the ability to accurately read and trace code. At the top of the hierarchy is the ability to write non-trivial and correct code using programming constructs (Lopez et al., 2008; Sheard et al., 2008; Venables, Tan, & Lister, 2009). With this hierarchy in mind, each iteration of the BRACElet project specifies that examinations should comprise at least one instance of each of the three components detailed below:

Explaining: Several "explain in plain English" questions to test comprehension of written code.

Tracing: A reading component of code which involves the ability of the student to trace code and not be distracted by variables and memory allocations. This task also includes the ability to understand and use diagrams to explain code.

Writing: A writing component, where students write a small program to solve a problem.

Figure 1 shows several instances of Explaining, Tracing, and Writing questions given in an examination at Victoria University. Note that when applying the framework, the choice of questions within types is decided by the local institution.

The first BRACElet workshop participants selected Bloom's taxonomy (Bloom, Englehart, Furst, Hill, & Kraythwohl, 1956) as the educational model for developing the common set of question types for use in examinations. The devised types were concerned with students' ability to trace, read, and reason about code which related to Bloom's levels of *Understand, Apply,* and *Analyse*. After this early experimentation with Bloom's classification, the SOLO (Structured of the Observed Learning Outcomes) taxonomy of Biggs and Collis (1982) was adopted to provide an agreed framework for classifying student responses to computing problems, and in particular, analysing the data from the "explain in plain English" questions (Whalley, Clear, & Lister, 2007). The appeal of using the SOLO taxonomy has been that testing has shown that the SOLO categories described in Figure 2 provide a reliable and repeatable classification of students' responses to the 'explain in plain English' questions (Clear et al., 2008).

Explaining

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

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

Explain the purpose of the following segment of code.

```
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);
```

Tracing

What is printed to the screen by the following code?

```
     for (int count=0; count<4; count--)
        System.out.print(count);
a) 0 + 0 + 0 + 0 + 0 + 0 + …continuously
b) 0 + 1 + 2 + 3 + 4 + 5 + …continuously
c) 0-1-2-3-4 …continuously
d) no output to the screen
```

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

```
int n = 4;
for (int row=1; row<=n; row++) {
  for (int column=1; column<=n; column++)
     if(row ==1||row ==n ||column ==1||column ==n)
         System.out.print ("* ");
     else
        Sytem.out.print ("  ");
     System.out.println( );    }

a) *         b) * * * *    c) * * * *    d) * * * *
   * *          *     *       * * * *         * * *
   * * *        *     *       * * * *           * *
   * * * *      * * * *       * * * *             *
```

Writing

Write a segment of code that will show the sum of 1 to n for every n from 1 to 20. That is, the program prints 1, 3 (the sum of 1 and 2), 6 (the sum of 1, 2 and 3), and so on.

Write a segment of code that continually asks the user to enter a list of positive numbers. When the user has completed their list, they type a negative number that stops the input process. Once input is complete, the program prints out the average of the numbers to the screen.

Figure 1: Two examples of each of the three BRACElet examination question components of Explaining, Tracing, and Writing.

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

Figure 2: The SOLO categories for the students' answers to the 'explain in plain English' questions as described by Whalley & Lister (2009).

The SOLO taxonomy describes a set of learner levels of comprehension, and it can be used as a measure of a student's level of abstraction and understanding. Novices tend to give Prestructural or Unistructural responses as they form concrete representations based on the code, whereas more skilled students tend to give Multistructural responses until they develop an advanced understanding based on abstract representations which is evidenced by Relational responses (Whalley & Lister, 2009). As eloquently put by Lister, Simon, Thompson, Whalley, and Prasad (2006), students who provide more abstract Relational responses are able to "see the forest and not just the trees." As an illustration, using the first instance of the Explain problem type given in Figure 1, a Unistructural response would be 'it puts value c into a', a Multistructural response would be 'line 1 puts value b into c, then line 2 puts value a into b before line 3 which puts the value c into a' whereas a Relational response correctly notes 'it swaps the values of a and b'.

The importance of using SOLO taxonomy is that it can be used as a measuring stick of an individual student's understanding in learning to program. Various studies have noticed that students tend to answer consistently at one SOLO level within a question type. For example, a student who gives a Multistructural answer in explaining one problem will tend to give a same response in a problem of equal difficulty. As well, it is noted that students who give Relational answers to explaining questions, tend to score very well on code tracing and writing questions whereas students who give Unistructural or Multistructural responses when explaining questions often are seen to make only rudimentary code writing attempts. Importantly, these patterns of learning have been reported repeatedly across many institutions and they occupy much of the BRACElet body of research literature. Some instances of SOLO analyses of end-of-semester examinations can be found in Lister, Fidge, and Teague (2009), Lister et al. (2006), Lopez et al. (2008), Sheard et al. (2008), Thompson, Whalley, Lister, and Simon (2006), Venables et al. (2009), and, Whalley et al. (2006).

Wearing the Assessment BRACElet: The Victoria University Experience

For some time, computing academics at Victoria University have been observing the BRACElet project through the various publications of outcomes at key computing education research conferences. The focus on novice programmers through a multi-institutional approach seemed a promising mechanism through which common patterns of learning to program could be uncovered. Additionally, the project would provide a valuable opportunity to examine our own teaching practices and to receive support to improve local assessments.

A commitment to join the research effort was made in 2008 when one of the authors joined a BRACElet workshop. Most attendees provided their most recent examination question scripts, which were shared without prejudice to any specific institutional background. The aim of the exercise was to decide the next iteration of question types and to display the range of suitable questions that fit within the framework. This exercise was most enlightening and, in itself, encouraged collegiality and created a collaborative atmosphere offering all members the opportunity to benchmark their own examinations. After the workshop, an invitation was extended to a key investigator of the BRACElet group to visit Victoria University and discuss the possibility of running a BRACElet iteration on the usual end of semester examination for introductory programming.

Analysis of our scheduled end of semester examination showed that it was comprised of three types of tasks: multiple choice questions, short answer coding questions, and full code writing questions. Through benchmarking our examination with others from the workshop, it was decided that much of it already fitted within the BRACElet framework. The multiple choice questions were easily identified as code tracing of varying degrees of complexity, and these also tested the students' knowledge of basic programming constructs. The short answer questions were revised to become the 'explain in plain English' questions for which the SOLO taxonomy could be applied in the analysis of responses. The full code questions remained as writing questions.

When our examination was run, data were collected across the cohort of 32 students. The data were analysed on an individual question basis and by pairing of questions looking for support, or otherwise, of the consistency of performance at specific SOLO levels as reported by Sheard et al. (2008). As well, we studied our student responses to see if they supported an earlier study by Lopez et al. (2008) who found that explain questions together with reading and tracing questions accounted for almost half of the variance on the code-writing portion of their examination.

Our analysis, like others, treated student performance on code writing questions as the dependent variable and examined the relationship between code writing questions and non-code writing questions. Despite noticing some variation in the strength of relationships due to the nature of the questions being asked, we did find significant relationships between tracing code, explaining code, and writing code without the need for the more sophisticated statistical models used by Lopez et al. (2008). The analysis of our results demonstrates that the combination of tracing and explaining questions, more so than each skill independently correlates highly to code writing skills. Therefore, our work supports the BRACElet notion of a hierarchical development of skills in learning to program. For a more detailed description of the analysis, see Venables et al. (2009).

The Local Impact of BRACElet

Much has changed in the assessment of students studying introductory programming at Victoria University since the first implementation of the BRACElet framework. The BRACElet approach calls to account many traditional computing examination practices that extensively reward concrete styles of reasoning – for example, by asking students to provide the value in a variable after a piece of code has finished executing, or by asking them to reproduce code that was studied in class during the semester. BRACElet argues that a student's performance on such tasks is not a good predictor of a good understanding, rather students require a more abstract grasp of programming if they are to write their own computer programs successfully. In our lectures and tutorials, we are more aware of the hierarchy and the mental hurdles that students must climb in their journey to programming competency. We have structured practice problems to more closely represent the natural progression students take as they learn programming and our assessments have been modified to reward these various steps. Consequently, we test more uniformly the various programming skills using the BRACElet format. To help develop and move students to a more relational understanding of programming, we now use regular formative assessments that practice code reading and explaining. In our assessments, we better reward answers that see the 'big picture' over detailed descriptions and thus encourage students to develop more generic problem solving and abstraction skills.

Additionally, through the mentoring process of fitting our examination to the BRACElet specification, we were introduced to new type of examination question labelled as Parson's problems (Parsons & Haden, 2006). Parson's problems are like puzzles of code fragments that can be used for solving a problem. As illustrated in Figure 3, typically pairs of possible lines of code are offered to students in random order. Students must select the correct lines of code and place them in the most appropriate order. It is hypothesized that Parson's problems provide useful insights into the transition between tracing and code writing skills as placing code in the correct order shows one type of understanding whilst identifying the correct syntax another type of comprehension (Denny, Luxton-Reilly, & Simon, 2008).

The lines of code provided below are jumbled. You are required to select the correct line of code from each pair, and place the selected lines of code in the correct order to define the method removeAllAs *for removing all occurrences of the character* 'a' *from a* String. *When the lines are correctly ordered, the code returns a* String *without any* 'a' *characters in it.*

	Alternate lines of code	Letter of the correct line of code (use `A` or `B`)	Order of the correct line of code (use `1, 2, 3,`….)

A
B
	`public String removeAllAs (String word) {`
	`public String removeAllAs (word) {`
		A	1

A
B
	`return word;  }`
	`return result; }`
		B	6

A
B
	`String result = " ";`
	`String result ;`
		A	2

A
B
	`if ( word.charAt == 'a'  )`
	`if ( word.charAt != 'a' )`
		B	4

A
B
	`result = result + word.charAt(i);`
	`result = word.charAt(i);`
		A	5

A
B
	`for ( int = 0; i < word.length( ); i++)`
	`for ( int = 0; i < word.length; i++)`
		B	3

Figure 3: An example of a Parson's problem given in an examination at Victoria University. Solutions have been highlighted in yellow.

The Broader Impact of BRACElet

Through involvement in collaborative examination setting and data collection, academics are nurtured through the assemblage of evidence to make informed decisions about their own assessment practices. By discussing assessment practices collectively, BRACElet meetings provide a forum for academics to see what goes on beyond the walls of their institutions, and they afford the opportunity to join like-minded educators in attempting to understand how their students learn. A BRACElet gathering offers a mentoring environment that allows the opportunity for academics to self assess in a non-threatening atmosphere and learn best assessment practice. It is hoped that this better understanding by lecturers will lead to better teaching and overall learning outcomes for students. The collegial sharing of assessment resources amongst lecturers means that students are more likely to be assessed in a manner that mirrors their learning developmental steps and students can be tested more uniformly, within programs and across institutions.

Over time through staged iterations, BRACElet has continued to challenge and test various folk pedagogical issues in the computing education literature concerning novice programmers. When the earliest BRACElet papers were published, some academics reacted strongly to the results (Lister et al., 2004). Those with a folk pedagogical approach raised the objection, among others, that "writing programs and reading programs are disjoint intellectual activities." The BRACElet group sustains an evidence-based discourse, by carrying out further experiments that support or argue against such objections. In reality, it provides a platform for the development of new and more valid ways of assessing novice programmers and it fulfils a practical need for computer science academics to compare their teaching practice, their assessments, and their understanding of how students learn.

By active participation in a large scale, multi-institutional research program academics can contribute to the larger picture of understanding in their discipline and education research. The iteration at Victoria University is one of many 'baby steps' taken by the BRACElet collective. Through use of the BRACElet framework in writing the examination, data is collected and analysed for comparison with other institutions and previous published work. Thus, the work done at Victoria University has become a contribution to a larger pool of knowledge and understanding of how novices learn to program.

Conclusion

Like many other institutions worldwide, we have become concerned over failure rates in introductory programming units and it is clearly in the interests of both universities and students to see these failure rates decline. One of the biggest benefits in joining the BRACElet collective is that we no longer feel isolated in our teaching practice but instead become part of a larger community engaged in a dialogue about teaching computing novices. Through this conversation, we as educators have moved past "intuitive theories about how other minds work" (Bruner, 1963). We are better able to marry our research and teaching scholarships, especially as the research itself remains so close to practice. By building upon authentic assessment practice, the BRACElet approach continues to build computer science education research understanding of how novice programmers learn. The BRACElet movement is currently encouraging all researchers to make available their data to others through a data repository. When implemented, the repository will allow for independent repetition of analyses and the possible collation of several independent studies for subsequent data-mining.

Any discipline, computing or otherwise, will be enriched by actively encouraging a scholarly and evidence-based approach to teaching through action research. Such an approach has the benefit of dispelling traditional folk pedagogies through the incremental introduction of innovative assessment practices. This iterative process is not haphazard, but rather it involves reflection, planning, observation, action, and further reflection of teaching and assessment practice to offer real benefits through improvements to student learning outcomes.

References

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

Bloom, B., Englehart, M., Furst, E., Hill, W., & Kraythwohl, D. (1956). *Taxonomy of educational objectives: Handbook I: Cognitive domain*. Longmans, Green and Company.

Bruner, J. (1963). *The process of education*. New York, NY: Random House.

Clear, T., Whalley, J. L., Lister, R., Carbone, A., Hu, M., Sheard, J., Simon, B., & Thompson, E. (2008). Reliably classifying novice programmer exam responses using the SOLO taxonomy. *Proceedings of the 21st Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ, 2008), Auckland, New Zealand*.

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems. *Proceedings of the 2008 International Workshop on Computing Education Research (ICER '08), Sydney, Australia*. ACM Press, New York, NY.

Dunican, E. (2002). Making the analogy: Alternative delivery techniques for first year programming courses. In J. Kuljis, L. Baldwin, & R. Scoble (Eds), *Proceedings from the 14th Workshop of the Psychology of Programming Interest Group, Brunel University*, June 2002, 89-99.

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

Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Proceedings of the 14th Annual ACM SIGCSE Conference on innovation and Technology in Computer Science Education (Paris, France, July 06 - 09, 2009). ITiCSE '09*. ACM, New York, NY, 161-165.

Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006). Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education. (Bologna, Italy, June 26 - 28, 2006). ITiCSE '06.* ACM Press, New York, NY, 118-122.

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. *Proceedings of the 2008 International Workshop on Computing Education Research (ICER '08), Sydney, Australia*. ACM Press, New York, NY.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagen, D., Kolikant, Y., Laxer, C., 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, 33*(4), 1-16.

Miliszewska, I. & Tan, G. (2007). Befriending computer programming: A proposed approach to teaching introductory programming. *The Journal of Issues in Informing Science and Information Technology, 4*, 277-289. Retrieved from http://proceedings.informingscience.org/InSITE2007/IISITv4p277-289Mili310.pdf

Parsons, D., & Haden, P. (2006). Parson's programming puzzles: A fun and effective learning tool for first programming courses. *Proceedings of the Eighth Australasian Computing Education Conference (ACE2006), Hobart, Australia*, 157-163.

Philpott, A., Robbins, P., & Whalley, J. (2007). Assessing the steps on the road to relational thinking. In S. Mann & N, Bridgeman (Eds.), *Proceedings of the 20th Annual NACCQ. Nelson, New Zealand*, July 8-11, 2007.

Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., & Whalley, J. (2008). Going SOLO to assess novice programmers. *Proceedings of the 13th Annual SIGSCE Conference on innovation and Technology in Computer Science Education (ITICSE '08), Madrid, Spain*, ACM Press, New York, NY, 209-213.

Soloway, E., Bonar, J., & Ehrlich, K. (1983). Cognitive strategies and looping constructs: An empirical study. *CACM, 26*(11), 853-860.

Thompson, E., Whalley, J., Lister, R., & Simon, B. (2006). Code classification as a learning and assessment exercise for novice programmers. *Proceedings of the 19th Annual Conference of the National Advisory Committee on Computing Qualifications, NACCQ, Wellington, New Zealand*, July 7-10, 291-298.

Venables, A., Tan, G., & Lister, R. (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. *Proceedings of the Fifth International Computing Education Research Workshop (Berkeley, CA, USA, August 10 - 11, 2009). ICER '09.* ACM, New York, NY, 117-128.

Whalley, J., Clear, T., & Lister, R. (2007). The many ways of the BRACElet project. *Bulletin of Applied Computing and Information Technology, 5*(1). Retrieved July 19, 2009, from http://www.naccq.co.nz/bacit/0501/2007Whalley_BRACELET_Ways.htm

Whalley, J., & Lister, R. (2009). The BRACElet 2009.1 (Wellington) Specification. *Proceedings of the Eleventh Australasian Computing Education Conference (ACE2009), Wellington, New Zealand*, 9-18.

Whalley, J., Lister, R., Thompson, E., Clear, T., Robbins, P., & Prasad, C. (2006) An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. *Australian Computer Science Communication, 52*, 243-252.

Biographies

Grace Tan is a senior lecturer in Computer Science at Victoria University, Melbourne, Australia. Her research interests include investigations of innovative teaching methods, the development of graduate attributes, and issues related to female students in computing courses and Grace has published in these areas.

Anne Venables lectures in Computer Science and Information Technology at Victoria University, Melbourne, Australia. She has research and teaching interests in innovations in computing education and the application of intelligent systems in biological systems.