This is the author's version of a work that was submitted/accepted for publication in the following source:

Lister, Raymond, Corney, Malcolm W., Curran, James, D'Souza, Daryl, Fidge, Colin J., Gluga, Richard, Hamilton, Margaret, Harland, James, Hogan, Jim, Kay, Judy, Murphy, Tara, Roggenkamp, Mike, Sheard, Judy, Simon, , & Teague, Donna M. (2012) Toward a shared understanding of competency in programming: an invitation to the BABELnot Project. In de Raadt, Michael & Carbone, Angela (Eds.) Proceedings of the 14th Australasian Computing Education Conference (ACE2012), Australian Computer Society, RMIT University, Melbourne.

This file was downloaded from: https://eprints.qut.edu.au/47265/

© Copyright 2012, Australian Computer Society, Inc.

This paper appeared at the 14th Australasian Computing Education Conference (ACE2012), Melbourne, Australia, January-February 2012. Conferences in Research and Practice in Information Technology, Vol. 123. M. de Raadt and A. Carbone, Eds. Reproduction for academic, not-for-profit purposes permitted provided this text is included.

Notice: Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source:

Toward a Shared Understanding of Competency in Programming:
An Invitation to the BABELnot Project

Raymond Lister  
Faculty of Engineering and Information Technology,  
University of Technology, Sydney,  
Sydney, NSW, Australia  
raymond.lister@uts.edu.au

Malcolm Corney  
Faculty of Science and Technology,  
Queensland University of Technology,  
Brisbane, Qld, Australia  
m.corney@qut.edu.au

James Curran  
School of Information Technologies,  
University of Sydney,  
Sydney, NSW, Australia  
james.r.curran@gmail.com

Daryl D’Souza  
School of Computer Science and Information Technology,  
RMIT University,  
Melbourne, Vic, Australia  
daryl.d souza@rmit.edu.au

Colin Fidge  
Faculty of Science and Technology,  
Queensland University of Technology,  
Brisbane, Qld, Australia  
c.fidge@qut.edu.au

Richard Gluga  
School of Information Technologies,  
University of Sydney,  
Sydney, NSW, Australia  
richard@gluga.com

Margaret Hamilton  
School of Computer Science and Information Technology,  
RMIT University,  
Melbourne, Vic, Australia  
margaret.hamilton@rmit.edu.au

James Harland  
School of Computer Science and Information Technology,  
RMIT University,  
Melbourne, Vic, Australia  
james.harland@rmit.edu.au

James Hogan  
Faculty of Science and Technology,  
Queensland University of Technology,  
Brisbane, Qld, Australia  
j.hogan@qut.edu.au

Judy Kay  
School of Information Technologies,  
University of Sydney,  
Sydney, NSW, Australia  
judy.kay@sydney.edu.au

Tara Murphy  
School of Information Technologies,  
University of Sydney,  
Sydney, NSW, Australia  
tm@it.usyd.edu.au

Mike Roggenkamp  
Faculty of Science and Technology,  
Queensland University of Technology,  
Brisbane, Qld, Australia  
m.roggenkamp@qut.edu.au

Judy Sheard  
Faculty of Information Technology,  
Monash University, Caulfield East,  
Victoria, Australia  
judy.sheard@monash.edu

Simon  
School of Design, Communication, & IT, University of Newcastle,  
Ourimbah, NSW, Australia  
simon@newcastle.edu.au

Donna Teague  
Faculty of Science and Technology,  
Queensland University of Technology,  
Brisbane, Qld, Australia  
d.teague@qut.edu.au

Abstract
The ICT degrees in most Australian universities have a sequence of up to three programming subjects, or units. BABELnot is an ALTC-funded project that will document the academic standards associated with those three subjects in the six participating universities and, if possible, at other universities. This will necessitate the development of a rich framework for describing the learning goals associated with programming. It will also be necessary to benchmark exam questions that are mapped onto this framework. As part of the project, workshops are planned for ACE 2012, ICER 2012 and ACE 2013, to elicit feedback from the broader Australasian computing education community, and to disseminate the project’s findings. The purpose of this paper is to introduce the project to that broader Australasian computing education community and to invite their active participation.

Keywords: programming, objectives, assessment.

1 Introduction
It is very common for ICT degrees to incorporate a sequence of up to three programming subjects (also known as courses, papers, or units of study). Traditionally, these three subjects have formed part of the
compulsory ‘core’ of ICT degrees, particularly software engineering degrees. There is certainly a great deal of variety between institutions as to what is covered in the subjects. While the first is typically thought of as an introduction to programming, the second might be a direct continuation of programming concepts, a data structures subject, a subject addressing program access to databases, and so on; and even more variation can be expected in the third subject. Nevertheless, it appears to be the case that many ICT degrees identify three specific subjects as an effective programming stream, and it is these three subjects with which this project is concerned. Despite the centrality of these three programming subjects, computing academics remain dissatisfied with the effectiveness of these subjects. Many students are also dissatisfied: in a widely discussed paper describing the educational ‘Grand Challenges’ in computing, McGettrick et al (2004) note that:

“educators cite failure in introductory programming courses and/or [student] disenchantment with programming as major factors underlying poor student retention”.

In programming subjects, as with most Australian university subjects, the semester begins in each classroom with the ritual distribution of the subject outline, which provides a brief description of the subject, the topics to be covered, and the assessment scheme. Although such outlines often run to many pages, the document can be ambiguous. For example, consider the following objective, taken from the outline of an introductory programming subject at one of the universities participating in this project:

On successful completion of this subject, the student will be able to ... Demonstrate a working knowledge of the basic constructs in the object-oriented language Java.

Which constructs are the “basic” constructs? What does it mean to have a “working knowledge”, and how does a student “demonstrate” it? Figure 1 shows an extract from Computer Science Curriculum 2008 (ACM/IEEE, 2008), which manifests numerous similar ambiguities.

Furthermore, while students may think of outlines as the contract between them and their teacher, outlines are conscripted into many roles. For example, outlines are presented to professional accreditation committees as evidence that the required subject matter is being taught. Mappings are sometimes made from subject outlines to a university’s graduate attributes. When a student moves to a new university and seeks credit for prior study, outlines are used to establish subject equivalence between the two universities. Very importantly, outlines are also a contract between teachers. In a three-semester sequence of programming subjects, for example, the second and third semester teachers rely upon the outline of the previous subject to define what students should know at the start of semester – and sometimes those teachers feel justified in complaining that the students cannot actually do what the previous subject’s outline says they can do.

If we ignore the relationship of a given subject to other subjects, be they different subjects at the same university or equivalent subjects at other universities, even a given

<table>
<thead>
<tr>
<th>PF/Fundamental Constructs [core]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum core coverage time: 9 hours</td>
</tr>
<tr>
<td>Topics:</td>
</tr>
<tr>
<td>• Basic syntax and semantics of a higher-level language</td>
</tr>
<tr>
<td>• Variables, types, expressions, and assignment</td>
</tr>
<tr>
<td>• Simple I/O</td>
</tr>
<tr>
<td>• Conditional and iterative control structures</td>
</tr>
<tr>
<td>• Functions and parameter passing</td>
</tr>
<tr>
<td>• Structured decomposition</td>
</tr>
<tr>
<td>Learning Objectives:</td>
</tr>
<tr>
<td>1. Analyze and explain the behavior of simple programs involving the fundamental programming constructs covered by this unit.</td>
</tr>
<tr>
<td>2. Modify and expand short programs that use standard conditional and iterative control structures and functions.</td>
</tr>
<tr>
<td>3. Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, and the definition of functions.</td>
</tr>
<tr>
<td>4. Choose appropriate conditional and iteration constructs for a given programming task.</td>
</tr>
<tr>
<td>5. Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.</td>
</tr>
<tr>
<td>6. Describe the mechanics of parameter passing.</td>
</tr>
</tbody>
</table>

Figure 1: An extract from Computer Science Curriculum 2008 (ACM/IEEE, 2008)

subject varies over time. A change to the final exam is one of the most important yet subtle ways that a subject can change. If one loiters long enough in a departmental tea room around the time of semester when exams are being written, one will hear quite passionate complaints that Professor Bloggs has ‘watered down’ the final exam in a particular subject (e.g. by changing from free response to multiple choice). Changes to an exam often do not require changes to the subject outline or any other documentation, and can thus be made with little management oversight. Academics who teach downstream of that subject may not even be aware of the change until well after it has taken place.

1.1 The Relationship with Software Engineering

Software engineering as a discipline has wrestled with problems that are analogous to the pedagogical problems described above. A software engineering project usually begins with a long negotiation between the software developers and the various stakeholders. The negotiation culminates in a design document, often called a specification, which forms a contract between the software developers and the various stakeholders. Even the most comprehensive specification documents leave implicit some aspects of the proposed system, which are
remembered as shared understandings and oral agreements arising from certain meetings. After the implementation of the software begins, changes are inevitably made to the specification. Some changes are documented, while others remain implicit. Almost inevitably, not everyone is aware of, or happy with, some of the changes. While software engineering is by no means a solved problem, some aspects of the culture of software engineering can usefully be adopted to attack the problems in ‘pedagogical engineering’ described above. Thus academics with an ICT background bring a special perspective to specifying academic standards.

1.2 The BABELnot Project: Desired Outcomes

The above considerations led the authors to propose the BABELnot project. (See section 8 for an explanation of the name.) We successfully applied to the Australian Learning and Teaching Council (ALTC) for funds to support the project across the six participating institutions. The funded work of the project began in October 2011 and will continue until August 2013.

Our aim is to achieve consensus on a framework for describing learning outcomes in computer programming, specifically the teaching of programming in the first three semesters, and also on how to map between learning outcomes and exam questions. We understand that assessment in programming subjects is not restricted to written exams, and that some learning outcomes are often assessed by way of other forms of assessment such as assignments and practical tests; but these other forms of assessment are beyond the current scope of the project.

Our desired outcomes are:

- The creation of a bottom-up, action research approach to articulating learning outcomes, in the context of the first three programming subjects
- A culture of scholarly teaching in ICT, spanning institutional boundaries, with a discourse based in evidence rather than anecdote
- Exams that are a more valid and reliable indicator of student programming ability
- Better learning of programming by students
- Attraction and retention of more students to programming and software engineering

More specifically, our desired, measurable project deliverables are:

- A system for describing learning outcomes and assessment by written exam. A method for mapping between learning outcomes and exam assessment, applicable to the first three programming subjects
- The learning outcomes of the first three programming subjects, from at least the six participating universities, re-expressed within the system
- A document summarising an archive of exam questions, with meta-tags mapping the questions to the system, serving as examples for use by other academics
- Performance data from real students for a subset of the archived exam questions

In this project, learning outcomes will tend to be articulated in terms of a characterisation of suitable assessment tasks, for example,

“On successful completion of this subject, a passing student will be able to implement iterative algorithms on arrays, such as linear search, binary search and quadratic sorting algorithms, in approximately half an hour, without reference to external notes”.

Note that this is merely an illustrative example, not a recommendation of a standard to be adopted. Like Wright, Hadgraft, and Cameron (2010), it is not our intention to be prescriptive about what students at a particular institution should know, but rather to provide the framework within which academics at that institution might be prescriptive.

2 Background

This section reviews relevant prior work that motivated the development of this project and influenced the project’s design.

2.1 The BRACElet Project

A number of papers about the BRACElet project have been presented at past ACE conferences (e.g. Whalley et al, 2006). Work on BRACElet started in New Zealand in 2004. In 2007, the ALTC funded a fellowship project by Lister and Edwards to explicitly extend BRACElet into Australia (Lister & Edwards, 2010). The final BRACElet workshop was held in 2010 (Clear et al, 2011).

BRACElet recruited academics from multiple universities into an action research approach that involved the systematic collection of evidence from end-of-semester programming exams. 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, and collected and analysed the data from the end-of-semester exams. This process was repeated several times. Contrary to the intuitions of many computing academics, the project participants 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 ‘cannot see the forest for the trees’.

Many traditional exam questions, however, largely test the novice programmer on the lower level nuts and bolts, and learning outcomes are often expressed in terms of these nuts and bolts.

Three workshops were held within Australia during the funding period of the ALTC fellowship. A total of 21 Australian academics, from 14 different Australian universities, either attended these workshops or actively participated in the project electronically. Academics from at least seven Australian universities have used end-of-semester exam questions that were designed as part of this project. The project has also attracted international attention, with academics from 14 universities in seven countries actively participating in data collection and analysis. During the ALTC Fellowship funding period, 26 project participants (co-)authored 16 published papers, further disseminating the outcomes of the project.
2.2 Course and Unit of Study Portal (CUSB)

The Course and Unit of Study Portal (CUSB) is a software product that was developed jointly by three faculties of the University of Sydney as a university-funded project to provide a common curriculum mapping framework for a diverse range of professional degrees across Engineering, IT, Architecture, Design, Urban Planning, and Health Sciences. CUSB is currently used at the University of Sydney for over 240 degrees and over 2,500 units of study across four faculties. (Note: the University of Sydney uses the term ‘unit of study’ for what some other universities call a ‘subject’ or a ‘course’, and the term ‘course’ for what some other universities call a ‘degree’ or a ‘program’.)

CUSB captures the representation of multiple sets of graduate attributes and accreditation competencies (named curriculum goals or curriculum goal frameworks) and maps these to the relevant degrees. Each degree structure is modelled into the system as a collection of core subjects plus the rules governing the selection of elective subjects. Each graduate attribute or accreditation competency is in turn mapped to each assessment and learning outcome within each subject of a degree. This design enables the CUSB system to generate reports that visualize the curriculum coverage for entire degrees against any of the curriculum goal frameworks attached. These reports in turn enable quick identification of any gaps in goal coverage or any sequencing problems in the degree structure and facilitate accreditation or other quality control review processes. This is described in greater detail by Gluga et al (2010).

Richard Gluga, a PhD candidate at the University of Sydney, is creating an enhancement of CUSB, known as ProGoSS – Program Goal Progression (Gluga et al, 2012), which can be used to map the detailed objectives for the programming fundamentals curriculum designed by the ACM/IEEE (2008). The extension is intended to support systematic design, modelling and monitoring of student progression as part of curriculum design using Bloom’s Taxonomy (Bloom, 1956) and neo-Piagetian cognitive development theory (Lister, 2011). It also supports curriculum design by allowing for the specification of the level of achievement of both higher- and lower-achieving students, so that institutions can design a curriculum, and assess how well it is achieving its learning outcomes, with full regard to the range of achievement of the students who complete degrees.

2.3 Exam Question Classification

The aim of the Exam Question Classification project is to investigate the nature and composition of formal examination instruments used in summative assessment of introductory programming students, and the pedagogical intentions of the educators who construct these instruments. The project leaders presented their first draft of a classification scheme in a half-day workshop at the 2011 ACE Conference in Perth. On the basis of the feedback received from the 20 or so workshop participants, the project leaders revised their initial scheme. Subsequently, project members formed pairs and applied the revised scheme to analysing a total of twelve exams, from nine different universities in Australia, the UK, New Zealand, Finland and the USA. A paper on this work was recently presented at the Seventh International Computing Education Research Workshop (Sheard et al 2011) and another is being presented at ACE 2012 (Simon et al 2012).

Properties encoded about an exam question in the current draft of the classification include type of question (e.g. short answer, multiple choice), topics examined (e.g. data types, loops, OO concepts, program design), type of skill required (e.g. knowledge recall, hand executing code, writing code, explaining code), and difficulty (high, medium, low).

Getting academics to agree on classifications of specific questions has not proved to be straightforward. For example, two of the project participants recently classified an introductory programming exam consisting entirely of multiple-choice questions. While computing academics are divided on the value and validity of multiple-choice questions (Shuhidan et al, 2010), they are nevertheless widely used (Simon et al, 2012). On the issue of degree of difficulty (high, medium, low) the two participants agreed independently on only one third of the multiple-choice questions. On skill required (e.g. knowledge recall, hand executing code, explaining code) they agreed independently on one quarter of the multiple-choice questions. It is hardly surprising that subject outlines and other documents are ambiguous, when two experienced teachers of introductory programming exhibit such a low level of agreement on a set of multiple-choice questions. Before there can be a substantive debate on the content and assessment of early programming courses, there needs to be greater consensus on a framework for the debate – a framework that this project aims to provide.

2.4 Neo-Piagetian Theory

Wright, Hadgraft and Cameron (2010) describe a dialectic in learning outcomes, with one part of the dialectic being a “list of discrete outcomes or aspirational statements” as opposed to the other part of the dialectic, ”threshold learning outcomes [that] reflect the way engineers and ICT professional approach, think and do their work”. In this project we adopt a cognitive development perspective to transcend that dialectic.

Piaget developed a very well known constructivist theory about the different levels of abstract reasoning exhibited by people as they mature from child to adult. While classical Piagetian theory has been largely abandoned, neo-Piagetian theory has overcome many of the problems that led to that abandonment. The types of abstract reasoning are broadly the same in both theories; but in neo-Piagetian theory, people, regardless of their age, are thought to progress through increasingly abstract forms of reasoning as they gain expertise in a specific problem domain. Neo-Piagetians attribute the increasing abstraction in reasoning not to biological maturity but to an increase in the effective capacity of working memory, as the learner ‘chunks’ knowledge. Neo-Piagetian theory is not esoteric – the popular SOLO taxonomy (Biggs and Collis, 1982) is based upon neo-Piagetian theory.

In a paper presented at the 2011 Australasian Computing Education Conference, Lister (2011) proposed a way of applying neo-Piagetian theory to the learning of programming. He defined the development of
the novice programmer in terms of three neo-Piagetian stages. At a pre-operational stage, students can trace the changing values in a piece of code, but do not reason in terms of abstraction of that code. At a concrete operational stage, students can reason in terms of abstractions, but only in the context of specific code. At a formal operational stage, students can reason in terms of programming abstractions without recourse to explicit code examples. Lister’s stage theory has already been adopted by CUSP participants at the University of Sydney, and empirical results from Queensland University of Technology (Corney et al, 2012; Teague et al, 2012) add support to the proposal.

3 Dissemination Strategy

There is very little point to this project, or to any other innovative, education-related project, if the outcomes of the project remain private to the direct project participants. In many respects, the success of any innovative, education-related project should be assessed by the degree of dissemination of the outcomes.

By ‘dissemination’, we do not simply mean the distribution of information via publications and seminars (although distribution of information is an essential component of a successful dissemination). For the authors of this paper, ‘dissemination’ is to be measured by the extent of adoption by others of the materials and techniques developed by the authors of this paper.

It is well documented that dissemination, as the term is used in this project, is difficult. Few innovative, education-related projects have succeeded at dissemination (Gannaway et al, 2011; McKenzie et al, 2005; Southwell et al, 2005 & 2010). To improve dissemination, the ALTC explicitly adopted a Dissemination Framework (ALTC, 2006), which has also guided the authors of this paper. Even with the untimely demise of the ALTC, this dissemination framework is likely to influence the design of Australasian education projects well into the future.

As advocated within the ALTC Dissemination Framework, this project has adopted an ‘engaged’ model for dissemination:

“involving consultation, collaboration and support for ongoing dissemination both during the project and after the project is completed”

Consequently, the dissemination of this project begins early in the project (indeed, it begins with the publication of this paper) and will continue throughout the project, based on proposed full-day workshops held at roughly six-monthly intervals in conjunction with major computing education conferences:

- ACE 2012 (January, Melbourne)
- ICER 2012 (August, Auckland)
- ACE 2013 (January, Adelaide)
- iTiCSE 2013 (June/July, Canterbury, UK).

While the first of these workshops is now confirmed, the other three will be subject to proposal and acceptance at the respective conferences. As the workshops also serve to define project stages and milestones, if any of the proposals is not accepted, alternative dissemination mechanisms will be formulated.

The workshops will be open to all interested academics, and will probably not require a registration fee.

The budget allocation for dissemination and evaluation workshops includes a limited number of ‘scholarships’ that will pay the registration fee for ACE 2013, to be held in Adelaide. These scholarships will be awarded to people outside the project who contribute documents, data or other material that manifestly advances the project.

The iTiCSE working group reports are among the most influential and highly cited papers in computing education. Thus an iTiCSE working group in 2013 will maximise the potential for international dissemination.

3.1 Monthly Meetings

As part of the project, full-day meetings will be held each month in at least two of Melbourne, Sydney and Brisbane.

The project values collaboration, so these meetings are not necessarily closed, and researchers not currently involved in the project may be invited to attend them. However, while the six-monthly workshops are open to anyone even if they merely wish to observe, an invitation to a monthly meeting will be made on the assumption that the invitee will play an active and continuing role. For the types of active roles that are suitable, see Section 5.1, ‘Rules of Engagement’. A person seeking to join the project under this arrangement may need to make an explicit time commitment. A 10% time commitment is roughly two days a month, and with such a level of commitment a person might spend one of those days at a project meeting in their own city and the other day working independently to prepare for the next meeting.

4 Project Organisation

This project unifies three existing projects, spread across six universities in three Australian states:

- **Exam Question Classification**: As described earlier, this sub-project is investigating the nature and composition of formal examination instruments used in summative assessment of introductory programming students, and the pedagogical intentions of the educators who construct these instruments.

- **Syllabus Specification**: This sub-project builds upon the CUSP system discussed earlier, to create a new system into which we can map the detailed curriculum of programming subjects.

- **Exam Question Generation and Benchmarking**: This sub-project aims to include some common questions in exams at participating institutions, and to benchmark student performance on those questions.

Each month there will be up to three one-day meetings in Brisbane, Sydney and/or Melbourne. The ‘major’ meeting will be attended by all participants from the city where it is held, along with one representative from each participating institution outside that city. One or two further meetings will be smaller, involving just the
project leader and the project participants located in that city.

4.1 Rules of Engagement
All project participants have agreed to the following rules about how they will work together:

- All members will help to procure the outlines and other public documents for the relevant programming subjects at their respective universities.
- All members will assist in rewriting those documents into the CUSP-derived system adopted by the project.
- Project members who teach one of the programming subjects will:
  - Provide other project members with the opportunity to run short, in-class, formative assessment exercises.
  - Give full consideration to using in their summative tests and exams the questions devised as part of this project. However, the final decision on the inclusion of any summative test or exam question remains with that teacher.
  - Consider contributing some of their own questions to a common, public pool of questions.
- All members will approach the (other) teachers of the first three programming subjects at their institutions, to request test papers, exam papers, and class performance data on those papers. Ideally, the papers thus solicited will be public domain, but the project will also accept and use papers on the understanding that they are to remain confidential.
- At least one member at each participating institution will complete any necessary ethical clearance process.
- Members interested in particular sub-projects will undertake tasks related to those sub-projects. Specifically:
  - Members most interested in syllabus specification will benchmark the enhanced CUSP system by encoding information about subjects at other institutions.
  - Members most interested in the exam question classification/archive will develop meta-tags for the classification/archive that has CUSP as its starting point, and will archive questions and information from the benchmarking of questions. They will also conduct and analyse interviews with relevant academics to explore the processes of writing and marking programming exams.
  - Members most interested in question benchmarking will provide their questions and student performance data for the classification/archive.
- In addition to the specific tasks listed above, all members will make some contribution across all parts of the project, and will actively pursue connections between the sub-project of primary interest to them and the other sub-projects.

Further matters still to be agreed by the participants include a protocol for authorship in papers produced within the project. What level of contribution is required to warrant recognition as an author? In what order should the authors’ names be listed? To these and similar questions Lister and Edwards (2010) propose answers that might well be adopted for the BABELnot project.

5 Evaluation Framework
A requirement of ALTC-funded projects of this size is that they have an external reference group and be formally evaluated by an outside evaluator. This is not an activity that we plan to leave until the end of the project. Instead, formative evaluation throughout will facilitate the attainment of better project outcomes.

While the most obvious role of the six-monthly workshops is dissemination, evaluation will also figure prominently at all the workshops. For example, by having workshop attendees perform training exercises such as classifying exam questions in the CUSP-derived formalism, we will collect evaluation data on whether the formalism can be understood easily and applied reliably.

The external evaluator will be appointed six months after the project begins, and will then work with the project team to develop an evaluation plan. The key sources of information for the evaluation will be: reports from the monthly meetings, dissemination events, and reference group review meetings; interviews with project members; and feedback sought from project members via surveys.

A working group will be proposed for the ITiCSE conference to be held in the UK in mid-2013. If the proposal is accepted, this working group will contribute to the summative evaluation of the project. An ITiCSE working group will provide a fresh set of academics, independent of the formative evaluations.

6 Conclusion
This paper represents a break in tradition, driven by our focus on dissemination. Traditionally, innovative education projects focus on their product, and do not report on their activities until either those activities are over, or at least a significant milestone has been attained. We describe that as the ‘disseminate late’ approach, and argue that this traditional approach has probably contributed to poor dissemination outcomes. Instead, we advocate a ‘disseminate early and often’ approach, which is what we are implementing in this project, primarily through the six-monthly workshops, but also through the very act of writing this paper – we are like software engineers who advocate writing the software tests early, even before writing the code, for well known reasons that we believe are analogous to why innovative education projects should begin dissemination early.

Through beginning the dissemination early, we now have the luxury of using this paper to invite others to participate in this project. Most may elect to attend any of the six monthly workshops, but others may accept our invitation to take on a more active role, and join us in the monthly meetings.

It is just over 10 years since McCracken et al’s (2001) paper appeared. That paper directly and indirectly inspired a raft of multi-institutional collaborations in computing education. However, most of those projects were either short-lived, or (like BRACElet) were loosely organised, with much of the work being done without...
formal financial support. Our project may be a prototype for new generation of multi-institutional projects. Our project is formally funded, and with that comes a commitment to deliver. That in turn leads to a more formally organised project structure. Also, applying lessons that have been learnt over the last ten years, the project has dissemination built explicitly into its structure.

As previously mentioned, this project builds upon the University of Sydney’s existing CUSP system, which is currently used for over 240 degrees across four faculties at that institution. It is therefore possible that, after this project is formally over, the extended version of CUSP would be adopted by the current users: a cycle of innovation would be completed, by applying the outcomes and deliverables of this project to disciplines other than computing.

7 The Name BABELnot

Here is a brief explanation for readers who are curious about the name BABELnot. The name is deliberately reminiscent of the BRACElet project, of which Raymond Lister was a leader. The capitalisation of BRACElet reflects its continuation from the BRACE project, in which the name BRACE was an acronym. The name BABELnot calls to mind the biblical tale of the Tower of Babel, whose builders failed in their task because they lost the ability to communicate with one another. So while Babel highlights the challenge of communication when we speak different languages, BABELnot will help overcome that challenge by devising a common language in which programming educators may better communicate with one another on matters of standards and assessment within their subjects.

Acknowledgements

This project was approved for funding by the Australian Learning and Teaching Council, and is funded under the auspices of the Australian Federal Government’s Department of Education, Employment and Workplace Relations (DEEWR). However, the views expressed in this paper are solely those of the authors, and not the views of the ALTC or DEEWR.

References


McKenzie, J, S Alexander, C Harper, & S Anderson (2005). Dissemination, adoption and adaptation of


Teague, D, M Corney, A Ahadi, & R Lister (2012). Swapping as the ‘Hello World’ of Relational Reasoning: Replications, Reflections and Extensions. 14th Australasian Computing Education Conference (ACE 2012), Melbourne, Australia.
