Queensland University of Technology

Contemporary Developments in Teaching and Learning Introductory Programming: Towards a Research Proposal

Christine Bruce
Camille McMahon

Faculty of Information Technology
Teaching and Learning Report 2002 – 2

Series Editor: Dr Peter Bancroft

Faculty of Information Technology
QUT Garden Point Campus
GPO Box 2434
Brisbane QLD 4000
AUSTRALIA

Telephone: + 61-7-3864 2782
Fax: + 61-7-3864 9279
Web Address: www.fit.qut.edu.au

Previous Titles

Web On-Line Feedback (Duncan Nulty, Peter Bancroft, Scott Brewster, Dan Smith)
INFOQUEST Application in Coursework (Michael Middleton, Sylvia Edwards & Juliet Collins) (Report No 1999-1)
A Virtual Tutor for B-trees (Frederic Marie)
The Utilisation of Video-Streaming Technology for Delivery of Course Content in ITB510 (Communication Networks)/ ITN510 (Data Networks) and an Evaluation of its Impact. (Alan Tickle)
Jute – A Java Tutor System (Paul Roe, MiMi Truong, Peter Bancroft) (Report No 2002-1)

Planned Titles

First Year Information Technology Curriculum: A Blueprint for the Future (Christine Bruce) (Report No 2002-3)
Graduate Capability Statements in Bachelor of Information Technology Unit Outlines. (Melanie Fleming) (Report No 2002-4)

ISSN 1447-3429

Publication of the Faculty of Information Technology, Teaching and Learning Committee.

Printed in Brisbane, Australia by QUT Publications and Printing.

Series Editor: Dr Peter Bancroft

Correspondence about Faculty of Information Technology, Teaching and Learning report should be addressed to:

The Secretary
Teaching and Learning Committee
Faculty of Information Technology
GPO Box 2434
Brisbane QLD 4000
AUSTRALIA

www.fit.qut.edu.au/staff/committees/tandl/

Contemporary Developments in Teaching and Learning Introductory Programming: Towards a Research Proposal

Christine Bruce
Camille McMahon

Faculty of Information Technology
Teaching and Learning Report 2002 – 2

Table of Contents

EXECUTIVE SUMMARY III

1 INTRODUCTION 1

2 WHY RESEARCH TEACHING AND LEARNING INTRODUCTORY PROGRAMMING? 3

2.1 FAILURE TO REACH EXPECTED STANDARDS OR OUTCOMES 3
2.2 PASS RATES AND PROGRESSION OF STUDENTS 3
2.3 GENDER ISSUES 4
2.4 FOUNDATION SUBJECTS 5

3 OUTLINE OF DATA SOURCES 5

4 CURRENT PARADIGMS INFLUENCING TEACHING AND LEARNING INTRODUCTORY PROGRAMMING 7

4.1 CONSTRUCTIVISM 7
4.1.1 Modelling Learning in the Constructivist Paradigm - '3' P 8
4.2 CONCEPTUAL CHANGE 10
4.3 CONSTITUTIONALISM 11
4.3.1 Relational view of teaching and learning 11
4.3.2 Phenomenography and learning 12
4.3.3 Phenomenography in IT 13

5 TEACHING AND LEARNING INTRODUCTORY PROGRAMMING AT QUT 19

5.1 ITB410 SOFTWARE DEVELOPMENT 1 20
5.2 ITB411 SOFTWARE DEVELOPMENT 2 21
5.3 ITB107 - PROGRAMMING LABORATORY 21
5.4 OTHER RESEARCH AND ACTIVITY FOCUSSED ON TEACHING INTRODUCTORY PROGRAMMING WITHIN THE FACULTY 22

6 EMERGENT APPROACHES AND STRATEGIES TO TEACHING PROGRAMMING 23

6.1 SYNTAX-FREE APPROACH: 23
6.2 LITERACY APPROACH: 24
6.3 PROBLEM-SOLVING APPROACH 24
6.4 COMPUTATION AS INTERACTION 25
6.5 ACTIVE LEARNING 27
6.6 EMPHASIS ON CONSTRUCTIVE AND COLLABORATIVE LEARNING 27
6.6.1 Collaborative learning strategies 27
6.6.2 Collaborative teaching strategies 33
6.7 CONCEPTS FIRST 34
6.8 STUDIO-BASED APPROACH 35

7 TEACHING AND LEARNING PROGRAMMING – WHAT DO WE KNOW ALREADY? 37

7.1 - ABOUT STUDENT PERSPECTIVES/EXPERIENCES? 37
7.1.1 Gender: 37

7.1.2 Variation in experiences: 37
7.1.3 Expectations and preconceptions: 40
7.1.4 Culture: 40
7.2 - ABOUT WHAT HELPS STUDENTS LEARN? 42
7.2.1 Nature of assessment tasks: 42

8 SUMMARY 45

9 RESEARCH RECOMMENDATIONS ARISING FROM THE LITERATURE 46

10 TOWARDS A RESEARCH PROPOSAL 47

11 REFERENCES 51

Executive Summary

The teaching and learning of introductory programming in tertiary institutions is problematic. Failure rates are high and the inability of students to complete small programming tasks at the completion of introductory units is not unusual. The literature on teaching programming contains many examples of changes in teaching strategies and curricula that have been implemented in an effort to reduce failure rates. This paper analyses contemporary research into the area, and summarises developments in the teaching of introductory programming. It also focuses on areas for future research which will potentially lead to improvements in both the teaching and learning of introductory programming. A graphical representation of the issues from the literature that are covered in the document is provided in the introduction.

The paper introduces the problematic nature of teaching introductory programming and presents some of the reasons why research in the area should be prioritised. Failure of students to reach expected outcomes, such as the inability to program after undertaking an introductory programming subject; low pass rates and correspondingly low levels of progression of students into further programming subjects; and controversy about gender and programming, contribute to the problematic nature of teaching introductory programming. The fact that introductory programming subjects are often foundation units with associated large numbers of diverse students, and large administrative and teaching loads, is also a factor.

Constructivism, a learning theory which is currently strongly influencing the direction of programming education, is introduced. Although this has led to a variety of good principles of teaching practice, these have propagated independently of research into how students learn to program. The '3P' model of learning and conceptual change theory are briefly introduced as examples of constructivist models that could provide some theoretical basis upon which to further influence the teaching of introductory programming. Underlying constructivist theory is the idea that knowledge is actively constructed by the student, not passively absorbed from textbooks and lectures. Since the construction builds recursively on knowledge that the student already has, each student therefore constructs an idiosyncratic version of knowledge. Constitutionalism, and a relational view of learning, is then presented as a complementary theory to

constructivism, and a paradigm that has the potential to provide a positive theoretical basis for further influencing programming teaching practice. Constitutionalism differs from constructivism in that learners are seen to experience what they are learning in a small, identifiable range of different ways. An identifiable range of variation is thus assumed to be present in any given group (as compared with the idiosyncratic construction of every individual). This, therefore, allows learning to be 'managed'. An example of prior research into programming education within the constitutionalist paradigm, using phenomenography, is provided. Phenomenography is presented as a research tool that enables the collection of empirical data that will assist in developing teaching practice within a consitutionalist theoretical perspective.

The next section of the document outlines a range of examples of teaching approaches and strategies that are used in the teaching of introductory programming. Each of these approaches is described in terms of its main focus, and in some cases the results are revealed of cases where such approaches have been implemented and evaluated. The aim of this paper is to provide a background to current practice. It does not focus on specific teaching tools or instructional materials used such as various pieces of software or specific intelligent tutoring systems. There are many cases of these reported in the literature. Rather, the document seeks to reveal some of the broader approaches currently being tried in programming education.

Finally some of the major findings in past research into the teaching and learning of programming are presented. These focus on what we know about the students' perspective and experience, and what we know already about what seems to help students learn. Specifically, the discussion focuses on the issues of gender, variation in experiences, expectations and preconceptions, and culture as all influencing the learning experiences of students. We analyse the gaps in the current research findings, and pose a number of questions that will help form the basis of further research.

The main outcome of this background document has been to reveal that there has been little, if any, research on how students go about learning to program. There are many examples of innovative teaching practice that have been implemented, but these usually appear to have been developed independently of any research into the students' experience of learning programming. We suggest that research into how

students go about learning to program will reveal a pathway to more positive outcomes in the teaching and learning of introductory programming at the university level.

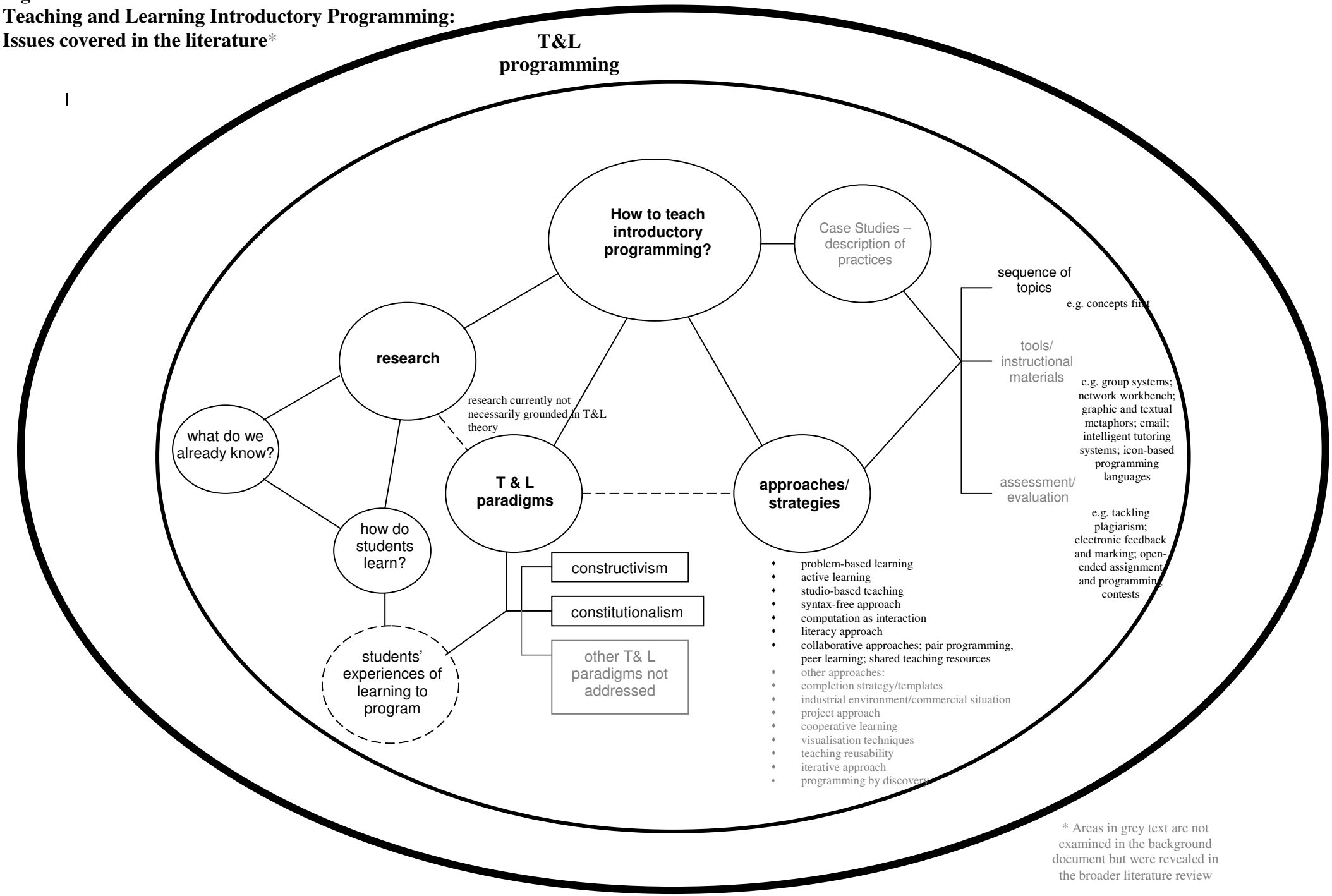
1 Introduction

This paper is a working document summarising some of the current issues and emerging directions in the area of teaching and learning introductory programming in tertiary institutions. The literature shows that the teaching (and learning) of programming is a perennial problem. The existence of high failure rates and students' subsequent inability to write simple programs at the end of a programming unit are just two of the issues penetrating Information Technology Faculties worldwide. In response to these problems there has been a trend to implementing changes to computer science curricula, teaching practice and even the environment in which students are taught, all in an effort to improve the outcomes of introductory programming units. Figure 1 represents a summary of the areas covered in this paper and includes a broader summary of the issues covered within the literature reviewed but which are outside the scope of this paper.

Amidst the attempts to improve outcomes of programming units, there appears to have been an overall trend towards constructivist teaching practices. Implicit within a constructivist paradigm is the notion that students learn in different ways, and that learning requires the student to actively construct personal meaning and understanding while thinking about previous experiences and considering alternative perspectives held by others (Van Gorp and Grissom, 2001, p.248). While many examples of good practice in teaching and learning programming are appearing, many are not grounded in understandings of students' learning experience. This paper briefly argues that the notions of conceptual change, learning and constitutionalism provide a theoretical basis from which research into improving the teaching and learning of programming can be effectively developed.

Following the brief introduction of teaching and learning paradigms, a summary of the programming unit structures at QUT is provided in order to create the context in which the teaching and learning of programming is being examined within the Faculty. Examples of emergent approaches and strategies used to teach programming are then outlined. Some of the major research findings in relation to what is currently known about teaching programming are also summarised. Particular attention is paid to research which examines learning to program from the students' perspective and to

Figure 1
Teaching and Learning Introductory Programming:
Issues covered in the literature*



* Areas in grey text are not examined in the background document but were revealed in the broader literature review

the contribution of such teaching and learning research in the IT field. The overall summary of research findings points to areas for further research. We propose questions and a research approach to pursue in order to help us address the gaps which continue to exist in our knowledge about what will actually improve programming curricula and teaching practice.

2 Why research Teaching and Learning introductory programming?

2.1 Failure to reach expected standards or outcomes

- 'The teaching (or perhaps we might more accurately say the learning) of programming is a problem. Few teachers of programming in higher education would claim that all their students reach a reasonable standard of competence by graduation. Indeed, most would confess that an alarmingly large proportion of graduates are unable to program in any meaningful sense' (Carter and Jenkins, 1999, p.1)

- 'The learning (and teaching) of programming in Higher Education is a perennial problem. Staff are all too familiar with students who approach their final year project work determined to avoid programming at all costs, presumably because they either cannot program or believe that they cannot' (Carter and Jenkins, 1999, p.1)

- 'Learning to program is a key objective in most introductory computing courses, yet many computing educators have voiced concern over whether their students are learning the necessary programming skills in those courses' (McCracken et al., 2001)

2.2 Pass rates and progression of students

General problems in programming subjects relate to pass rates and progression of students. Within the Faculty of Information technology at QUT, for example, failure rates are often in excess of 40% (Taylor et al., 2002). A study that commenced in

1995 at Monash University, aimed at tackling perceived problems in the teaching and learning of first year programming found that the main concerns were high failure rates, a low flow of students into higher degrees and a perception of a wide variation of teaching skills (Carbone et al., 2000).

- The research team, known as Edproj, focussed on the nature of learning and teaching in two Departments of the faculty of IT. Edproj comprised staff from Information Technology and the Faculty of Education. The initial Edproj investigation indicated the value to academics of studying student learning in a programming discipline (Carbone et al., 2000).

2.3 Gender issues

The issue of gender in programming is somewhat controversial, with some researchers arguing over whether or not women and men simply program in different ways (eg. Turkle, 1984). Others (e.g. McKenna, 2000; 2001) argue that this distinction is superficial and a 'damaging fallacy' (McKenna, 2000, p. 49) which has unwittingly led '…to a deepening of perceptions of programming and computing as a masculine culture' and to the implicit assumption of women as innately unsuited to the skills required for large programming projects in real organisations (McKenna, 2000, p. 37).

Whether or not women and men program differently, research into learning styles does tend to show differences in the way in which men and women approach learning and that this is a complicating factor in teaching programming at an introductory level. For example, in research reported by Carter and Jenkins (1999) the authors point to previous studies which have shown that female students lack confidence in this domain and that one significant corollary of this is often an underestimation of their own ability (Carter and Jenkins, 1999, p. 3).

- '…research shows that gender is a significant factor in determining the way in which students approach learning to program. A better understanding of the issues raised would lead to more effective teaching and thus better learning' (Carter and Jenkins, 1999).

2.4 Foundation subjects

Although not specifically related to programming, the following statement from Kay et al. (2000) relates to the challenges faced in any foundation course in computer science, and therefore, also programming.

- 'Foundation courses in computer science pose particular challenges for the teacher: the courses develop basic skills and attitudes which are important for effective learning in later courses; they are often large courses with correspondingly large management and administrative loads; teaching staff often find them demanding and, for some staff, they are seen as onerous. Now consider the critical role of foundation courses from the learner's perspective. They give a large cohort of students their first real taste of the discipline. Negative experiences may discourage students from further study. This is a very serious problem if those negative experiences are not indicative of the discipline as a whole' (Kay et al., 2000).

3 Outline of data sources

An extensive search across a range of data sources was undertaken over the period of November 2001 to January 2002 in order to develop a broad understanding of the current issues and trends in the teaching and learning of computer programming. Table 1 summarises the major databases and search terms used.

Table 1 Data sources

Database	*search terms*
Science direct	*browsed journals*
	computer programming and teaching
	computer programming and teaching and (university or undergraduate) all sciences
Proquest (all databases)	(teaching or learning) and programming
	refined: peer-reviewed only…
Proquest computing	"learning programming" or "learning to program" or "teaching Programming"
Ebsco	Journal: computer science education
Electric library	computer programming and (teaching or learning)
Springer link	teaching and programming (ABS)
Swetsnet Navigator	(ABS) teaching computer programming OR learning to program*
Synergy (Blackwells)	*browsed journals*

Webspirs (AEI; engine, alisa)	(teaching adj programming) or (learning adj programming)
	teaching adj computer adj programming
	learning adj computer adj programming
Emerald	*browsed journals*
First search (in Education >> select dissertations; Education index/ Eco)	computer programming teaching w computer w programming
IEEExplore	learning <and> computer <and> programming <in> ti

A range of journals, conference proceedings and home pages of institutions and academics were also accessed. Those resources which were used in the overall literature review are summarised below.

Journals

Association for Computing Machinery. Communications of the ACM
Computer Science Education
Computers & Education
Computers in Human Behavior,
Educational Psychology
Higher Education Research & Development
IEEE Computational Science & Engineering
IEEE Transactions on Education
IEEE Transactions on Software Engineering
International Journal of Human - Computer Studies
Journal of Computer Assisted Learning
Journal of Educational Computing Research
Journal of Educational Technology Systems
Journal of Object - Oriented Programming
SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)
T.H.E. Journal

Conference Proceedings

ASCILITE – Australian Society for Computers in Learning in Tertiary Education
ASEE Annual Conference
Asia Pacific Software Engineering Conference (APSEC)
Computer Science Education Research Groups International Workshop
Conference on Integrating Technology into Computer Science Education (ITiCSE)
Conference on Software Engineering Education and Training
European Conference for Research on Learning and Instruction (EARLI)
Frontiers in Education Conference

The Higher Education Research and Development Society of Australasia (HERDSA)
International Literacy & Education Research Network Conference on Learning
Software Engineering Education Conference

Institutions

The Centre for Informatics Educational Research, Open University, UK
Chalmers University of Technology – Centre for Educational Development
Monash Computing Education Research Group, Monash
Latrobe University, Division of Information Technology
School of Information Management Systems, Monash University
Computers and Education Research Group, University of Kent, UK

4 Current Paradigms influencing teaching and learning introductory programming

4.1 Constructivism

> *Learning is a complex process and, as described by the constructivist paradigm, knowledge is internally constructed by the learner. This paradigm encompasses a collection of different perspectives but acknowledges that learning involves making meaning of experiences and therefore that knowledge constructed by the learner is unique*
>
> (Fowler et al., 2001, p.270)

Constructivism is a theory of learning which claims that students construct knowledge rather than merely receive and store knowledge transmitted by the teacher. Constructivism has been extremely influential in science and mathematics education, but, until recently, has been much less influential in computer science education (Ben-Ari, 1998, p.1).

Within the constructivist paradigm, learning requires the student to actively construct personal meaning and understanding while thinking about previous experiences and considering alternative perspectives held by others (Van Gorp and Grissom, 2001, p.248). Knowledge is actively constructed by the student, not passively absorbed from textbooks and lectures. Since the construction builds recursively on knowledge that the student already has, each student will construct an idiosyncratic version of knowledge (Ben-Ari, 1998, p.1).

There is a range of views within the constructivist paradigm. For instance there is the debate between cognitive and social constructivists, based on the relative importance placed on individual construction or socio-cultural effects on learning. Generally, however, it might be summarised that knowledge construction depends on the following:

- What is already known
- Previous experience
- Organisation of these experiences
- Beliefs that the individual uses to interpret the reality of objects and events encountered (Fowler et al., 2001, p.264 citing Bruner 1962, Vygotsky 1978, Piaget, 1980).

Constructivism explicitly acknowledges '…that students do not learn well in a passive transmissive environment, but that they learn through a variety of knowledge building processes, and that teaching should encourage students to work actively towards understanding within a framework of personal responsibility and institutional freedom' (Booth, 2001a, p.170). 'Constructivist classrooms are often viewed as problem-solving environments manifested through three C's: context, construction and collaboration' (Van Gorp and Grissom, 2001).

In their editorial of the special issue of *Computer Science Education* focusing on Constructivism, Tony Greening and Judy Kay (2001) suggest that constructivist principles are now exerting strong influences on professional practice in computer science education. This is despite the low visibility that constructivism – as a body of theory – has within the discipline. In other words, constructivism has 'spawned a host of principles for good practice that have propagated independently of theoretical roots' (p. 168).

4.1.1 Modelling Learning in the Constructivist Paradigm - '3' P

The '3P model' demonstrates the relationships between teachers' thoughts and actions, students' thoughts and actions and the quality of learning outcomes[1]. In the 90's, John Biggs developed a systems approach to student learning, known as the 3P

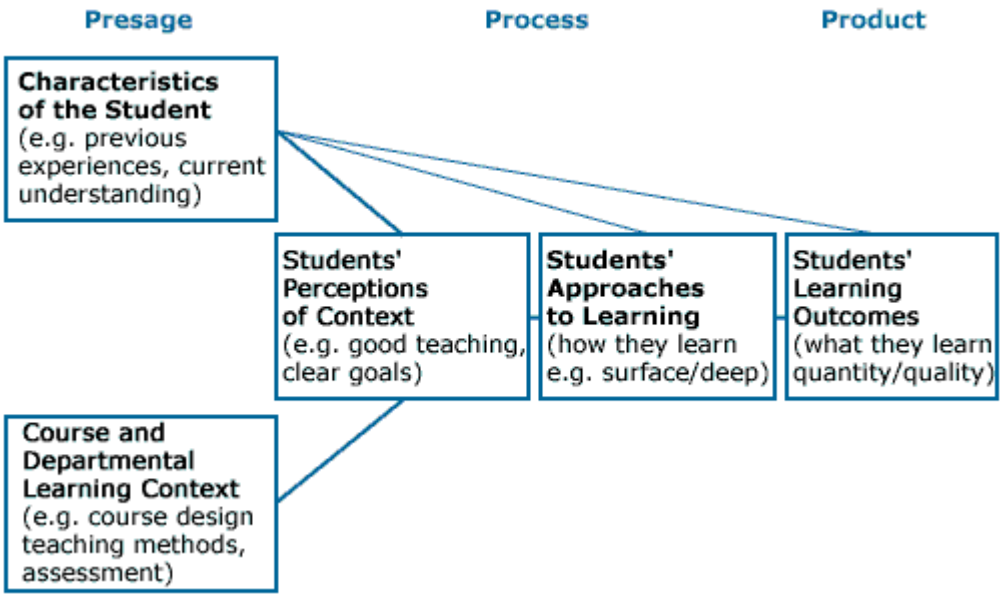
[1] Source: http://education.curtin.edu.au/iier/iier8/bookrev.html

model (presage – process – product). Biggs' (1999) 3P model of teaching and learning describes three points which are critical to the learning experience and its outcomes:

- Presage, before learning takes place;
- Process, during learning; and
- Product, the outcome of learning.

The model essentially describes the relationships between students' prior experience, the learning context, students' perceptions of their context, their approaches to learning and their learning outcome[2].

Figure 1
The 3P model[3]



Presage relates to what both the student and the teacher bring to the learning situation. For instance, student-based presage factors include how much they know about the topic already, their level of interest, their ability and their commitment to university. Teaching-based presage factors include the expertise of the teacher, what is intended to be taught, how the subject will be assessed and the ethos of the institution.

Student and teaching presage factors combine to influence the learning activities or the students' approaches to learning. For instance, Biggs (1999) and Trigwell and

[2] Source: http://www.learning.ox.ac.uk/iaul/IAUL+3+4+3+main.asp
[3] Source: http://www.learning.ox.ac.uk/iaul/IAUL+1+2+5+main.asp

Prosser (1997), refer to their influence on 'surface' and 'deep' approaches to learning. 'The surface approach arises from an intention to get the task out of the way with minimum trouble, while appearing to meet requirements' (Biggs, 1999, p. 15). Current teaching and assessment methods often encourage a surface approach. For example, short answer and multiple-choice tests, if designed poorly, allow rote-learning without necessarily understanding the content. 'The Deep approach arises from a felt need to engage the task appropriately and meaningfully so the student tries to use the most appropriate cognitive activities for handling it' (Biggs, 1999, p. 16). In other words, the student aims to focus on the underlying meaning of the content.

Outcomes of research by Shirley Booth (1997) suggest that deep and surface approaches are visible in students' experiences of learning to program (see section 7.1.2).

4.2 Conceptual Change

Schema theory suggests that all human beings possess categorical rules or scripts that they use to interpret the world. New information is processed according to how it fits into these rules, called schema[4]. Conceptual change theory focuses on the conditions whereby one's existing schema are modified by new knowledge and is constructivist in nature. Research indicates that changes in instruction must occur in order to promote conceptual changes in students and improve student learning. Elements present in teaching strategies which promote conceptual change include (a) maintaining student interest through hands-on instruction and relevant content and (b) an approach which integrates context, process and reflection with respect to the content. In order to promote learning through conceptual change, both the roles of the teachers and the learners should change[5].

A basic assumption in teaching for conceptual change is 'the key constructivist idea that construction of new conceptions (learning) is possible only on the basis of already existing conceptions' (Duit, 1999, p. 275). Because we use our existing conceptions to make our way about the world, we may not necessarily be conscious of them. Thus, the first and most significant step in teaching for conceptual change is to make

[4] Schema Theory: An Introduction, Sharon Alayne Widmayer, George Mason University.
[5] http://ww2.riverdeep.net/for_teachers/pro_development/iowa3/session2/2_simul_read.htm

students aware of their own preconceptions about the topic. Teaching for conceptual change primarily involves (a) uncovering students' preconceptions about a particular topic or phenomenon, and (b) using various techniques to help students change their conceptual framework. Teaching for conceptual change requires a constructivist approach in which the learner takes an active role in building and reorganising their knowledge (Davis, 2001).

4.3 Constitutionalism

4.3.1 Relational view of teaching and learning

> *There is a dualistic assumption underlying constructivism: thinking takes place in an inner subjective world, divorced from the outer objective reality and knowledge is constructed there by the individual through material and mental acts. In a phenomenological framework the fundamental unity between human beings and the world in which they live is assumed. Knowledge represents ways of seeing, experiencing, thinking about the world and it is constituted through the internal relation between the knower (subject) and the known (object).* (Marton and Neuman, 1989)

The most fundamental principle underlying a relational view of learning is that: 'learning should be seen as a qualitative change in a person's way of seeing, experiencing understanding, conceptualising something in the real world…' (Marton and Ramsden 1988, p. 271)

Some features of a relational approach are as follows:

- Learning is about coming to see the world differently
- Learning has a content as well as a process
- Improving learning is about relations between the learner and the subject matter, not teaching methods and student characteristics
- Improving learning is about understanding the students' perspective – once the students' conceptions of the phenomenon are explored and revealed, it becomes possible for alternative conceptions to be recognised as different, understood and perhaps adopted (Ramsden, 1988).

Additionally,

'*We have to know what views of a particular phenomenon we would like a learner to develop*' (Marton and Ramsden 1988, p. 272).

Within this relational view of learning, however, new categories do not necessarily replace the students' initial conceptions. In other words, students do not necessarily give up their earlier conceptions when they acquire new knowledge. Instead, the old and the new models may coexist as hierarchically ordered structures (Pozo, 1997 cited in Tynjälä, 1998).

The constitutionalist view differs significantly from constructivism in that learners are seen to experience what they are learning in a small, identifiable range of different ways (usually between three and seven). An identifiable range of variation is thus assumed to be present in any given group (as compared with the idiosyncratic construction of every individual) (Bowden and Marton, 1998; Marton and Booth, 1997). This essentially allows learning to be 'managed'.

Booth (1992, p. 262) argues that learning is about:

> *'gaining access to views of further faces (or conceptions of phenomenon) and developing an intuitive relationship with the object so that an appropriate face or set of faces is seen in appropriate circumstances'*

She noted that programmers need to have access to a complete range of conceptions of programming, and need to be able to adopt the conception or set of conceptions most appropriate to a given circumstance.

4.3.2 Phenomenography and learning

From a phenomenographic perspective, learning is seen as a broadening awareness, or widening experience of ways of seeing the world. In phenomenographic studies students' conceptions are usually presented in the form of categories of description. Learning is seen as a process of making sense of the world and the phenomena that constitute it, in the sense of coming to see the world and its phenomena in qualitatively new ways. The object of analysis is ways of experience at a collective level. The results are neither expressions of individual differences nor case studies of archetypes of identity; they are expressions of the potential ways of experiencing a phenomenon that might be found in a collective of people of similar characteristics to those involved in the data collection (Booth, 2001b).

> *[Learning]..means coming to an understanding of curricular content as a result of tackling various learning activities. As a result of the task, a new way of experiencing the content is reached. Thus there are two aspects to any learning situation which, while being inextricably intertwined and probably unconsidered for the learner, are important analytical aspects for the researcher. They are referred to as the 'what' of learning and the 'how' of learning; the 'what' concerns the quality of the understanding arrived at, or the perspective taken on, or the conception held of the content of the learning task, as a result of the learning activity; and the 'how' concerns more the nature of the act of tackling the learning task*
>
> (Booth, 1997, p. 135)

From a phenomenographic perspective, learning is shifting from not being able to do something to being able to do it, as a result of some experience (Booth, 1997, p. 136).

4.3.3 Phenomenography in IT

Phenomenography has its roots in educational research (e.g. Marton and Säljö, 1976; Svensson, 1977,), but has since been adopted in other domains including business (Sandberg, 1994), health (Barnard, McCosker and Gerber, 1999), information science (Bruce, 1999), information technology (Bruce and Pham, 2001) and information systems (Cope, 2000). Emerging phenomenographic research in areas other than education, has been interdisciplinary, often bringing together technology, education and a host discipline such as health or business. Extensive annotated bibliographies (Bruce and Gerber, 1995; Klaus and Bruce, 1997) and The Land of Phenomenography web-site (Hasselgren et al., 2001) provide a useful documentation of important work to date.

In Australia, phenomenography has been used in Information Systems (IS) research in two locations: La Trobe University in Victoria, and the Queensland University of Technology. These studies have pursued the latter two of three established lines of phenomenographic research : 1) the study of conceptions of learning, 2) the study of conceptions in specific disciplines of study, and 3) the study of how people conceive of various aspects of their everyday world that have not, for them, been the object of formal studies (Marton 1988, p.189). IS researchers have predominantly pursued the latter two lines of research.

At La Trobe University, the focus has been on IS Education, in pursuit of the second line of phenomenographic research. Cope's (2000) study represents a classical use of phenomenography in response to particular kinds of teaching and learning questions – what does it mean to learn about information systems? What kinds of learning outcomes can be found amongst groups of IS students and what kinds of learning outcomes are desirable? Students' different ways of seeing information systems have been the object of investigation, providing insights into how students ways of seeing differ from the views of experts in the field. The differences identified are educationally critical, in that each way of seeing information systems involves different ways of assigning meaning to, and perceptually structuring, such systems. Phenomenography has also been used to explore how information systems are conceived by academics, students and practitioners (Cope, Horan and Garner, 1997). Booth (1992; 1993) has similarly investigated students' different ways of conceiving programming and learning to program. Booth's work is explicated further in section 7.1.2 below.

If we accept that the character of university learning involves achieving a level of competence which involves seeing the world as experts do (Bowden and Marton, 1998), then educational research like this is critical to the design of effective professional education.

At the Queensland University of Technology, largely in the Information Systems Management Research Centre, the focus has been on information and technology experiences in both educational and workplace settings. Researchers here have been concerned with the third line of phenomenographic research, largely investigating people's experiences that have not been the formal object of learning. They are interested in investigating the different meanings associated with working with information and technology, with implications derived for education, training and systems design. Several recently completed studies each provide significant insights into important phenomena including geographical information systems, IT leadership, thesaurus use, ERP knowledge management and effective information use, and raise questions and implications for research and practice which have been raised by the authors referred to in the relevant publications. These studies are presented here in chronological order of their completion.

1) The first investigation was conducted by a team of Geographical Information Systems (GIS) and geography educators and researchers in Brisbane and Perth (Gerber et al., 1992). Twenty-six GIS vendors, Government and industry users, as well as educators and researchers were asked to describe their experience and views of GIS, including how they would use GIS for particular tasks. Outcomes of the investigation revealed that GIS were experienced in five qualitatively different ways, each involving different foci. GIS were found to be experienced as 1) a graphics interface – foci on a user and the graphical interface, 2) a geographical data organizer – foci on the user and the underlying database, 3) data collection representation – foci on the user, the graphical interfaces and the database, 4) the process of interaction between an expert in geographical information and extensive datasets to solve geographical problems – foci on an expert user and problem solving, and 5) an evolving spatial technology – foci on an expert user and research and development. Clearly the more sophisticated ways of interpreting GIS are associated with different foci, raising important questions for university and workplace educators, researchers, systems designers and implementers. What kinds of educational strategies will elicit and expand the foci of learners? How can systems be designed to facilitate more sophisticated ways of experiencing the technology?

2) An investigation of variation in effective information use (information literacy) (Bruce, 1997; 1999) was conducted in Australian universities. Sixty academics, librarians, IT professionals, academic developers and student learning advisors described their experience of using information effectively at work, and made observations about colleagues and friends. Outcomes revealed that professional employees, in technologically sophisticated workplaces, experience information literacy as 1) using IT for information awareness and communication – focus on IT, 2) finding information from appropriate sources – focus on information sources, 3) executing a process – focus on information process, 4) controlling information – focus on information control, 5) building up a personal knowledge bade in a new area of interest – focus on critical analysis, 6) working with knowledge and personal perspectives in such a way

that novel insights are gained – focus on intuition, and 7) using information wisely for the benefit of others – focus on personal values. Critical insights include the need for technology to be increasingly unobtrusive as information use becomes more sophisticated, the significance of collaboration or interdependence between colleagues and the need for partnership of information intermediaries. Questions arise for managers interested in fostering learning organisations, staff development and change management; information systems managers interested in training and education of systems users; and educators preparing learners for their chosen profession. How can university students and professionals be helped to use information more effectively; both through systems design and professional development or educational programs? How can the different foci be effectively harnessed in fostering workplace cultures suited to knowledge management and learning organisations?

3) Klaus (2000) investigated the varying conceptions of thesaurus use amongst neophyte researchers searching social science databases. Approximately ten participants discussed their experience of searching indexing and abstracting databases, and were encouraged to attend to how they worked with thesauri in that context. Three different kinds of experience were discovered. In the first, category 'zero', the thesaurus is essentially indistinguishable from the database, it is neither seen nor understood by the user who simply enters keywords and scans extensive sets retrieved for relevant data. In the second, category one, the thesaurus is experienced as an intrinsic part of the database, essentially inseparable from it. Searchers with this perspective use the thesaurus to improve their searching, essentially to broaden and refine queries. In the third and final category the thesaurus is understood as an entity separable from the database, the internal structure of the thesaurus is recognised and its evolving nature – and thus its deficiencies – is understood. Implications of this research may be drawn for both education of database users and for database design, in order to maximise the value of thesaurus features for users.

4) Interest in knowledge management and enterprise resource systems have been combined to investigate senior managers' understandings of knowledge management in the context of enterprise systems (Klaus and Gable, 2000). Six interviewees from major ERP vendors, consulting companies and government agencies participated in extensive interviews (up to seventy-five minutes) focusing on their experience of knowledge management. The depth of data proved sufficient for a phenomenographic analysis, revealing three differing categories of experience: Knowledge management is seen as 1) change management for implementing and maintaining an ERP system, 2) corporate information management based on and beyond an ERP system, and 3) integrating corporate information management and change management by means of an ERP system. Each of these different ways of experiencing knowledge management is associated with a set of foci that is configured differently in each specific experience, namely temporal – the phase of the system life cycle concerned; social – the categories of people involved; topical – the object of knowledge management, i.e. the system, business processes or data, or the business environment; dynamic – the state of information preferred; and instrumental – formal aspects of knowledge management such as the use of databases, templates and decision rules. Surprisingly this group made no distinction between information management and knowledge management. The research outcomes provide an important aid to communication, surfacing major differences in ways of thinking about knowledge management between vendors, consultants and client groups.

5) The business-IT relationship has also been subject to phenomenographic investigation (Stewart and Klaus, 2000). Twenty two senior business executives, IT executives and IT managers were interviewed to elicit their experience of leading business and IT executives and to probe the relationship between Business and IT professionals. Four distinctive ways of experiencing that relationship were identified: 1) an impersonal relationship in which one party undertakes a simple transaction of service with others, 2) An ambiguous relationship in which both parties are enmeshed in conflict prone contexts, 3) A supportive relationship characterized by both parties referring to each other in a positive manner, and 4) A lateral-creative relationship in which either the

business executives or the IT function assume the leading role in providing for the organisation's strategic framework. These outcomes provide a model for characterising Business-IT relationships in a range of organisations and may be applied to determine the 'health' of the relationship between these groups. Thus issues arise for chief executive officers, senior executives personally involved in such relationships and consultants with a focus on the business-IT function. How can we discern what kind of relationship exists? How do particular kinds of relationships come to be established? How can existing relationships be reconstructed to form more synergistic practices?

6) Stewart (2002) reports on a study to determine the variation in perception of competent leadership and leadership success between business executive and IT management communities. The objective of this project was to improve leadership practices within an industry partner agency in order to make more effective and strategic use of IT resources. In particular, the project sought to determine if there were any variations in leadership expectations of managers between the executives and managers of the IT unit and those of the executives of business units within the organisation. Any sources of difference could point to problems in the relationships, and significant differences in expectations could explain the lack of exploitation of IT by the business community. Phenomenography was used to determine an operant model of leadership as held by the senior business executives and the senior IT managers. Results revealed variations in the beliefs of 'good leadership'. The results also demonstrate the applicability of phenomenographic techniques to determining implicit leadership beliefs. Stewart reports that the phenomenographic approach was also well received by the senior management team, who found the questions useful in opening up dialogue between the different groups or communities. This was an important outcome for the research project. It led to reported improvements in the relationships between the business and IT management communities, and gained continued support for the project to move into the full benchmarking of actual leadership practices.

7) The collective consciousness of IT research (Bruce and Pham, 2001), has been the subject of investigation through two studies:

   a) an analysis of IT researchers' and industry professionals' views of the significance and value of IT research projects (Bruce and Pham, 2001; Bruce, Pham and Stoodley, 2002a; Bruce, Pham and Stoodley, 2002b; Pham, Bruce and Stoodley, 2002); and

   b) an analysis of how the IT research domain is constituted by IT researchers.

Both studies draw heavily on phenomenography (Marton and Booth 1997, Bowden and Walsh, 2000) in the research design. Outcomes of these projects reveal clear differences in ways of seeing, both within and between stakeholder groups. For example**,** *aspects of IT research may be interpreted very differently by researchers in the same collegial environment***.** If we assume that commitment, or willingness to pursue a research project is predicated, at least in part, on a valuing of that project, then we already have some evidence that such valuings may not be interpreted in the same way by prospective research partners. *Ways of seeing the IT research territory also vary widely*. Some emphasise the artefacts of information technology, others emphasise software engineering or information processing, communication or the dynamic character of the territory. The phenomenographic research approach has proven effective for these investigations, the different ways of seeing being clearly discernible in terms of different meanings and different awareness structures.

5 Teaching and Learning Introductory Programming at QUT

In order to develop the context for teaching and learning introductory programming at QUT, the following are direct extracts from course unit abstracts taken from the unit outlines available on the QUT Online Teaching system (OLT)[6]. ITB410 is the prerequisite unit for ITB411 and ITB107. We have thus provided more details from the ITB410 unit outline.

[6] https://olt.qut.edu.au accessed June 2002

5.1 ITB410 Software Development 1

This unit develops problem-solving and programming skills essential in professional programming and used in all the Information Technology majors. The skills are transferable to other programming languages and applications. The unit is part of the Common First Year, and is a pre-requisite to the units ITB411 Software Development 2 and ITB107 Programming Laboratory.

The objectives of the subject include the following:

*Theory***:** Students will be able to demonstrate knowledge of:

1. The principles and techniques of structured, object-oriented programming
2. The syntax and semantics of a modern object-oriented language
3. A range of problem solving methods
4. The software development lifecycle

*Practice***:** Students will be able to:

5. Design simple algorithms using a disciplined and structured approach
6. Implement simple algorithms using an object-oriented language
7. Desk check algorithms for logical errors
8. Compile, execute and test programs

The subject is delivered via a 2-hour lecture and a 1-hour tutorial per week. Lectures and tutorials emphasise both the underlying theory and the practical aspects of programming. It is expected that students will familiarise themselves with the lecture content from both notes and textbook before the lecture. Weekly problems are first examined by small groups in tutorials, then worked on individually during the week and some are peer reviewed under tutor guidance at the next tutorial. Partial code for tutorial exercises and other sample code is available via the World Wide Web. Assessment is via two assignments (worth 10% and 15% respectively) and one final exam (worth 75%).

5.2 ITB411 Software Development 2

Software Development 2 is part of the IT21 common first year, and follows on from ITB410. Students entering this unit are assumed to have a rudimentary grasp of programming, up to the point of exposure to iterative processes on arrays, the decomposition of small scale problems to appropriate methods, and the concept of parameter passing by value and by reference. Software Development 2 reinforces this base and builds upon it by introducing the concept of an abstract data type (ADT) and considering several examples. The unit prepares the student for future programming units, in any of the Faculty's schools, involving sophisticated data structures, industry standard 3GL languages, or large-scale software engineering.

The two hour weekly lectures are in the traditional style for large classes. The large amount of software presented is structured as a series of variations on a small number of fundamental conceptual themes. The one hour weekly tutorial introduces a small amount of novel material, but for the most part tutorials are an opportunity for the students to ask questions and clarify their understanding of the lecture material. The focus of the unit is both theory and practice, but with an emphasis on theory. The unit emphasises conceptual aspects of object-oriented programming, leaving heavy ‘hands on’ practise of these skills to concurrent and subsequent units.

Assessment is via 2 assignments worth 2% each, two assignments worth 3% each, two assignments worth 10% each and a final exam worth 70%.

5.3 ITB107 - Programming Laboratory

This unit follows ITB410 - Software Development 1 and provides a practical focus to cement the concepts introduced there by concentrating on the practice of programming so that the benefits of techniques learnt can be appreciated. In particular, emphasis is placed on well documented programs, making use of data and procedural abstraction and using a defensive approach to programming. Also, to be able to write a program to specification, on time and on budget requires that an appropriate process be followed. Students will be required to apply a process, which includes developing time management skills, quality and process awareness as well as defect tracking and correction skills.

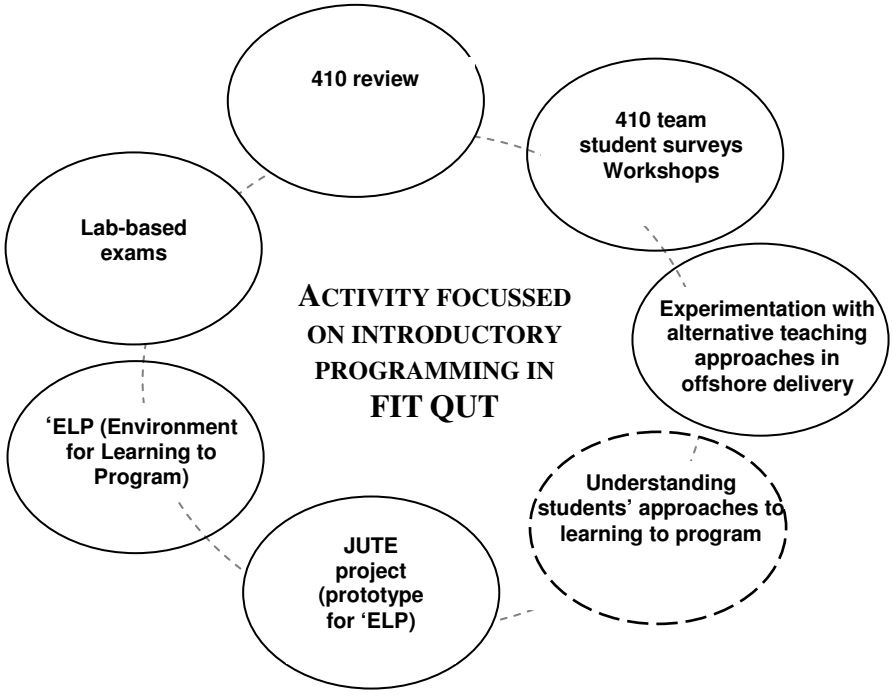
This unit uses the Java programming language as a vehicle to provide students with practical experience in designing, implementing and testing software. The approach to teaching will be to support and encourage the students' own exploration and development of a solution to an overarching problem. The assignments and the practical work will develop a solution to the problem. This unit will provide lectures; facilitated tutorial sessions; and supervised practical sessions. Students will learn generic problem solving skills to enable them to become effective software engineers and team members.

Students will research a problem and the required techniques to solve it within a group context. Evidence of the proper application of a personal process by each student will form part of the assessment. Assessment includes three assignments (worth 12%, 13%, 25% respectively), tutorial participation (10%) and a final exam (40%).

5.4 Other research and activity focussed on teaching introductory programming within the Faculty

Figure 2 summarises the research and activity that has taken place, or is currently in progress, within the faculty where the focus is on teaching introductory programming.

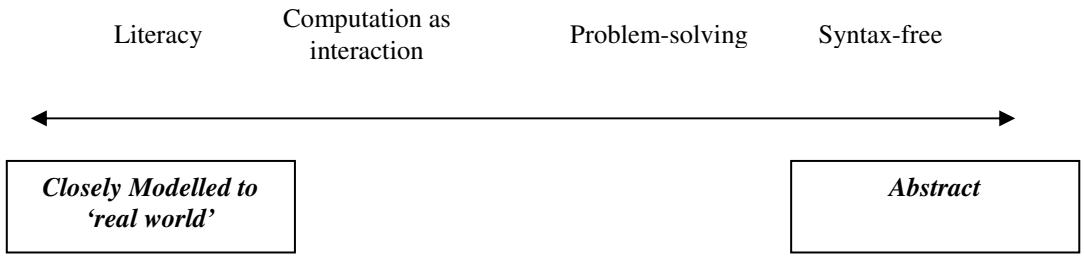
Figure 2
Activity focussed on introductory programming in FITQUT



6 Emergent approaches and strategies to teaching programming

The following section outlines a range of approaches and strategies to teaching programming which have been revealed in the literature. Fincher (1999) places the first four of the approaches to the teaching of programming in a preliminary taxonomic framework of approaches based on how closely they model activity in the 'real world'. Each approach addresses the central concern of relationship between the teaching of programming to the learning of Computer Science.

Figure 3
A preliminary taxonomic framework of approaches to teaching programming



(source: Fincher 1999, p. 12a4-4)

Additionally, the literacy and problem-solving approaches require minimal change to curriculum, whereas, the syntax-free and computation-as-interaction approaches require more adaptation of existing materials (across the whole computer science course too).

6.1 Syntax-free approach:

An approach espoused by Richard Bornat (1987), in which he recommends teaching programming as a skill separate from coding (i.e. without language). Indeed he asserts that (p. xvi);

> *…the "damage" caused by early exposure to a particular code … is real enough but is not caused by the evil properties of any particular notation; it is the delusion that to learn a code is to learn to program which is truly harmful.*

In his book Bornat (1987) presents five sections which encapsulate his approach to teaching and learning programming: Basic Concepts, Structured Instructions, Some

Extended Examples, Structures of Values and Transcribing other Codes (reflecting issues of importance when using the imperative languages popular at the time). All the exercises he presents in the book can be done with pencil and paper. To use them in an electronic environment requires translation into a programming language.

6.2 Literacy approach:

The central feature of this approach is that:

> *…learning to program is a new (and difficult) skill. Students need their learning to be supported in such environments, and the supports this approach provides are those which mimic the acquisition of the skills of reading and writing prose*
>
> (Fincher, 1999, p. 12a4-3)

This approach also separates the skill of programming away from the skill of expressing the program in code, but in contrast to 6.1 it has a focus on real world application.

Because of the older age group than most often learns to read/write (Fincher, 1999), the approach focuses on features of the learning process such as;

- aspects of **achievability**, (e.g. ensuring students can achieve a small working application within a 2 hour lab session)
- **motivation**, (e.g. setting problems/projects that result in software which resembles applications they will encounter in real world)
- **relevance** (in contrast to syntax-free approach – using a current, real language). A variation of this is the 'apprenticeship approach' (e.g. Astrachan et al., 1995) where students are assumed to be able to understand more complex code than they can write, so are given real-world examples to study and extend.

6.3 Problem-solving approach

This approach is also referred to as Problem-based Learning (PBL) and is often taught under the terms 'analysis and design'.

Barg et al., (2000) suggests PBL is characterised by:

- open-ended, authentic, substantial problems which drive the learning
- explicit teaching and assessment of generic and metacognitive skills

- collaborative learning in groups

But Fincher (1999) argues that PBL is often not a very useful pedagogical approach because it is assumed that;

a) the student wants to learn problem-solving
b) they usually do it via one single syntax
c) the student does not know how to problem solve

Fincher suggests that a more pedagogical based approach based on problem solving is that described by Barnes et al. (1997) where they describe how programming tasks were reconceptualised for the students away from a coding exercise towards an activity requiring a separate and distinct skill set. They derived a simple cycle of activity:

Understand – Design – Write – Review

This is then applied not only to programming tasks in a specific syntax, but across several courses and syntaxes. This allows the student to apprehend that *problem solving is a distinct set of behaviours that can be applied across many areas*. In other words, the student comes to understand that problem solving is a transferable skill.

6.4 Computation as Interaction

Stein's (1999) approach has been influenced by what she sees as a change in the paradigm underlying programming and programming languages and to the conditions/experiences of computing which students have before they come to be computer science (CS) students.

> *We live in a time of transition. Computer science is undergoing a Kuhnian revolution. The traditional foundations of our field are shifting, making way for an alternate conceptualization that better explains the phenomena we see. The previous metaphor—computation as calculation, sequencing steps to produce a result—was crucially empowering in computation's early history. Today, that metaphor creates more puzzles than it solves. We cannot even explain our field's best-known artifact—the world-wide web—in traditional terms.*
>
> (Stein, 1999)

Stein (1999) refers to the 'computational metaphor' and suggests that changing the computational metaphor – moving from 'computation as [sequential] calculation' to 'computation as interaction'– has far-reaching and fundamental effects on the way that we think and has particular implications for how and what we teach.

The approach is based on her argument that all contemporary CS students experience computers as 'multi-threaded, GUI-driven devices' (i.e. experiencing computation as interaction) and thus to present these students with a model of single-threaded problem-solving based on 'the sequence of calculations required to get from a particular instance of the question to the corresponding instance of the answer is cognitively inappropriate' i.e. it doesn't really correspond to the way that computation exists in the world at large.

Stein (1999) describes 'a new curriculum for the introductory programming course, i.e. for students with no prior programming experience. This course differs from the traditional one both in the questions that are asked and in the territory that is covered as a consequence. Every program that students encounter in this class is inherently concurrent and embedded in a context. Functionality to be implemented is always specified in terms of interactions and ongoing behavior' (Stein, 1999, p.13)

> *In this single semester course, students progress from simple expressions and statements to client/server chat programs and networked video games. Although this sounds like extremely advanced material, these topics proceed naturally and straightforwardly from the interactive computational metaphor. Because the programmer's questions concern the relationships between components, topics like push vs. pull, event-driven vs. message passing, and local vs. networked communication are integral aspects of this course. The curriculum exploits this shift in the fundamental story of programming to restructure what is basic and what is advanced curricular material. In other words, this course does not go deeper into the curriculum than a traditional introductory course; rather, it stands the traditional curriculum on its end.*
>
> (Stein, 1999, p.13)[7]

Beyond Fincher's (1999) framework of approaches, the following strategies are also being applied to the teaching of programming.

[7] See example class schedule in Fincher (1999).

6.5 Active learning

Gottfried (1997) describes how to provide effective instruction in computer programming within an active-learning environment. The use of active-learning does not in itself ensure success in this area, however, Gottfried (1997) found that they can provide effective instruction by:

- utilising a series of 'mini-lectures' based upon carefully prepared examples that illustrate key features;
- by providing students with copies of the examples and encouraging them to write their own notes on the examples;
- by assigning simple in-class programming exercises that reinforce the material presented in the 'mini-lectures' and
- by supplementing the in-class activities with weekly programming assignments of a more comprehensive nature.

Gottfried's (1997) paper describes each of these course characteristics in some detail. It also includes a list of features that work well, and another list of features, including some traditional teaching techniques, that we feel should be avoided.

6.6 Emphasis on constructive and collaborative learning

6.6.1 Collaborative learning strategies

The constructivist elements of collaboration and cooperation are highlighted in much computer science literature (Van Gorp and Grissom, 2001). It is suggested that collaborative learning provides the advantages of interchanging ideas among students and an increase motivation to learn (Vizcaino, 2000). Particular reference has been made to the evidence that girls tend to do better in computing environments where learning is collaborative (e.g. Gorriz and Medina, 2000). Vizcaino et al. (2000) argue that collaborative learning is especially good for learning programming because students naturally look for the experience and collaboration of other people through such means as email message lists, news and work groups, and advice from other programmers.

Lidtke and Zhou (1999) look at the Collaborative Laboratory as a key element in introductory computer science courses. Their approach supports groupwork from the

very first course rather than later in the CS course. Specific collaborative strategies include the following.

6.6.1.1 Peer Learning

The term peer learning is used by Wills et al. (1999) to broadly include collaborative and cooperative learning. It involves students working together as part of their learning experience (p. 71). In their study, the majority of teaching staff who had attended workshops on peer learning indicated that their students were more satisfied with the courses when peer learning techniques were used (Wills et al., 1999).

Collaborative learning has been shown to increase both the academic performance and persistence of new college students, improve accessibility for minority students, and encourages students to become part of a social network (Wills et al., 1999, p.73). Some other anticipated benefits of peer learning outlined in Wills et al. (1999) include:

- better retention of students because students have more opportunities to make contacts with peers and thus less social isolation.
- student performance in the course improves due to creating an environment with more active learning
- students will be better prepared for the work environment and thus be more successful in their jobs.

These anticipated benefits, however, were not formally evaluated in the Wills et al. (1999) study.

Peer learning tasks discussed in Wills et al. (1999) were broadly categorised into:

- *Get Acquainted Tasks*
- *Group Tasks in Class or Laboratory* e.g. a group quiz
- *Out-of-Class Projects e.g.* assign a large programming project in which each student group collectively designs and individually codes different aspects.

In addition to a range of different types of activities, Wills et al. (1999) outlines many other organizational details associated with using peer learning. These are briefly introduced below.

Group Composition

In developing specific tasks, participants generally suggested group sizes of 3–5 with pairs of students when appropriate and larger groups for less formal activities. Groups that are too large do not function well on larger projects. Long-term group assignments allow students to get comfortable within their group, but do not allow as many interactions between different sets of students.

Group Dynamics

One of the issues with large-scale, group exercises is the dynamics of how students work together. Participants at the workshops described in Wills et al. (1999) note that a teacher cannot simply tell students to 'work in groups'. Students must be taught to work cooperatively in teams and should understand how team members and team projects will be evaluated.

Grading

When a group project grade is a significant part of a student's course grade, then students need to believe that they are being fairly graded for their individual contributions or they may not be open to the use of cooperative learning.

Appropriate Tasks

Participants in the workshops discovered that there is not a standard use of peer learning. Rather, there are different types of activities that serve different purposes and are appropriate for different situations. It is important that an instructor use a peer learning activity that is a good fit for the learning objective and one in which it is both natural and beneficial for the students to work together. If students cannot see a benefit to working as a team, they may prefer to work individually rather than as part of a team. A good cooperative task needs to have *positive interdependence* between group members (Johnson et al., 1991).

A number of lessons learned in relation to implementing peer learning are also suggested in Wills et al. (1999):

- *Peer learning is important* but should not be used as the only technique
- *Start small* with 'low-risk' activities in class, then larger out-of-class projects
- *Instructors must be willing to relinquish control*
- *Group project grading can cause anxiety for students*
- *Group projects require careful planning by the instructor* (must consider the role of each group member)
- *Students need to see the benefit for group activities to work*

6.6.1.2 Pair Programming

> *Team programming usually means coordinating efforts of individual programmers who divide up the programming tasks for a large, complex system. Collaborative programming is used here to mean two programmers working jointly on the same algorithm and code*[8]

Pair programming differs from normal two-person team projects in its level of collaboration. Team projects are usually divided into 'my' part and 'your' part. However, with collaborative programming, the entire project is 'ours' (Williams and Kessler, 2000b).

Williams and Kessler (2000a) cite studies that show the general benefits of pair-programming:

- producing 'finished and tested code faster than ever…. nearly 100% bug free… Two programmers in tandem is not redundancy; it's a direct route to greater efficiency and better quality'.
- Leads to more confidence in programmers' own programming.
- Working in pairs, collaborative programmers perform a continuous code review which leads to efficient defect removal

[8] Source: http://c2.com/cgi/wiki?AcmOnCollaborativeProgramming

In Williams and Kessler (2000a) pair programming is studied in an educational setting (University of Utah). Students were asked, through a variety of means, to describe their experiences of pair programming.

The sample group:

Students were all familiar with programming, but not the languages used in the class.

The methodology:

Feedback from students was through web-based journals where they answered specific questions. Students also completed anonymous surveys on their collaborative experience. Also, in a final exam, students wrote a letter giving advice to future collaborative programmers.

Results:

Examined in relation to:

- Quality,
- Productivity and learning,
- Student morale and
- Teaching Staff workload

Quality. Collaborative programming and the effects of pair-pressure seemed to have a positive effect on the product (the final program/software), the capacity to meet assignment deadlines, and the number of defects in the programs (Williams and Kessler, 2000b). Additionally, the students performed much more consistently and with higher quality in pairs than they did individually – even the less motivated students performed well on the programming projects.

Overall, 95% of the class agreed with the statement *'I was more confident in our assignments because we pair programmed*' (Williams and Kessler, 2000b, p.6)

Productivity and learning. Students felt they were more productive when working collaboratively.

Student morale. The students were extremely positive about their collaborative experience. Students were happier and less frustrated with the class (Williams and Kessler, 2000b). Ninety-two percent of the students said they were more confident in their projects when working with a partner; 96% of the students said they enjoyed the class work more when working with a partner (Williams and Kessler, 2000b). Most enjoyed the experience, felt that they learned faster and better with a partner because it helped them learn when they had to explain something. Defect removal was also less frustrating (Williams and Kessler, 2000a).

Teaching Staff workload. The instructors felt more positive about the class. Assignments are handed in on-time and are of higher quality. Less questions came from students. There were less partner problems than with 'team-based' classes. The number of cheating cases teachers need to deal with is also reduced (Williams and Kessler, 2000b).

Despite the apparent benefits of the pair programming strategy to learning programming, it is not possible to conclude from Williams' and Kessler's studies (2000a; 2000b) whether the strategy has any effects on pass rates. It would be useful to consider these impacts as part of a similar study.

6.6.1.3 Other collaborative activities

Van Gorp and Grissom (2001) outline a series of constructivist and collaborative strategies to utilise in teaching and learning programming:

- ***Code Walkthroughs***: where students step through existing code and predict the output which helps students practice and better understand flow of control.
- ***Writing Code***: where groups write code to solve a small problem. Instructors provide guidance when appropriate but aim to let group members answer their own questions.
- ***Scaffolding***: which recognises that novices need additional support to solve a problem, so they aim to build on partly solved problems. Examples of this are where groups of students are given code to solve a particular

problem and are asked to insert comments to describe the semantics of the code. Alternatively, they may be given comments that describe an algorithm, and then students are asked to write code that corresponds to the comments.

- ***Code Debugging***: where students are given syntactically and logically buggy code and students contribute to finding errors. Constructive thinking is promoted further in this activity when students disagree on what is or is not an error in the code.
- ***Lecture Note Reconstruction***: where students are asked not to take notes in a lecture and at the end of the lecture time is given for them to reconstruct an outline of the lecture from memory. They then meet in groups to refine their notes further. (This activity also assists students develop their listening skills!) (Van Gorp and Grissom, 2001, p. 249-50).

6.6.2 Collaborative teaching strategies

The following strategies focus on collaboration from the perspective of the teaching staff in the development of resources across the faculty. Collaborative learning strategies may be used within such a framework.

6.6.2.1 Interfaculty Team approach

An example of this occurred between education experts and programming lecturers at Monash (Hagan et al., 1997) to foster 'ownership'.

- Previous to changes structure was (weekly): two hour lecture, two hour lab session, email questions to lecturer and tutor.
- Changes: (weekly) 2 x one hour lecture, two hour lab session, one hour discussion class with focus on educational techniques such as Predict-Observe-Explain, mimics, role playing and grids. Collaboration strongly encouraged and this facilitated understanding (while lab sessions facilitated hands on programming)
- Iterative approach to lectures i.e. topic covered a little at a time

Results:

- first semester similar results as those prior to changes

– in second semester, numbers failing or discontinuing were about the same, but the percentage of students doing very well (i.e., achieving distinctions and high distinctions) rose *from* 31% to 40%.

6.6.2.2 Shared teaching resources across faculty

> *In the current economic climate of diminishing resources, financial constraints and the ever-changing nature of the computing field, there is an increasing need to collaborate to provide shared materials for teaching. However, the "not invented here syndrome" that causes a continual reinvention of the wheel … and the egocentric nature of some academics can hinder the sharing of resources.*
>
> (Ellis et al., 1999)

Ellis et al. (1999) report on a strategy involving collaboration between staff in different/competing schools to implement a strategy to develop faculty-wide Java teaching resources to support first year programming. The group includes representation from the three main computing foci of the Faculty (computer science, commercial computing, network computing) as well as three different educational approaches (lectures and tutorials; problem-based learning; distance education). They developed a group process based on working together in the following stages:

- selecting the topic areas considered integral to all subjects for which the resources will be used;
- defining the details and identifying areas/concepts of a topic;
- determining basic, intermediate and advanced levels of information;
- determining appropriate educational techniques that support the desired learning objectives for the concept;
- investigating existing resources, and;
- building new resources

6.7 Concepts first

In addition to a collaborative approach, Lidtke and Zhou (1999) assert that a broad 'concepts first' approach is necessary.

> *Concepts of computing are emphasized, students work on systems problems, not textbook problems, express the solutions to problems as algorithms in a pseudo-language, and test these algorithms without conversion to a programming language. Students develop confidence in their understanding of the fundamental*

> *principles of computing, learn to work in groups, practice communication skills, and are prepared to learn a programming language to implement the concepts they have learned. Students with this background are well prepared for majoring in any area of computing*
>
> (Lidtke and Zhou, 1999 p. 12a4-23)

6.8 Studio-based approach

The teaching of the Bachelor of Information Management and Systems (BIMS) at Monash University has instituted a teaching model based on a studio approach (Carbone and Sheard, 2002). The traditional lecture theatre, tutorial room, and laboratory environment is replaced by a model based around the development of collaborative learning, integrated curriculum, and problem-based learning. The approach used within the BIMS enables the development and expression of a model in which the teaching spaces, support infrastructure, subject content, teaching methods, and student learning environments are integrated. The studio-based approach to teaching IT in the BIMS program at Monash University was commenced in semester 1, 2000. However, it was not until semester 2 that the purpose built studio space was ready for occupation.

In discussing the Studio-based teaching model adopted by Monash University Carbone and Sheard (2002) suggest that when constructing new learning environments four aspects of future learning environments need to be considered:

- the physical space,
- the teaching approach,
- the assessment method and
- the IT facilities provided

Curriculum. One of the features of the studio-based approach in the BIMS course is the integration of the core subjects at each level in the degree. Planning and development workshops specifically designed for the BIMS staff have enabled the development of, and constant focus on, an integrated curriculum between the core subjects (Carbone and Sheard, 2002). Complementing the integrated curriculum is the use of a problem-based learning approach to the content in the studio subject. In the studio subject, students have the opportunity to develop strategies, cooperate,

collaborate, be individual, and acquire or develop the required skills to develop a system (Carbone and Sheard, 2002).

Assessment. Within the BIMS course, assessment involves presentation of a portfolio. The students undertake core studio work in collaborative groups, where the students gain skills in collaboration, communication, and context specific skills (Carbone and Sheard, 2002). Selections of these items are designated as mandatory and are required in a student's portfolio. Other items for the portfolio are ones that the students select themselves. During the semester students collect and correlate items that reflect what they have been learning, portray their chosen area of expertise and their development as a group member. The portfolio is assessed on at least two occasions throughout the year by tutors, BIMS academic staff and where possible, members of the profession or colleagues from other academic environments. A group oral presentation to a panel of examiners is also part of the studio subject's assessment. In addition to the examiner's marks, each student in the group allocates marks to each of the group's members for collaboration, co-operation, being a team player and being responsible within the group (Carbone and Sheard, 2002).

A survey of students at the end of first semester revealed some useful information for the teaching team to use in planning and implementation for semester two. However, because the purpose built studio space was not built until semester 2, the data was of little value in assessing the impacts of the studio-based model. A second survey was conducted towards the end of semester two, 2000. This survey focused on the students' perception of the teaching and learning approach, and the physical environment (spaces and facilities), as compared to the traditional university teaching and learning approach. A preliminary examination of the data indicated that the students prefer the studio-based approach to learning IT than the more traditional methods, and that they see the physical environment as one that is preparing them for the professional environment they will find themselves at the end of the course. In addition, the emphasis placed on collaboration during the year seems to be beneficial to the students as reflected in both the comments contained in the survey and their portfolios (Carbone and Sheard, 2002).

7 Teaching and learning programming – what do we know already?

7.1 - about student perspectives/experiences?

7.1.1 Gender:

Carter and Jenkins (1999) found that when extra tutorial classes were offered for students who approached the staff and asked for additional support, significantly more females attended even though they made up a smaller proportion of the total student group. Their study investigated students' attitudes and approaches to the learning of programming so that they might understand why mainly women attended the classes. They found differences in the way students choose (or are conditioned) to study. Previous studies had highlighted that females tend to lack confidence in the domain of computer science (e.g. Scragg and Smith, 1998; Spender, 1995), and also underestimate their abilities (e.g. Bernstein, 1991; Beyer and Bowden, 1990; Haller and Fossum, 1998). They also noted previous research that suggested differences in *motivation* between genders or how genders approach learning/learning styles. For example,

- females tend to stamp out difficulties before they are a problem
- females perform better in participative approaches
- females are more likely to approach staff for help (males prefer bulletin boards/email)

7.1.2 Variation in experiences:

The central question addressed in Booth's 1992 research was:

'What does it mean and what does it take to learn computer programming?'

A group of computer science and computer engineering undergraduates were followed for a half-year while they took an introductory course in programming (Booth, 1993). The research is in the phenomenographic tradition – i.e. the central phenomena of programming were analysed in terms of qualitatively distinct conceptions identified among the students. Programming is seen as a complex learning activity involving the development of interphenomenal and intraphenomenal relationships (Booth, 1993).

Booth (1993; 2001b) talks of 'framework constituents' which were seen as being essential for the outcome of programming studies, but are not thematised in instruction. These included the computer, the nature of programming, the nature of programming languages and what it takes to learn to program. These three 'framework constituents' can be seen within a phenomenographic perspective as three distinct aspects of the experience of learning to program, and capabilities to experience them in one way or another, or in a number of ways, can be seen as supporting or hindering the quality of learning (Booth, 2001b).

The fundamental phenomenon that was studied in the original research is ***how students experience programming***. Three qualitatively distinct orientations were seen:

1. towards the computer;
2. towards the problem that the programming activity is intended to solve; and
3. towards the product that would thereby be devised.

The second phenomenon which leads on from the nature of programming is the nature of programming languages. Four qualitatively distinct ***ways of experiencing programming languages*** were found:

1. as a utility program inherent in the computer system with certain properties such as speed;
2. as a code of which programs are built;
3. as a medium of expression which enables the programmer to express an idea or a solution in a way that can be effected by the computer; and
4. as a means of communication between the parts of a programming system such as programmer, computer, operating system and user.

The third framework constituent that was considered is what it means to learn to program. Four ways of understanding ***what it means to learn to program*** were seen:

1. The least complex is that learning to program is learning a programming language, in which focus is on learning the features and the details of one or more programming languages;

2. Learning to write programs in a programming language in which making use of techniques and special features is in focus and
3. Learning to solve problems in the form of programs, where focus is rather on analysing a problem so that a program can be written;
4. Learning to program was seen as becoming part of the programming community, focusing on producing programs that solve problems in collaboration with other programmers, or for someone else.

Booth's (1997) focus on programming students revealed variation in:

1. How students were experiencing the concept of ***recursion***.

   The research revealed three different understandings: recursion as programming construct in SML; as a means of bringing about repetition in SML, and; as self-reference.

Often those with the less developed understanding, did better in exams because of rote memorisation, whereas those students with better understanding produced seriously flawed programs.

2. The research revealed a variation in ***approaches to writing*** programs (see p. 153-4)

   Interpretive approaches*:*

   - structural approach (principle focus is on the structure of the problem in its own domain)
   - operations approach (focus on what the program is going to have to do)

   Opportunistic approaches*:*

   - constructual approach (focus on constructs and elements identified from current repertoire that might be used to make up the program)
   - expedient approach (where an existing program is taken up because of some clue in the problem, followed by an attempt to adapt it to the constraints of the problem at hand.)

> *What the individual teacher has to do is design the teaching situation in such a way that the variation in approaches to tackling tasks in just that area is revealed, both to the teacher who thereby gains further insight into the experience of the learner, and to the learners who thereby gain insight not only into the range of possibilities but also that a range is actually possible.*
> (Booth, 1997, p. 146)

The main tool for revealing variation is through focussed student discussion. It is not enough just to work in pairs (as tasks get divided and there is usually minimal discourse). Students need tasks which demand discusson in larger groups at all stages of work.

7.1.3 Expectations and preconceptions:

At the start of the academic year, when the new undergraduates arrive, teachers have certain expectations about them, about their understanding of certain concepts and the ways in which they will approach their studies. We don't always know, particularly in the current climate of the constantly changing school curriculum, what they initially expect from us, and what past experiences they are drawing upon to form these expectations. The students undoubtedly hold a variety of preconceptions about the courses they have chosen. Students choose computing modules based upon these preconceptions (Carter, 2001).

Booth (2001b) also places emphasis on the different expectations students bring to their learning context and how it influences their 'learning trajectory'. Students enter the university not only with a history but also with an expectation of the immediate and long-term future, with a starting point ('where they have been') and an intended and potential learning trajectory ('where they are going').

7.1.4 Culture:

(Booth, 2001b) extends her earlier research by relating it to a socio-cultural perspective.

In a wider socio-cultural framework, Booth's (1992; 1993) question is transformed from '*What does it mean and what does it take to learn computer programming?'* to something like:

> *What does it mean and what does it take to enter the culture of computer programming?*

For example;

> *When the new student of computer science and engineering arrives at the University, all is new. Non-technical aspects of the nature of programming, the nature of programming languages, what it means to learn to program, even of the computer itself, are features of a cultural context which are largely taken for granted by teachers - old-timers within the culture. The new-comer, the student, however, has to make what sense she or he can of these features.*
> (Booth, 2001b, p. 1)

Booth's (2001b) study describes the variation in ways new computer science students experience the culture they are meeting when they enter the world of computer science studies, and considers instructional implications.

'Datalogy' and 'Datalogical' are terms used by Booth to cover the field that might refer to and include computer programming, computing science, and computing technology, in all its forms (Booth, 2001b). She describes three 'datalogical' cultures:

an *academic datalogical culture;*
a *professional datalogical culture* and
an *informal datalogical culture* - outside academia and industry

These three identified cultures are '… qualitatively distinct ways of relating to the computer and all that is associated with making it work…' (Booth, 2001b, p. 9). i.e. 'datalogical identities'. How students experience programming, and what they think it is, can be seen as underlying the identity as programmer they are on their way to becoming.

Within a socio-cultural framework of communities of practice (based on Etienne Wenger, 1998) and culture, the different ways of experiencing the framework constituents equate to three qualitatively distinct forms of datalogical identity with which the newcomer to computing science enters the university. The relationship between the students' datalogical identity (the way the student relates to computers) and the academic datalogical culture (the way the people around them within the academic institution relate to computers) will influence their learning trajectories.

Learning is seen as 'a trajectory of identity in a community of practice' (Booth, 2001b, p. 12).

Booth (2001b) asserts that phenomenography is helpful in understanding '…the learning trajectories that students might be on, and why certain trajectories, with associated cultural webs of significance, might lead to more successful studies, and hence learning outcomes, than others.' (Booth, 2001b p. 18).

7.2 - about what helps students learn?

7.2.1 Nature of assessment tasks:

> *Ultimately lecturers teaching programming would expect their students to be able to design, implement and test a relatively complex piece of software. It is a common belief that the larger and more complex the code students write, the better programmers they will be.....However, if students are asked to write complex pieces of code at too early a stage they can be pushed into adopting a poor learning tendency* (Carbone et al., 2000)

Carbone et al., (2000) look at three poor learning tendencies which arise from the nature of programming courses' tasks:

- superficial attention; This involves skimming over a communication, with no attempt to actively process the task in order to generate personal meaning
- impulsive attention; Some parts of a communication are attended to but others are overlooked. For example, the learner may focus on an interesting example and ignore a major point
- staying stuck; Lack of any strategy to cope with getting stuck except to call for help. No attempt to return to the instructions, reflect on the strategy selected, analyse what has been done so far or consider alternative approaches (Carbone et al., 2000).

They (Carbone et al., 2000) suggest the following areas should be considered when designing tasks. Improvements that can be made to tasks to minimise superficial attention include:

- *Not always coding:* Often students are required to write lines of code, and very rarely are students required to do alternative activities. Getting

students to diagrammatically present material often highlights misconceptions that can be addressed immediately. Including tasks that require tracing code, or answering a series of questions, can be used as alternatives to purely writing code.

- *Rewards for understanding not completing:* If students were aware that understanding was rewarded, and not copious amount of code, they might be less likely to take a crude approach to completing the task.

- *Outline a method of attack:* Without a design students can wander from the intended pathway and ultimately reach a point where the only source of help is seen to be copying extracts of code provided.

- *Smaller coding questions:* Introduce questions that don't consume too much of the students' time, so they don't feel pressured into copying straight from the notes.

Improvements that can be made to tasks to minimise impulsive attention include:

- *Emphasise the key point:* Usually there are many ways to code a solution to a problem. If the important points are emphasised in the tasks, through the task's aim and the type of question, impulsive attention might *be minimised*

- *Provide adequate resources for the introduction of unfamiliar material:* Sometimes subjects are so tightly structured that it's not possible to cover everything in lectures. As a consequence, many lecturers introduce new material in the laboratory and tutorial exercises. Often it is not possible to remove material from the subject, leaving some of the material introduced for the first time outside of that environment. While the idea of introducing new material is fine, the resources needed to help students' understanding are usually inadequate, rather than carefully planned.

Improvements that can be made to tasks to minimise staying stuck include:

- *Tactics on how to start with graded helps:* Challenge a student first, don't explain everything

- *Provide useful references and resources:* Often the only resources students are aware of are their text book, their lecture notes and tutor.

- *Provide guidelines to writing and testing code in manageable chunks:* Include debugging strategies. Guides to writing code should be provided rather than providing the code. Build a program in stages, for example, if part of the problem requires file input and output, students could write a small program to ensure they understand how to do that part.

Carbone et al. (2000) also look at features of tasks that lead to the poor learning behaviours of non-retrieval, lack of internal reflective thinking and lack of external reflective thinking. Suggested improvements to tasks in order to improve the following processing habits of students explored in the study:

1. ***non-retrieval*** when no attempt is made to retrieve one's own existing views/understandings relevant to the knowledge being presented.
   To improve the tasks:
   - familiarity (with new material, referral to earlier work should be made)
   - reinforcement by repetition
   - retrieve existing understanding

2. ***lack of internal reflective thinking*** (about subject content)
   To improve the tasks:
   - tie the work into the "big ideas" of the lesson
   - build on previous work
   - extract the links

3. ***lack of external reflective thinking*** (about linking subject with outside world or other subjects)
   To improve the tasks:

- when introducing new concepts, work out why there is a need for the new concept and how it relates to external matters

8 Summary

> *Changing an approach to teaching requires first the knowledge that other approaches are possible; secondly it requires reflective practitioners. However it also requires evaluation and evidences of the success of any given approach and there is little of this work in the literature, and much less which is comparable across institutions and diverse student populations* (Fincher, 1999, p. 12a4-5).

When reviewing the literature on teaching and learning introductory computer programming, it is easy to find examples of teaching approaches adopted for specific computer classes at specific institutions. It is perhaps surprising, however, that there is a dearth of empirical evidence as to the effects of implemented changes to curriculum and teaching approaches, particularly in relation to impacts on failure rates – the very thing which seems to be driving the need for change. It is also often difficult to unearth the rationale which informs the choice of WHY the subject is taught in a particular fashion (Fincher, 1999). As Greening and Kay suggest in their introduction to the special issue of CSE focusing on constructivism, practice has not necessarily followed theoretical underpinnings (Greening and Kay, 2001). Rather than an *ad hoc* approach to designing and implementing change, a rigorous research agenda is likely necessary in order to develop changes based on significant theoretical understanding.

In the quest to improve approaches to teaching programming it is also significant that there has not been much work on *how* students learn to program. Booth (Booth, 1993) conducted early research into this question by utilising a phenomenographic approach to obtain empirical evidence of variation in experiences. Subsequent research by Booth (e.g. Booth, 1993; Booth, 1997; Booth, 2001b) and Carter (e.g. Carter, 2001; Carter and Jenkins, 1999) has further illuminated the variation in experiences that potentially affect students' learning of programming. It appears that a focus on the 'teaching' rather than the 'learning' of programming has limitations. Taylor et al. (2001, p. 19) draw an analogy between a focus on a teaching perspective as akin to designing software from a designer's perspective rather than the client's perspective.

Essentially, students of first year programming subjects experience programming in a variety of ways. Whether this is due to gender effects, their past experiences, their expectations and preconceptions, or a range of socio-cultural effects, surely improvements in teaching programming would more easily be implemented with an improved understanding of where students are 'coming from' in their approach to learning how to program. It is not the demographics (e.g. sex, socio-economic status, ethnic origin etc), of the very diverse population constituting first year programming students that will determine the appropriate changes in curriculum and teaching practices. Rather, it is the exploration and revelation of the range of students' conceptions of the phenomenon of programming and learning to program that is necessary. Once the variation is revealed, it becomes possible for both the student and the teacher to not only recognise that different conceptions exist, but also that different conceptions and thus approaches might be adopted.

The following section outlines some research recommendations that have come out of the review of the literature on teaching and learning programming.

9 Research Recommendations arising from the literature

- 'The key must surely be in the questions that we ask' (Fincher, 1999).

- 'Very little attention has been paid to the rationale which informs the choice of why we teach the subject in a particular fashion' (Fincher, 1999, p. 12a4-1).

- Carter and Jenkins (1999) suggest that investigating/understanding the ways in which students learn to program will lead to improvements.

- 'Issues of how the course is taught and who the students are influence the outcome, rather than being simply a matter of programming language X vs programming language Y' (McCracken, 2001).

  Research into the teaching and learning introductory programming needs to investigate what these broader issues are.

- 'To efficiently teach computer programming skills is difficult. The kinds of assessment that instructors use throughout their courses must provide appropriate information for understanding students' processes of developing programming skill' (McCracken, 2001, p. 134).

- '…requires that research looks deeper than merely evaluating implementations, deep enough to examine what changes in teaching practice reveal about underlying issues such as concept acquisition, development of skills and expertise, sources of misconception and superstition, learning processes, the roles of different types of interaction between teachers, students, and materials, and so on.' (Daniels et al., 1998).

10 Towards a research proposal

This paper has provided a preliminary background into some of the current developments in the research and literature on teaching and learning introductory programming. Gaps in the research have been identified and some suggestions for further research are briefly introduced in section 9. As we have uncovered the current issues and gaps, a series of questions has begun to form that will influence the direction of our future research in the area. The focus of these questions is on the experiences of the students learning to program, that is, they focus on the learners' perspective. We will reflect on these preliminary questions, develop and re-work them to form the basis of a research proposal aimed at improving the teaching (and learning) of introductory programming at university.

At this stage a series of preliminary, general questions have been divided into a number of levels of priority, and are included in Table 2 below.

Table 2
Preliminary research questions

PRIORITY LEVEL	RESEARCH QUESTION
Level 1	How do students learn to program?
Level 2 (a)	How do students go about writing a program?
	How do students experience learning to write a program?
Level 2 (b)	What do students think a program is?
	What are the different ways students see programming languages?
	How do students experience learning programming languages?
Level 3	What do students see as effective means of help in learning to program?
Level 4	Are there differences in how males/females learn to program?
	What is the relationship between learning a programming language and learning to program?
	How are institutions/people tackling failure rates?

In order to answer at least some of these questions we will seek to investigate the experiences of a range of students who have undertaken, or are undertaking, introductory programming units – specifically itb410, itb411, and itb107 – at Queensland University of Technology. Students with a range of programming capabilities will be involved, in order to capture the differences in ways of tackling the task of learning to program between good programmers and those who have failed or are failing the subject. The research team will consult with faculty teaching staff who have been involved in the programming units and the faculty teaching support officer, in the selection of participants.

Semi-structured interviews will be used to uncover the variation in ways students approach learning to program. Interview questions will be piloted with 3 participants of varying capability. The pilot process will help clarify the research questions. For instance, it is anticipated that the pilot process will illuminate subtle, yet important, distinctions between questions such as ‘How do students learn to program’ and ‘How do students experience learning to write a program?’ This will influence the direction of the final research project. It will also serve to assist with finalising the wording of the questions and highlight any further areas which will provide data to answer the research questions.

It is anticipated that there may be implications associated with the type of person who conducts the interviews. For example, potentially different results may arise from a non-programmer interviewer compared to those obtained by an interviewer who is a programmer. These potential differences will need to be considered and analysed.

It is a specific design feature of the questions in phenomenographic research that they should be broad enough to obtain meaningful responses in relation to the aim without forcing a particular structure, or way or responding upon the participant. The questions, therefore, will be worded in such a way that they are open enough to '…allow the subjects to express their own way of structuring the aspects of reality they are relating to' (Johansson *et al.,* 1985, p. 252). Each question serves as an 'opening', from which the interviewer will develop a trail of further questions in order to achieve a mutual understanding of the theme in focus. The interviewer can assist in the process of developing a shared understanding by: confirming meaning by returning to particular statements, following up unexpected threads in the discussion, attempting to unblock unexpected obstacles and closing interviews by enabling the student to put their own questions and other points of view (Booth, 2001a).

Table 3 shows the range of potential interview questions to be considered and possibly piloted in the first instance. The Table is not complete, but is designed to show how each interview question relates to each particular research question.

The primary research outcome for our study will be a mapping of the variation in experience of learning to program. The particular descriptive focus inherent in phenomenography has produced two distinct presentational outcomes of any phenomenographic study: categories of description, and an outcome space. Categories of description represent each 'conception' or way of *experiencing* or *being aware of* learning to program. Each category highlights the critical difference in meaning and structure between conceptions. The outcome space is a diagrammatic representation of the logical relationships between conceptions as described in the categories of description. In other words, the outcome space captures the essential experience at the collective level (Booth, 2001a). The focus on uncovering the structural framework is fundamental to phenomenographic research, and is the factor which will provide the unique insights into the experiences of learning to program.

Table 3
Preliminary research questions and potential interview questions

LEVEL	RESEARCH QUESTION	INTERVIEW SCHEDULE - POTENTIAL QUESTION
Level 1	How do students learn to program?	How do you go about learning to program?
		What do you see as the major 'things involved' in learning to program? (McCracken et al., 2001)
		Is there a particular order in which to learn these 'things' that you feel would help you in your learning progress?
Level 2 (a)	How do students go about writing a program?	*Potentially set a Task:* *e.g. present basic programming problem and have student describe step by step how they approach the problem.*
	How do students experience learning to write a program?	Can you describe how you went about learning to write programs?
Level 2 (b)	What do students think a program is?	How would you describe a computer program?
	What are the different ways students see programming languages?	*Develop from Booth's (1992) interview questions/results.*
	How do students experience learning programming languages?	How do you go about learning to program?
Level 3	What do students see as effective means of help in learning to program?	What types of assessment do you feel would help you to better learn programming?
		Did the methods of assessment in the programming unit(s) you have done help you in [understanding] / [learning] programming?
		What was it about the way the programming unit(s) you have done was/were taught that most helped you learn to program?
		What was it about the way the programming unit(s) you have done was/were taught that least helped you?
		Can you suggest any improvements in the way the programming unit(s) you have done was/were taught that would improve your understanding of how to program?
Level 4	Are there differences in how males/females learn to program?	
	What is the relationship between learning a programming language and learning to program?	
	How are institutions/people tackling failure rates?	

11 References

Astrachan, Owen and Reed, David (1995) *AAA and CS1: The Applied Apprenticeship Approach to CS 1.* In Proceedings of the ACM SIGCSE Symposium, 1995.

Barg, M., Fekete, A., Greening, T., Hollands, O., Kay, J., Kingston, J. H. and Crawford, K. (2000) Problem-based learning for Foundation Computer Science Courses. *Computer Science Education, v.10,* no.2, pp. 109-128.

Barnard, A, McCosker, H and Gerber, R. (1999) Phenomenography: a qualitative research approach for exploring understanding in health care, *Qualitative Health Research*, vol. 9, no, 2, pp. 212-226.

Barnes, David J., Fincher, Sally and Thompson, Simon (1997) Introductory Problem Solving in Computer Science. In Daughton, Goretti and Magee, Patricia, (Eds), *5th Annual Conference on the Teaching of Computing*, Centre for Teaching Computing, Dublin City University, Dublin 9, Ireland, August 1997, pp 36-39.

Ben-Ari, M. (1998) Constructivism in Computer Science Education. In *The Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, pp. 257 -261.

Bernstein, D. (1991) Comfort and experience with computing: are they the same for women and men? *SIGCSE Bulletin 23* (1991) pp. 57 –60.

Beyer, S., and Bowden, E. (1990) Gender differences in the accuracy of self-evaluations of performance. *Journal of personality and social psychology v. 5,* no. 3 pp 960-970.

Biggs, J. (1999) *Teaching for quality outcomes at university: What the student does.* Society for Research into Higher Education and Open University Press, Buckingham, UK.

Booth, S. (1992) *Learning to program: a phenomenographic perspective*, Acta Universitatis Gothoburgensis, Goteborg.

Booth, S. (1993) A Study of Learning to Program From an Experiential Perspective. *Computers in Human Behavior, v.9,* pp. 185-202.

Booth, S. (1997) On Phenomenography, Learning and Teaching. *Higher Education Research and Development, v.16,* no.2, pp. 135-158.

Booth, S. (2001a) Learning Computer Science and Engineering in Context. *Computer Science Education, v.11,* no.3, pp. 169-188.

Booth, S. (2001b) Learning to program as entering the datalogical culture: a phenomenographic exploration. In *9th European Conference for Research on Learning and Instruction (EARLI),* Fribourg, Switzerland.

Bornat, R. *Programming from First Principles.* Prentice Hall International, 1987.

Bowden J. & Marton, F. (1998). *The University of learning: Beyond quality and competence in higher education.* London: Kogan Page.

Bowden, J. & Walsh, E. (Eds) (2000). *Phenomenography*. Melbourne: RMIT Press.

Bruce, C. (1997) *The Seven Faces of Information Literacy,* Auslib Press, Adelaide.

Bruce, C. (1999) Workplace experiences of information literacy*, International Journal of Information Management*, vol. 19, no 1, pp. 33-48.

Bruce, C. and Gerber, R. (1995) Phenomenographic Research: An Annotated Bibliography. Centre for Applied Environmental and Social Education Research. Occasional Paper 95.2 Available online at http://sky.fit.qut.edu.au/~bruce/anabib/title.html

Bruce, C and Pham, B. (2001) Investigating ways of seeing IT research: a tool for facilitating effective research partnerships. Paper to be presented at the Higher Education Research and Development Society of Australasia Conference, July 2001, Newcastle, New South Wales.

Bruce, C., Pham, B. and Stoodley, I. (2002a). FINAL REPORT. The Collective Consciousness of Information Technology Research: The Significance and Value of Research Projects A. The Views of IT Researchers. QUT, FIT Technical Report Series.

Bruce, C., Pham, B. and Stoodley, I. (2002b). FINAL REPORT. The Collective Consciousness of Information Technology Research: The Significance and Value of Research Projects B. The Views of Industry Professionals, QUT, FIT Technical Report Series.

Carbone, A., Hurst, J., Mitchell, I. and Gunstone, D. (2000) Principles for Designing Programming Exercises to Minimise Poor Learning Behaviours in Students. In *Proceedings of the conference on Australasian computing education conference* ACM, Melbourne Australia, pp. 26-33.

Carbone, A. and Sheard, J. (2002) A Studio-based Teaching and Learning Model in IT: What do First Year Students think? *Seventh Annual Conference on Innovation and Technology in Computer Science Education*, University of Aarhus, Denmark, June 24-26, 2002.

Carter, J. (2001) What they think - students' preconceptions of computing. In *International Conference on Engineering Education,* Oslo, Norway.

Carter, J. and Jenkins, T. (1999) Gender and programming: What's going on? In *Proceedings of the 1999 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'99*ACM SIGCSE/SIGCUE, pp. 1-4.

Cope, C (2000) *Educationally critical aspects of the experience of learning about the concept of information systems*. PhD Thesis, La Trobe University, Australia.

Cope, C, Horan, P, and Garner, M (1997) Conceptions of an information system and their use in teaching about IS, *Informing Science*, vol. 1, no. 1, pp. 9-22.

Daniels, M., Petrie, M. and Bergland, A. (1998) Building a Rigorous Research Agenda into Changes to Teaching. *3rd ACM Australasian Computer Science Education Conference*, Brisbane, Australia, July 98.

Duit, R. (1999). Conceptual change approaches in science education. In W. Schnotz, S. Vosniadou, & M. Carretero (Eds.), *New Perspectives on Conceptual Change* (pp. 263-282). Oxford: Pergamon.

Ellis, A., Lowder, J., Robinson, J., Hagan, D., Doube, W., Tucker, S., Sheard, J. and Carbone, A. (1999) Collaborative strategy for developing shared Java teaching resources to support first year programming. In *Proceedings of the 1999 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'99*ACM SIGCSE/SIGCUE, Cracow, Poland.

Fincher, S. (1999) What are we doing when we teach programming? In *29th Annual Frontiers in Education Conference: 'Designing the Future of Science and Engineering Education'*IEEE Education Society; IEEE Computer Society; ASEE Educational Research and Methods Division, San Juan, Puerto Rico.

Fowler, L., Armarego, J. and Allen, M. (2001) CASE Tools: Constructivism and its Application to Learning and Usability of Software Engineering Tools. *Computer Science Education, v.11,* no.3, pp. 261-272.

Gerber, R., Buzer, S., Worth, C. and Bruce, C. (1992) Is GIS a GIS? Or coming to an experiential understanding of GIS. In Gerber, R. (Ed) *AURISA 92 Power to the People: the Community, Information and Management, Proceedings of the Twentieth Annual International Conference of the Australasian Urban and Regional Information Systems Incorporated*, AURISA, pp. 18-25.

Gorriz, C.M. and Medina, C. (2000). Engaging girls with computers through software games. *Communications of the ACM*, *43*, pp. 42 - 49.

Gottfried, B. S. (1997) Teaching computer programming effectively using active learning. In *Proceedings of the 1997 ASEE Annual Conference* Milwaukee, WI, USA.

Greening, T. and Kay, J. (2001) Editorial. *Computer Science Education, v.11,* no.3, pp. 167-168.

Hagan, D., Sheard, J. and Macdonald, I. (1997) Monitoring and evaluating a redesigned first year programming course. In *Proceedings of the 1997 Conference on Integrating Technology into Computer Science Education, ITiCSE* SIGCUE, Uppsala, Sweden.

Haller, S., and Fossum, T., (1998) Retaining women in CS with accessible role models. *Proceedings of SIGCSE'98 conference*, Atlanta, 1998.

Hasselgren, B, Nordieng, T and Osterlund, A. (webmasters) (2001) *The Land of Phenomenography.* Website [online]. Available: http://www.ped.gu.se/biorn/phen.home.html

Johnson, D.W., Johnson, R.T., & Smith, K.A. (1991). *Active learning: Cooperation in the college classroom.* Interaction Book Company.

Kay, J., Barg, M., Fekete, A., Greening, T., Hollands, O., Kingston, J. H. and Crawford, K. (2000) Problem-Based Learning for Foundation Computer Science Courses. *Computer Science Education, v.10,* no.2, pp. 109 (20p).

Klaus, H. (2000) Understanding scholarly and professional communication: thesauri and database searching. In C Bruce and P Candy (eds) *Information Literacy Around the World: advances in programs and research,* Charles Sturt University Press, Riverina.

Klaus, H. and Bruce, C. (1997) Phenomenographic Research, An Annotated Bibliography. 1997 Supplement. Available online at http://www.fit.qut.edu.au/InfoSys/ism/Bibliography/index.html

Klaus, H. and Gable, G. (2000). Senior managers' understandings of knowledge management in the context of enterprise systems. *6th Americas Conference on Information Systems, Long Beach, California, August 10th - 13th, 2000*: 981-987

Lidtke, D. K. and Zhou, H. H. (1999) New approach to an introduction to computer science. In *29th Annual Frontiers in Education Conference: 'Designing the Future of Science and Engineering Education '*IEEE Education Society; IEEE Computer Society; ASEE Educational Research and Methods Division, San Juan, Puerto Rico.

Marton, F. (1998) Phenomenography; exploring different conception of reality, in *Qualitative Approaches to Evaluation in Education: the Silent Revolution,* ed. David Fetterman, Praeger, New York, pp. 176-205.

Marton, F. and Booth, S. (1997). *Learning and Awareness*, Lawrence Erlbaum, New Jersey.

Marton, F. and Neuman, D. (1989) Constructivism and constitutionalism. Some implications for elementary mathematics education. *Scandinavian Journal of Psychological Research, v.33,* no.1, pp. 34-46.

Marton, F. and Säljö, R. (1976) On qualitative differences in learning. I. Outcome and process. *British Journal of Educational Psychology, v. 46*, pp. 4-11.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I. and Wilusz, T. (2001) A multi-national, multi-institutional study of assessment of programming skills of first-year CS students: Report by the ITiCSE 2001 Working Group on Assessment of Programming Skills of First-year CS Students. In *ITiCSE2001*Canterbury, UK.

McKenna, P. (2000) Transparent and opaque boxes: do women and men have different computer programming psychologies and styles? *Computer Science Education, v.35,* pp. 37 - 49.

McKenna, P. (2001) Programmers: concrete women and abstract men? *Journal of Computer Assisted Learning, v.17,* no.4, pp. 386-95.

Davis, J. (2001) Conceptual Change. *In Emerging Perspectives on Learning, Teaching and Technology* (Ed.) M. Orey (2001-2002). URL http://itstudio.coe.uga.edu/ebook/htm accessed 6 December 2001.

Pozo, J.I. (1997). Conceptual change as a process of restructuring, explication and hierarchical integration. Paper presented at the Seventh European Conference for Research on Learning and Instruction, August 26-30, 1997, Athens, Greece.

Ramsden, Paul (1988) Studying learning: Improving teaching. In P. Ramsden (Ed.) *Improving Learning: new perspectives*. Kogan Page. London, pp. 13-31.

Sandberg, J. (1994) *Human Competence at Work: an interpretative approach*. Goteborg, Bas.

Scragg, G., and Smith, J., A study of barriers to women in undergraduate computer science. P*roceedings of SIGCSE'98 conference*, Atlanta, 1998.

Spender, D. (1995) *Nattering on the net: women, power and cyberspace*. Spinifex North Melbourne.

Stein, L. A. (1999) Challenging the Computational Metaphor: Implications for How We Think. *Cybernetics and Systems, v.30,* no.6, pp.1 - 35.

Stewart, G. and Klaus, H. (2000) Characterising the Business-IT Executive relationship. *Proceedings of the 6th Americas Conference on Information Systems, Long Beach, California, August 10th - 13th, 2000*:1913-1918.

Swensson, L. (1977) On qualitative differences in learning. III – Study skill and learning. *British Journal of Educational Psychology, v. 47*, 233-243.

Taylor, P., McWilliam, E., Burnett, B. and Thyer, N. (2001) *Teaching and Learning Improvement Partnership Project Final Report.* Faculty of Information Technology, Queensland University of Technology, Brisbane.

Trigwell, K. and Prosser, M. (1997) Towards an Understanding of Individual Acts of Teaching and Learning. *Higher Education Research and Development, v.16,* no.2, pp. 241-252.

Turkle, S. (1984). Women and computer programming: a different approach. *Technology Review*, pp. 49-50.

Tynjälä, P. (1998) Writing And Conceptual Change In University Studies. In *1998 European Writing Conference,* Poitiers, France.

Van Gorp, M. J. and Grissom, S. (2001) An empirical evaluation of using constructive classroom activities to teach introductory programming. *Computer Science Education, v.11,* no.3, pp. 247-60.

Vizcaino, A., Contreras, J., Favela, J. and Prieto, M. (2000) An Adaptive, Collaborative Environment to Develop Good Habits in Programming. In *Fifth International Conference on Intelligent Tutoring Systems (ITS 2000)* University of Québec, Montréal.

Williams, L. A. and Kessler, R. R. (2000a) Effects of 'pair-pressure' and 'pair-learning' on software engineering education. In *The 13th Conference on Software Engineering Education and Conference (CSEE and T 2000)* IEEE Computer Society, Austin, TX, USA, pp. 59 - 65.

Williams, L. A. and Kessler, R. R. (2000b) Experimenting with Industry's "Pair-Programming" Model in the Computer Science Classroom. *Journal on Software Engineering Education,* December 2000.

Wills, C. E., Deremer, D., McCauley, R. A. and Null, L. (1999) Studying the Use of Peer Learning in the Introductory Computer Science Curriculum. *Computer Science Education, v.9,* no.2, pp. 71- 79.

Authors

Christine Bruce
Christine Bruce, BA, Grad Dip Lib Sc, MEd(Res), PhD, is Associate Professor and Director of Teaching and Learning in the Faculty of Information Technology. Christine's research focuses on higher education teaching and learning. She has published extensively in the area of information literacy and postgraduate study and supervision. She uses phenomenographic and action research approaches for the development of learning models. Christine is presently working on the Collective Consciousness of IT Research and students' introductory experiences of learning to program.

Camille McMahon
Camille Mc Mahon, BA, DipEd, MEnvtSt is researching teaching and learning programming in the Faculty of Information Technology. She has methodological interests in action research and phenomenography. She is also interested in the use of information and communication technologies in developing countries.