

CONFERENCES IN RESEARCH AND PRACTICE IN  
INFORMATION TECHNOLOGY

VOLUME 66

# COMPUTING EDUCATION 2007

AUSTRALIAN COMPUTER SCIENCE COMMUNICATIONS, VOLUME 29, NUMBER 5.



AUSTRALIAN  
COMPUTER  
SOCIETY



CO<sub>mputing</sub>  
R<sub>esearch</sub>  
& E<sub>ducation</sub>



# COMPUTING EDUCATION 2007

Proceedings of the Ninth  
Australasian Computing Education Conference (ACE2007),  
Ballarat, Victoria, Australia,  
January 30 to February 2, 2007

Samuel Mann and Simon, Eds.

Volume 66 in the Conferences in Research and Practice in Information Technology Series.  
Published by the Australian Computer Society Inc.



Published in association with the ACM Digital Library.

**Proceedings of the Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, January 30 to February 2, 2007**

**Conferences in Research and Practice in Information Technology, Volume 66.**

Copyright ©2007, Australian Computer Society. Reproduction for academic, not-for profit purposes permitted provided the copyright text at the foot of the first page of each paper is included.

Editors:

**Samuel Mann**

Department of Information Technology  
Otago Polytechnic  
New Zealand  
Email: [smann@tekotago.ac.nz](mailto:smann@tekotago.ac.nz)

**Simon**

School of Design, Communication and Information Technology  
University of Newcastle  
Ourimbah Campus  
PO Box 127  
Ourimbah  
NSW 2258  
Email: [Simon@newcastle.edu.au](mailto:Simon@newcastle.edu.au)

Series Editors:

Vladimir Estivill-Castro, Griffith University, Queensland  
John F. Roddick, Flinders University, South Australia  
Simeon Simoff, University of Technology, Sydney, NSW  
[crpit@infoeng.flinders.edu.au](mailto:crpit@infoeng.flinders.edu.au)

Publisher: Australian Computer Society Inc.  
PO Box Q534, QVB Post Office  
Sydney 1230  
New South Wales  
Australia.

Conferences in Research and Practice in Information Technology, Volume 66.  
ISSN 1445-1336.  
ISBN 1-920-68247-3.

Printed, December 2006 by Flinders Press, PO Box 2100, Bedford Park, SA 5042, South Australia.  
Cover Design by Modern Planet Design, (08) 8340 1361.

The *Conferences in Research and Practice in Information Technology* series aims to disseminate the results of peer-reviewed research in all areas of Information Technology. Further details can be found at <http://crpit.com/>.



# Table of Contents

## Proceedings of the Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, January 30 to February 2, 2007

Preface .....	vii
Programme Committee and Review Panel.....	viii
Organising Committee .....	ix
CORE - Computing Research and Education .....	xi
ACSW Conferences and the Australian Computer Science Communications .....	xii
ACSW and ACE 2007 Sponsors .....	xv

## Panel

The Carrick Vision and Computing Education: Four Case Studies in Multi-institutional Collaboration <i>Angela Carbone, Michael de Raadt, Judy Kay, Raymond Lister, Andrew Litchfield, Richard Raban, Paul Roe, Daniel Santamaria, Judy Sheard, John Shepherd, Andrew Solomon and Richard Thomas</i>	3
---	---

## Contributed Papers

WAT - A Tool for Classifying Learning Activities from a Log File .....	11
<i>Jason Ceddia, Judy Sheard and Grant Tibbey</i>	
Why Teach Unix? .....	19
<i>Bernard Doyle and Raymond Lister</i>	
Software Development Marketplaces - Implications for Plagiarism .....	27
<i>Daryl D'Souza, Margaret Hamilton and Michael C. Harris</i>	
Measuring Improvement in Latent Semantic Analysis-Based Marking Systems: Using a Computer to Mark Questions about HTML .....	35
<i>Debra T. Haley, Pete Thomas, Anne De Roeck and Marian Petre</i>	
Peer Assessment Using Aropä .....	43
<i>John Hamer, Catherine Kell and Fiona Spence</i>	
Mobile Computing, Programming, Games and Online Communities: Changes to the Communicative Ecology of Design Students Through Mobile Computing: Part 2 .....	55
<i>Margaret Hamilton and Marsha Berry</i>	
Quantitative Peer Assessment: Can students be objective? .....	63
<i>Nicole Herbert</i>	
Student timesheets can aid in curriculum coordination .....	73
<i>Nicole Herbert and Zhong Wang</i>	
Difficulties experienced by students in maintaining object-oriented systems: an empirical study .....	81
<i>Amela Karahasanović and Richard C. Thomas</i>	

Learner Reflection in Student Self-assessment . . . . .	89
<i>Judy Kay, Lichao Li and Alan Fekete</i>	
Differing Ways that Computing Academics Understand Teaching . . . . .	97
<i>Raymond Lister, Anders Berglund, Ilona Box, Chris Cope, Arnold Pears, Chris Avram, Mat Bower, Angela Carbone, Bill Davey, Michael de Raadt, Bernard Doyle, Sue Fitzgerald, Linda Mannila, Cat Kutay, Mia Peltomäki, Judy Sheard, Simon, Ken Sutton, Des Traynor, Jodi Tutty and Anne Venables</i>	
A Maturity Model for Computing Education . . . . .	107
<i>Christof Lutteroth, Andrew Luxton-Reilly, Gillian Dobbie and John Hamer</i>	
Software engineering class eating its own tail . . . . .	115
<i>Samuel Mann and Lesley Smith</i>	
Review of Work Experience in a Bachelor of Information Technology . . . . .	125
<i>Joel W. Pauling and Peter Komisarczuk</i>	
Mandatory fields - a case study in the divergence between education and practice . . . . .	133
<i>Simon</i>	
Simple Eight Bit, Emulated Computers for Illustrating Computer Architecture Concepts and Providing a Starting Point for Student Designs . . . . .	141
<i>Timothy D. Stanley, Thanh Quach Xuan, Leslie Fife and Don Colton</i>	
Evaluation of a New Assesment Scheme for a Third-year Concurrency Course . . . . .	147
<i>Paul Strooper and Larissa Meinicke</i>	
Holistic assessment criteria - Applying SOLO to programming projects . . . . .	155
<i>Errol Thompson</i>	
Engaging Undergraduates in Discussions about Ethics in Computing . . . . .	163
<i>Brian R. von Kinsky, Jim Ivins and Susan J. Gribble</i>	
Decoding Doodles: Novice Programmers and Their Annotations . . . . .	171
<i>Jacqueline Whalley, Christine Prasad and P. K. Ajith Kumar</i>	
<b>Author Index . . . . .</b>	<b>179</b>

# Preface

Welcome to the Ninth Australasian Computer Education Conference 2007, held as part of Australasian Computer Science Week 2007 in Ballarat, Australia. This year's conference promises to continue the tradition of a high quality conference providing the opportunity for educators from all areas of computing to meet together to share experiences, and most importantly, computing education research.

The conference programme this year includes a keynote paper, two panel sessions and 20 paper presentations. The papers presented in these proceedings reflect leading edge developments in the field of computing education research. They were selected after an international call for papers resulting in 43 papers being received. Submissions came from Australia, Malaysia, New Zealand, Singapore, Sweden, the UK and the USA. Each paper was subject to a double blind refereeing process and 21 (48% including a paper to accompany one of the panels) were finally included for publication and presentation. The chairs are very grateful to all the members of the Programme Committee and the Review Panel who gave their time and expertise to this process.

We thank everyone involved in Australasian Computer Science Week for making this conference and publication possible, and we thank the Australasian Computing Education Executive for the opportunity given to us to chair this conference.

**Samuel Mann**

Otago Polytechnic, New Zealand

**Simon**

University of Newcastle, Australia

ACE 2007 Programme Chairs

January 2007

# Programme Committee and Review Panel

## Programme Committee

Samuel Mann, Otago Polytechnic, Otago, New Zealand  
Simon, University of Newcastle, Australia  
Alison Young, Unitec New Zealand, Auckland, New Zealand  
Raymond Lister, University of Technology Sydney, Australia

## Review Panel

Simon, University of Newcastle, Australia  
Tiru Arthanari, University of Auckland, New Zealand  
Anders Berglund, Uppsala University, Sweden  
Ilona Box, University of Western Sydney, Australia  
David Bremer, Otago Polytechnic, New Zealand  
Kay Bryant, Griffith University, Australia  
Tony Clear, Auckland University of Technology, New Zealand  
Mats Daniels, Uppsala University, Sweden  
Martin Dick, RMIT University, Australia  
Daryl D'Souza, RMIT University, Australia  
Kathy Egea, Central Queensland University, Australia  
Alan Fekete, University of Sydney, Australia  
Sally Fincher, University of Kent, United Kingdom  
Patricia Haden, Otago Polytechnic, New Zealand  
John Hamer, University of Auckland, New Zealand  
Margaret Hamilton, RMIT University, Australia  
John Hurst, Monash University, Australia  
Judy Kay, University of Sydney, Australia  
Xiaosong Li, Unitec New Zealand, New Zealand  
Raymond Lister, University of Technology Sydney, Australia  
Samuel Mann, Otago Polytechnic, New Zealand  
John Paynter, University of Auckland, New Zealand  
Judy Sheard, Monash University, Australia  
Lesley Smith, Otago Polytechnic, New Zealand  
Denise Tolhurst, University of New South Wales, Australia  
Brian Von Kinsky, Curtin University of Technology, Australia  
Jacqueline Whalley, Auckland University of Technology, New Zealand  
Alison Young, Unitec New Zealand, New Zealand

# Organising Committee

## Welcome

I would like to welcome you to the University of Ballarat and ACSW 2007.

Ballarat is one of Australia's largest inland cities with a population of 83,000, and is nestled peacefully in the heart of Victoria just over an hour from Melbourne. Ballarat is regarded as the birthplace of democracy in Australia and has one of the finest tourist attractions, namely Sovereign Hill.

The University of Ballarat is the third oldest tertiary institution in Australia. It is a medium-size University with about 22,000 students. The School of Information Technology and Mathematical Sciences of the University of Ballarat has 80 academic and general staff and includes the research Centre for Informatics and Applied Optimization (CIAO) and the Collaborative Centre for eHealth (CCeH).

ACSW 2007 includes the following conferences:

- Australasian Computer Science Conference (ACSC),
- Australasian Database Conference (ADC),
- Australasian Computer Education Conference (ACE),
- Computing: The Australian Theory Symposium (CATS),
- Asia-Pacific Conference of Conceptual Modelling (APCCM),
- Australasian User Interface Conference (AUIC),
- Australasian Symposium on Grid Computing and Research (AUSGRID),
- Australasian Workshop on Health Knowledge Management and Discovery (HKMD),
- Australasian Information Security Workshop: Privacy Enhancing Systems (AISW), and the
- Australasian Computing Doctoral Consortium (ACDC).

I thank all those who have worked to ensure the success of ACSW2007 including the Organizing Committee, the Conference Chairs and Programme Committees, the invited speakers and the delegates.



### Professor Sid Morris

Head, School of Information Technology and Mathematical Sciences  
University of Ballarat  
January, 2007

## General Chair

Professor Sid Morris, School of Information Technology and Mathematical Sciences, University of Ballarat

## Organising Committee Members

Ms Nadine Gass  
Mr Sasha Ivkovic  
Ms Kathleen Keogh  
Dr Liping Ma  
Dr Prabhu Manyem  
Mr Greg Simmons  
Ms Rosemary Torney  
Dr Chris Turville  
Ms Belinda Wallesz  
Dr David Yost



# CORE - Computing Research and Education

CORE welcomes all delegates to ACSW2007 in Ballarat.

ACSW, the Australasian Computer Science Week continues to grow with new conferences becoming entrenched in the week. As the premier annual Computer Science event in Australia and New Zealand, it provides an unparalleled opportunity for the wide community of Computer Science academics and researchers to meet, network, promote IT research and be exposed to the latest research in other areas of IT. The research presented at each conference is of the highest standard and essential for the growth and future of our region, in an ever more competitive world.

2006 has been a difficult year for IT and especially CORE's members. Falling student numbers have meant cutbacks in many of our universities. However, despite offshoring, industry is now calling for more graduates. We'll continue to work with ACS and industry bodies to try to convey this message to school leavers and their parents. We also have to continue to impress on industry the need to provide entry level positions so that young people can work towards the more senior positions which are so understaffed.

RQF is still hovering over Australian universities. In preparation, CORE has been part of a major exercise this year to rank ICT conferences and we'll contribute to the work of our sister organisation, ACPHIS in a similar journal ranking exercise.

Thank you all for your contributions in 2006 and we look forward to an exciting 2007.



**Jenny Edwards**

President, Computing Research and Education

January, 2007

# ACSW Conferences and the Australian Computer Science Communications

The Australasian Computer Science Week of conferences has been running in some form continuously since 1978. This makes it one of the longest running conferences in computer science. The proceedings of the week have been published as the *Australian Computer Science Communications* since 1979 (with the 1978 proceedings often referred to as *Volume 0*). Thus the sequence number of the Australasian Computer Science Conference is always one greater than the volume of the Communications. Below is a list of the conferences, their locations and hosts.

- 2009 (Proposed).** Communications Volume Number 31. Host and Venue - Victoria University, Wellington, New Zealand.
- 2008.** Volume 30. Host and Venue - University of Wollongong, NSW.
- 2007. Volume 29. Host and Venue - University of Ballarat, VIC.**
- 2006.** Volume 28. Host and Venue - University of Tasmania, TAS.
- 2005.** Volume 27. Host - University of Newcastle, NSW. APBC held separately from 2005.
- 2004.** Volume 26. Host and Venue - University of Otago, Dunedin, New Zealand. First running of APCCM.
- 2003.** Volume 25. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Adelaide Convention Centre, Adelaide, SA. First running of APBC. Incorporation of ACE. ACSAC held separately from 2003.
- 2002.** Volume 24. Host and Venue - Monash University, Melbourne, VIC.
- 2001.** Volume 23. Hosts - Bond University and Griffith University (Gold Coast). Venue - Gold Coast, QLD.
- 2000.** Volume 22. Hosts - Australian National University and University of Canberra. Venue - ANU, Canberra, ACT. First running of AUIC.
- 1999.** Volume 21. Host and Venue - University of Auckland, New Zealand.
- 1998.** Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.
- 1997.** Volume 19. Hosts - Macquarie University and University of Technology, Sydney. Venue - Sydney, NSW. ADC held with DASFAA (rather than ACSW) in 1997.
- 1996.** Volume 18. Host - University of Melbourne and RMIT University. Venue - Melbourne, Australia. CATS joins ACSW.
- 1995.** Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.
- 1994.** Volume 16. Host and Venue - University of Canterbury, Christchurch, New Zealand. CATS run for the first time separately in Sydney.
- 1993.** Volume 15. Hosts - Griffith University and Queensland University of Technology. Venue - Nathan, QLD.
- 1992.** Volume 14. Host and Venue - University of Tasmania, TAS. (ADC held separately at La Trobe University).
- 1991.** Volume 13. Host and Venue - University of New South Wales, NSW.
- 1990.** Volume 12. Host and Venue - Monash University, Melbourne, VIC. Joined by Database and Information Systems Conference which in 1992 became ADC (which stayed with ACSW) and ACIS (which now operates independently).
- 1989.** Volume 11. Host and Venue - University of Wollongong, NSW.
- 1988.** Volume 10. Host and Venue - University of Queensland, QLD.
- 1987.** Volume 9. Host and Venue - Deakin University, VIC.
- 1986.** Volume 8. Host and Venue - Australian National University, Canberra, ACT.
- 1985.** Volume 7. Hosts - University of Melbourne and Monash University. Venue - Melbourne, VIC.
- 1984.** Volume 6. Host and Venue - University of Adelaide, SA.
- 1983.** Volume 5. Host and Venue - University of Sydney, NSW.
- 1982.** Volume 4. Host and Venue - University of Western Australia, WA.
- 1981.** Volume 3. Host and Venue - University of Queensland, QLD.
- 1980.** Volume 2. Host and Venue - Australian National University, Canberra, ACT.
- 1979.** Volume 1. Host and Venue - University of Tasmania, TAS.
- 1978.** Volume 0. Host and Venue - University of New South Wales, NSW.



## Conference Acronyms

**ACE.** Australian/Australasian Conference on Computing Education.  
**ACSAC.** Asia-Pacific Computer Systems Architecture Conference (previously Australian Computer Architecture Conference (ACAC)).  
**ACSC.** Australian/Australasian Computer Science Conference.  
**ACSW.** Australian/Australasian Computer Science Week.  
**ADC.** Australian/Australasian Database Conference.  
**AISW.** Australasian Information Security Workshop.  
**APBC.** Asia-Pacific Bioinformatics Conference.  
**APCCM.** Asia-Pacific Conference on Conceptual Modelling.  
**AUIC.** Australian/Australasian User Interface Conference.  
**AusGrid.** Australasian Workshop on Grid Computing and e-Research.  
**CATS.** Computing - The Australian/Australasian Theory Symposium.

Note that various name changes have occurred, most notably the change of the names of conferences to reflect a wider geographical area.



## ACSW and ACE 2007 Sponsors

We wish to thank the following sponsors for their contribution towards this conference. For an up-to-date overview of sponsors of ACSW 2007 and ACE 2007, please see <http://www.ballarat.edu.au/acsw/>.



University of Ballarat, Australia



AUSTRALIAN  
COMPUTER  
SOCIETY

Australian Computer Society

CO<sub>mputing</sub>  
R<sub>esearch</sub>  
& E<sub>ducation</sub>

CORE - Computing Research and Education



Information Technology



School of Design, Communication and Information Technology



PANEL



# The Carrick Vision and Computing Education: Four Case Studies in Multi-institutional Collaboration

## Angela Carbone

Faculty of IT, Monash Uni., Australia  
Angela.Carbone@infotech.  
monash.edu.au

## Michael de Raadt

Faculty of Sciences, University of  
Southern Queensland, Australia  
deraadt@usq.edu.au

## Judy Kay

School of Information Technologies,  
University of Sydney, Australia  
judy@cs.usyd.edu.au

## Raymond Lister

Faculty of IT, Uni. of Technology,  
Sydney, Australia  
raymond@it.uts.edu.au

## Andrew Litchfield

Faculty of IT, Uni. of Technology,  
Sydney, Australia  
ajl@it.uts.edu.au

## Richard Raban

Faculty of IT, Uni. of Technology,  
Sydney, Australia  
richard@it.uts.edu.au

## Paul Roe

Faculty of IT, Queensland Uni. of  
Technology, Brisbane, Australia  
p.roe@qut.edu.au

## Daniel Santamaria

Faculty of IT, Uni. of Technology,  
Sydney, Australia  
dmsantam@gmail.com

## Judy Sheard

Faculty of IT, Monash Uni., Australia  
Judy.Sheard@infotech.  
monash.edu.au

## John Shepherd

School of Computer Science and  
Engineering, UNSW, Australia  
jas@cse.unsw.edu.au

## Andrew Solomon

Faculty of IT, Uni. of Technology,  
Sydney, Australia  
andrews@it.uts.edu.au

## Richard Thomas

Faculty of IT, Queensland Uni. of  
Technology, Brisbane, Australia  
r.thomas@qut.edu.au

## Abstract

The Carrick Institute is an initiative of the Australia federal government. It is aimed at generating strategic change in Australian University education, via grants and other awards to approximately \$20 million annually. By previous Australian standards, the potential funding for projects is large. However, the Carrick Institute has a well focused vision, and grant applications need to be aligned with that vision. This paper first describes some key aspects of the Carrick vision, before describing four multi-institutional computing education projects that successfully attracted funding from the Carrick Institute in 2006. Three of the projects are funded under Carrick's Priority Program, and are concerned with different aspects of automated assessment: (1) assessing Unix scripting skills, (2) self and peer assessment in groupwork, and (3) the assessment of novice programmers. The fourth project is funded under Carrick's Disciplinary-Based Initiatives Scheme. Commonalities in the structure of these three projects are observed.

**Keywords:** Carrick Institute, Assessment, Unix Scripting, Group work, Novice programming, Multi-institutional Collaboration.

## 1 Introduction

Among its six objectives (Carrick, 2006a) the Carrick Institute lists two objectives of particular interest to potential grant applicants from the computer education community:

- Promote and support strategic change in higher education institutions for the enhancement of learning and teaching, including curriculum development and assessment.
- Develop effective mechanisms for the identification, development, dissemination and embedding of good individual and institutional practice in learning and teaching in Australian higher education.

Among its five values, the Carrick Institute lists three that are of particular interest to potential grant applicants:

- Inclusiveness - by assisting the development of networks and communities which support higher education staff who have a direct impact on the advancement of learning and teaching.
- Long-term change - through a focus on systemic change.
- Collaboration - through the programs it funds and in its work practices.

In this paper, we refer to the Carrick objectives and values – in particular those listed above – as the Carrick vision. In the remainder of this introduction, we interpret that vision from an historical perspective, by describing the factors leading to earlier education funding institutions being succeeded by the Carrick Institute.

From 1993 to 1996, a grant scheme was run by the Committee for the Advancement of University Teaching

(CAUT). It funded 448 National Teaching Development Grants, with an average funding level per project of \$37K (Carrick, 2006b). The scheme emulated the traditional process of research funding, with individual academics competing for funding by submitting proposals for peer review. In 1997, CAUT was succeeded by the Committee for University Teaching and Staff Development (CUTSD), which existed for three years (Carrick, 2006c). As part of its brief, CUTSD ran the National Teaching Development Grants scheme. This scheme offered grants in three categories: individual; organisational; and staff development. In three years of operation, 272 grants were awarded. CUTSD also organised activities, such as annual forums, to encourage dissemination of project outcomes.

CUTSD was superseded by the Australian Universities Teaching Committee (AUTC). In a report commissioned by the AUTC, Schofield and Olsen (2000) were critical of the benefit of CUTSD funded projects beyond the immediate benefit to those directly involved in the projects. In their executive summary, they wrote the following:

*“Overall there is a shared perception that dissemination of CUTSD grant outcomes was generally weak ...”* [page 4]

*“The idea of hundreds of individual grant holders being able to disseminate the outcomes of their work sector wide in the proactive manner required to stimulate adoption is unrealistic. What is needed are high quality professional dissemination mechanisms which build on recognised professional networks.”* [page 6]

In the long transition from the AUTC to Carrick, the AUTC provided extensive input into the development of the Carrick Vision. As part of that input, the AUTC commissioned two large studies on the question of how to organize educational projects to achieve effective dissemination (McKenzie *et al.*, 2005; Southwell *et al.*, 2000). In their executive summary, McKenzie *et al.* made a number of recommendations, some of which are of particular interest to computer educators who are potential grant applicants. That subset of the McKenzie *et al.* recommendations are listed below, in the remainder of this section of the paper

Types of projects recommended for funding include:

- Projects aimed at adapting and implementing successful innovations in new institutional and/or disciplinary contexts, in addition to well designed innovation projects;
- Both individual projects and collaborative projects, funded through separate granting pools with a greater proportion of the total funds available for collaborative projects;
- Within the definition of collaborative projects, those involving a lead institution and a set of consultation partners, collaborations involving a small group of partners and cascade models of collaboration.

In relation to application processes:

- [enable applicants to] develop well-designed evaluation and adoption-focused dissemination plans.

In relation to the criteria for assessing applications for funding, include criteria which:

- Emphasise scholarship, particularly in project design and evaluation;
- Emphasise effective evaluation (as described above);
- Require adequate proportions of the budget to be devoted to project management, evaluation and dissemination;
- Require applicants to consider approaches to dissemination which engage potential users throughout development and are focused on the intended adoption, implementation and embedding of project outcomes.

## 1.1 The Priority and Discipline-Based Schemes

To implement these recommendations, the Carrick Institute has developed several different funding schemes for education projects. Two of these schemes are of particular interest to computing educators as (in 2006) several computing education projects have been funded via these two schemes:

- The **Priority Program** (Carrick, 2006d) has funded three of the projects described below. These three computing education projects were successful in a competitive process against applications across many disciplines. Each of these three projects is intended to run for two years, and each has a total budget of approximately \$200K.
- The **Disciplinary-Based Initiatives Scheme** (Carrick, 2006e, p.3) does not operate under the usual competitive model of academic grant funding. Instead, Carrick identifies disciplinary leadership groups and through a consultative process with those groups allocates funding to support large-scale projects in those disciplines. In the case of computing education, Carrick consulted with several Deans and Associate Deans of Information Technology.

## 1.2 Overview

The next four sections each describe a computing education project which was granted funding by the Carrick Institute in 2006. The first three sections are the three projects funded under the Priority Program.

## 2 Assessing Scripting Skills: LinuxGym

Scripting is the “glue” which holds together many different applications in enterprises from banking to genome sequencing. While 10% of all jobs advertised in the IT/Telecommunications sector seek scripting as a skill, it is to the detriment of both industry and education that it has been very difficult to teach and learn.

Developing scripting skills requires consistent practice and feedback. However, with universities nowadays having a large student-to-staff ratio, there is very little scope for providing the necessary feedback, and university degrees have produced only a small percentage of graduates with the necessary scripting skills.

### 2.1 Prior to Carrick Funding

Prior to the application for funding to the Carrick Institute, Solomon began the development of LinuxGym



and used it to teach and assess scripting skills with students at UTS.

In LinuxGym, a student attempts to write a script with a well specified behaviour. When the student asks for feedback, LinuxGym runs the student's script in several ways to compare its behaviour with what is expected. While this testing does not judge the clarity and maintainability of the student's script, it provides feedback on its functional correctness. In terms of education, functionality is the most time-consuming aspect for teachers to assess. The system therefore enables students to gain considerably more practice and feedback on their skills than would otherwise be possible.

Over the last four years LinuxGym has successfully been used to increase both syllabus coverage and the pass rate of large student cohorts at UTS. The undergraduate syllabus coverage has gone from merely copying and editing files to writing complex scripts, while the postgraduate student failure rate has dropped from its previous level of 30–50% failure to almost zero. Furthermore the marking load of teaching staff has been reduced. LinuxGym has been described in greater detail elsewhere (Solomon, Santamaria and Lister, 2006).

## 2.2 The Carrick-funded Priority Project

This Carrick-funded phase of the project is led by Solomon, with Santamaria, Kay, Shepherd and Lister as partners. The project thus spans three institutions: UTS, Sydney, and UNSW.

The funding covers three broad types of activities, development, dissemination, and evaluation. These activities are discussed below.

- LinuxGym will be further developed, so that the software is stable, easy-to-use, free, and Open Source. Also, a website will be developed to support the use of LinuxGym by a broader teaching community.
- The Carrick Institute is also supplying funds to build a community of academics and practitioners, beyond UTS. LinuxGym will be introduced to students in four universities across Australia as a sequence of fun certification events ("LinuxGym 101"). These events will be followed by a workshop for lecturers.
- The Carrick Institute is also supplying funds to formally evaluate the effectiveness of LinuxGym, both for students and teachers.

## 3 Peer Assessment in Group work: TeCTra

The ability to assess the work of others is one of the core skills expected of professionals across many disciplines. Developing this graduate attribute requires the learning by students of self-and-peer evaluation, feedback, and review skills and understandings. Many professionally oriented higher education courses include capstone subjects involving projects that require large student teams. The common assessment strategy for group work of allocating the same mark to all team members is not adequate, as the project tasks are extensive, the teams are large in number (more than 4 members), extend for the whole semester and group work can constitute 100% of the final student assessment. Furthermore, the subject coordinator has limited opportunities to observe and assess the complex group and teamwork dynamics that

are taking place. A peer-assessment and review strategy is required which is ideally formative, diagnostic and summative

### 3.1 Prior to Carrick Funding

Since 2004, the TeCTra system has been developed and trialed in the Faculty of Information Technology at UTS. In addition to recording time spent on the project by individuals, TeCTra also requires the students to rate and provide confidential feedback on each other's contributions on a weekly basis. The time records and ratings are converted into contribution factors for each week.

Each team member is able to see their relative position in his/her own group in terms of contribution and can take corrective actions. The contribution-factors are also monitored by the coordinator for early identification of non-performers who can be offered timely intervention and assistance. The peer review comments provide valuable and influential qualitative feedback to all team members.

The utility of TeCTra is apparent from a comparison of peer assessment behaviours in students before and after the introduction of TeCTra. In three semesters prior to the introduction of TeCTra (1998-01) students did not have any support in allocating individual marks apart from a set of written rules and suggested practices. In this period between 75-90% of all groups opted to have equal or almost equal distribution of marks within the groups. Such a distribution of marks was hardly plausible as groups of 10 students would rarely be so finely balanced in terms of sharing the group-workload. In three recent semesters (2004-5), with TeCTra in use, only 15-20% of all groups allocated marks almost equally

Over the years the TeCTra prototype design has developed more qualitative and quantitative peer feedback, evaluation, review and assessment capacity. Features to facilitate better communication for non-confrontational feedback, and the visibility of data and processes has dramatically changed the students attitudes to peer-assessment.

Tectra has been described in greater detail elsewhere (Raban & Litchfield 2006a, 2006b).

### 3.2 The Carrick-funded Priority Project

This Carrick-funded phase of the project is led by Raban and Litchfield, both of UTS. This phase will make the TeCTra-prototype tool available to other Australian Universities through the further development of the software together with an extensive dissemination strategy.

Further development and pilot-testing will occur in five sites involving in excess of 1500 students across UTS (in three faculties), QUT and Curtin. The tool will be extensively trialed, evaluated, and areas for improvement identified. After two cycles of iterative educational-design and improvement, the tool will be packaged as an Open-Source application and disseminated for use in Australian Universities.

## 4 Assessment of Novice Programmers

The problems of teaching and assessing novice programmers are well known. The project participants are from five Australian universities, all five with a prior record of innovative approaches to assessing the programming skills of students. By combining their existing work, the participants will produce a comprehensive approach to improving the novice programmer assessment experience.

### 4.1 Prior to Carrick Funding

Two of the participating institutions have built software systems for automatic formative and/or summative assessment of programming. Academics at the other three participating institutions have developed methodologies for assessing novice programmers. These five prior activities are described in the next five paragraphs.

At Sydney University, Kay has explored ways to support reflection in novice programmers. There is a large body of evidence suggesting that learning effectiveness is enhanced when learners reflect on their learning experiences. Kay has developed systems to support reflection. Earlier versions were called “Assess”, while more recent versions have been known as “Reflect” (Kay *et al.*, 2000, 2001, 2002).

QUT has developed two software systems for automatic formative and/or summative assessment of programming. The Environment for Learning to Program (ELP) is an online, active, collaborative, and constructive environment. It provides timely formative assessment by presenting students with fine-grained online exercises and answers for Java and C# programming problems. The other QUT system, called ExamGen, is a GUI-based, stand-alone, Java application, designed for the purpose of managing multiple choice and short answer questions. Both systems have been presented at prior ACE conferences (Truong, Bancroft, and Roe, 2003, 2004, 2005; Rhodes, Bower, and Bancroft, 2004; Woodford and Bancroft, 2005).

At Monash, Carbone has worked on ways of systematizing programming assessment tasks (Carbone *et al.*, 2000, 2001, 2002). She advocates that assessment task should follow her NOCCA ORLA principle (detailed in her PhD thesis), an acronym standing for:

- Novelty, Openness, Complexity, Collaboration, Authenticity
- Ownership, Reflection, Linkage, Assessment

At USQ, de Raadt has studied methods for the explicit teaching and assessment of problem solving strategies, using Soloway’s Goal/Plan approach. He has generated new curricular components and altered the teaching and assessment of students at USQ to incorporate explicit problem solving instruction and assessment based on a Goal/Plan framework (de Raadt *et al.*, 2004, 2006).

At UTS, Lister has studied the program comprehension skills of novices, and applied the Bloom and SOLO taxonomies to structuring assessment tasks for novice programmers (Lister *et al.*, 2001–2006).

### 4.2 The Carrick-funded Priority Project

The project work comprises four main areas, as follows:

- **System Development:** The existing systems have been developed for use within their respective institutions. These systems will need to be adapted and generalized to cater to the needs of other institutions.
- **Content Development:** The participants will pool their expertise in assessment design, to populate the existing QUT and University of Sydney online assessment systems with items developed according to the various assessment methodologies.
- **System/Content Evaluation:** As the systems are used across the participating institutions, the effectiveness of the systems, and their content, will be evaluated via the analysis of the performance of students, and also via surveys of students and teachers using the systems.
- **Dissemination:** In the first year, the project participants will go to other universities in their respective cities, and give presentations on the project. These universities will be asked to express interest in eventually using these systems. In the second half of the project, one-day workshops will be held in each of Brisbane, Sydney, and Melbourne, to (1) prepare academics at other institutions to use these systems, and (2) improve the assessment strategies used by those academics. Also in the second year, this project will directly assist other institutions in setting up the infrastructure to use these systems.

## 5 Disciplinary-Based Initiatives Scheme

Under this scheme, a distinction is made between disciplines that have undertaken prior investigations (Category A) and those that have not (Category B). ICT is one of three disciplines selected in the first round of Category A funding for this scheme

### 5.1 Prior to Carrick Funding

Category A funding for ICT is largely in recognition of a project that was conducted by the Computing Education Research Group (CERG) at Monash University. In 2001, the AUTC funded a national project (ICT-Ed) that investigated teaching and learning initiatives in the ICT discipline in Australian universities (Hurst, 2001). The project was broad focused, involving a review of research and using a variety of methods to collect data from and about key stakeholder groups in ICT education — educators, students and employers. From this extensive data collection, information was gained about the types of teaching initiatives undertaken by ICT educators, the ICT educators’ perceptions of factors that promote and inhibit innovation in ICT education, employers’ views of the preparedness of ICT graduates for the workforce, and ICT graduates’ perceptions of the value of their courses.

Among key findings of the ICT-Ed project were that ICT educators face many challenges working in a discipline that is fast changing and with increasing pressures to respond to the needs and demands of students, employers and institutions. Educators generally felt that teaching was unrewarded and undervalued, and institutional agendas tend to discourage educational innovation. The educators generally found their students to be conservative and resistant to innovation and change in teaching practice and learning activities. A constant tension they faced was whether to teach specific skills,

which may help students gain employment, or to teach more generic skills from a foundation of principles and theory, which can prepare students better for long term employment and life long learning. The project also found that little is known about ICT graduates' experiences in the workforce and how they perceived their courses.

An outcome of the ICT-Ed project was a set of 12 recommendations for the improvement of interactions with the outside world, the fostering of educational innovation and the promotion of evaluation of teaching and learning initiatives (Hurst, 2001).

## 5.2 The Carrick-funded Disciplinary Project

Under the Disciplinary-Based Initiatives Scheme, funding for the ICT discipline has been given to a consortium of IT faculties from Monash, QUT, UTS and Wollongong universities, with Wollongong as the lead institution. The project titled "*Managing Educational Change in ICT Discipline at Tertiary Education*" aims to study the nature and dynamics of change in the ICT discipline. This will be focused in the university sector but will investigate the interfaces with the schools and employment sectors. The long-term goals are to identify strategies and develop models to better prepare secondary school students for ICT degrees, further improve the ICT curriculum, build capacity in ICT educators for managing educational change and create better ICT professionals.

The project will be conducted in two stages. The first is a review and scoping stage in which the key issues will be identified and prioritised. This will build upon the work conducted in the ICT-Ed project and other relevant studies. In the second stage, a series of pilot projects will work towards developing a model for ongoing collaborative effort.

## 6 Discussion

The reasons why these projects were funded are perhaps only known to the Carrick Institute. However, all of the projects share four characteristics, which we the authors of this paper believe implement the Carrick vision:

- Prior work: All projects had a track record in the project area prior to approaching Carrick.
- Multi-institutional: All projects are collaborations that cross institutional boundaries. One of the projects (TeCTra) is also multidisciplinary.
- Formal Evaluation: All projects contain mechanisms to formally evaluate the project outcomes.
- Dissemination: All projects have strategies for spreading the developed systems outside the collaborating institutions. (Indeed, the presentation of this paper at this conference is an early dissemination activity of these projects.)

### 6.1 The Competitive Grants Scheme

While the three Priority Program projects are funded under Priority 1 of that scheme, each year Carrick targets specific disciplines for funding under that priority. While Computing and Information Science was targeted in 2006, it is not a target in 2007. Thus, these three projects would not have been eligible for funding under the Priority Program had they been submitted in 2007.

However, computing education projects similar to these three projects may be eligible for funding under Carrick's "Competitive" grants scheme, given that one of the priorities of that scheme is "Innovation in learning and teaching, particularly in relation to the role of new technologies".

In 2007, both full proposals and expressions of interest for the Competitive Grants Program are due on April 23. Applicants who submit an expression of interest that is judged suitable are then required to submit a full proposal by August 13. Negotiating and assembling a multi-institutional collaboration is a major undertaking. It would be ambitious to initiate a collaboration after ACE2007 and complete a full proposal by the April deadline. While existing collaborations might aim to submit a full proposal by April, the authors of this paper suggest that participants in a new collaboration consider a two-stage approach, where the applicants first aim to submit an expression of interest, followed by a full proposal.

## Acknowledgements

The full ICT Discipline-based Initiatives project team is Prof. Joe Chicharo, A/Prof. Fazel Naghdy (Wollongong), Prof. Ron Weber, Dr Judy Sheard (Monash), Prof. Simon Kaplan, A/Prof. Christine Bruce (QUT), Prof. Tharam Dillon, A/Prof. David Wilson (UTS).

## References

- Carbone, A., Mitchell, I. J., Hurst, A. J. and Gunstone, R. (2000). *Principles for designing programming exercises to minimise poor learning behaviours in students*. The Fourth Australasian Computing Education Conference, Monash University, Victoria, Australia.
- Carbone, A., Mitchell, I. J., Gunstone, R. and Hurst, A. J. (2001). *Characteristics of programming exercises that lead to poor learning tendencies: Part II*. The 6<sup>th</sup> Annual Conference on Innovation Technology in Computer Science Education, University of Kent, Canterbury, UK.
- Carbone, A., Mitchell, I. J., Gunstone, R. and Hurst, A. J. (2002). *Designing programming tasks to elicit self management metacognitive behaviour*. International Conference on Computers in Education, (ICCE 2002), Auckland, New Zealand.
- Carrick (2006a) Mission, Objectives, Values. <http://www.carrickinstitute.edu.au/carrick/go/pid/10>. [Accessed October 2006].
- Carrick (2006b) CAUT Program 1994-5. <http://www.carrickinstitute.edu.au/carrick/go/pid/68> [Accessed October 2006].
- Carrick (2006c) CUTSD Program 1997-99. <http://www.carrickinstitute.edu.au/carrick/go/pid/69> [Accessed October 2006].
- Carrick (2006d) Priority Projects Program. <http://www.carrickinstitute.edu.au/carrick/go/pid/111> [Accessed October 2006].
- Carrick (2006e) Discipline-based Initiatives Scheme. <http://www.carrickinstitute.edu.au/carrick/go/pid/120> [Accessed October 2006].

- de Raadt, M., Toleman, M. and Watson, R. (2006) *Chicken sexing and novice programmers: Explicit instruction of problem solving strategies*, Australasian Computing Education Conference (ACE 2006), Conferences in Research and Practice in Information Technology, **52**, 55-62 (Hobart, Australia, 16-19 January).
- de Raadt, M., Watson, R., Toleman, M. (2004) Training strategic problem solvers, *ACM SIGCSE Bulletin*, **36**(2). Impagliazzo, J (Ed). ACM Press.
- Hurst, J., Lynch, J., Collins, F. & Markham, S. (2001). Teaching ICT: The ICT-Ed Project. Higher Education Division, Department of Education, Training and Youth Affairs. Canberra, Australia.
- Kay, J. (2000). Stereotypes, student models and scrutability. *Intelligent Tutoring Systems*. Lecture Notes In Computer Science; **1839**, 19-30.
- Kay, J. (2001). Learner control. *User Modeling and User-Adapted Interaction*, **11**(1/2) 111-127.
- Kay, J. B. Kummerfeld, and P. Lauder. (2002). *Personis: A server for user models*. Adaptive Hypermedia and Adaptive Web-Based Systems, Second International Conference, 203-212.
- Lister, R. (2001). *Objectives and Objective Assessment in CSI*, 32nd Technical Symposium on Computer Science Education (SIGCSE 2001), Charlotte NC, USA, February 21-25, 2001. pp 292-296.
- Lister, R. and Leaney, J. (2003). *Introductory Programming, Criterion Referencing, and Bloom*, 34th Technical Symposium on Computer Science Education (SIGCSE 2003), Reno, Nevada USA, February 19-23, 2003, pp 143-147.
- Lister R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, E., Sanders, K., Seppälä, O., Simon, B., Thomas, L., (2004). *A Multi-National Study of Reading and Tracing Skills in Novice Programmers*, SIGCSE Bulletin, Volume 36, Issue 4 (December), pp. 119-150.
- Lister, R., (2005). *One Small Step Toward a Culture of Peer Review and Multi-Institutional Sharing of Educational Resources: A Multiple Choice Exam for First Semester Programming Students*. Seventh Australasian Computing Education Conference (ACE2005). Newcastle, Australia. January 31 – February 3. pp. 155- 164.
- Lister, R., Simon, B., Thompson, E., Whalley, J., and Prasad, C. (2006). *Not Seeing the Forest for the Trees: Testing Novice Programmers to Encourage the Development of Relational Thinking*. Eleventh Annual Conference on Innovation and Technology in Computer Science Education, Bologna, Italy. pp. 118 - 122.
- McKenzie, J., Alexander, S., Harper, C., and Anderson, S. (2005) *Dissemination, Adoption & Adaptation of Project Innovations in Higher Education*. <http://www.carrickinstitute.edu.au/carrick/go/op/edit/pid/98> [Accessed October 2006]
- Raban and Litchfield (2006a). *Peer assessment in large group projects: forming professional attitudes in IT students*. <http://science.uniserve.edu.au/pubs/procs/2006/litchfield.pdf>
- Raban and Litchfield (2006b). *Supporting Peer Assessment of Individual Contributions in Groupwork*. Annual conference of the Australasian Society for Computers in Learning in Tertiary Education (ASCLiTE 2006). Sydney, December 3-6.
- Rhodes, Anthony, Bower, Karyn and Bancroft Peter (2004). *Managing Large Class Assessment*. Australasian Computing Education Conference (ACE2004), Dunedin, New Zealand. pp. 285-289.
- Schofield, A. and Olsen, A. (2000) *A Report for AUTC on: An Evaluation of the CUTSD Initiative*. <http://www.carrickinstitute.edu.au/carrick/webdav/site/carricksite/users/siteadmin/public/116.pdf> [Accessed October 2006]
- Solomon, A., Santamaria, D. and Lister, R. (2006) *Automated Testing of Unix Command-line and Scripting Skills*, 7th International Conference on Information Technology Based Higher Education and Training. (Sydney, Australia, July 10 - 13, 2006).
- Southwell, D., Gannaway, D., Orrell, J., Chalmers, D., Abraham, C. (2005) *Strategies for effective dissemination of project outcomes*. <http://www.carrickinstitute.edu.au/carrick/go/op/edit/pid/94> [Accessed October 2006]
- Truong, Nghi, Bancroft, Peter and Roe, Paul (2005). *Testing 'Fill in the Gap' Programming Exercises*. Australasian Computing Education Conference (ACE2005), Newcastle, Australia.
- Truong, Nghi, Bancroft, Peter and Roe, Paul (2004). *Static Analysis of Students' Java Programs*. Australasian Computing Education Conference (ACE2004), Dunedin, New Zealand. pp. 317-325.
- Truong, Nghi, Bancroft, Peter and Roe, Paul (2003). *A Web Based Environment for Learning to Program*. Australasian Computing Education Conference (ACE2003), Adelaide, Australia. pp. 255-264.
- Woodford, Karyn and Bancroft, Peter (2005). *Multiple Choice Questions Not Considered Harmful*. Australasian Computing Education Conference (ACE2005), Newcastle, Australia. pp.109-116.

# CONTRIBUTED PAPERS



# WAT – A Tool for Classifying Learning Activities from a Log File

Jason Ceddia, Judy Sheard, Grant Tibbey

Caulfield School of Information Technology

Monash University

PO Box 197, Caulfield East, VIC, 3145, Australia

jason.ceddia{judy.sheard,grant.tibbey}@infotech.monash.edu.au

## Abstract

Web-based learning environments are now extensively used as integral components of course delivery in tertiary education. To provide an effective learning environment, it is important that educators understand how these environments are used by their students. This paper describes an approach to analysing website interactions data captured on log files that goes beyond simply tabulating frequencies of navigation path traversals through a website. A Weblog Analysis Tool (WAT) has been built to facilitate this process. WAT allows the educator to describe *activities* from sequences of website interactions that are meaningful in the course context. The tool is then used to process the log file to determine which students completed these activities. The performance of activities provides insights into learning behaviour of students over the duration of the course. In this paper the application of the WAT tool is demonstrated through a study of interactions with a courseware website. The activities defined in this study relate to the learning objectives of the course and so provide the educator with an indication of how successful the website has been in assisting the students meet these objects.

**Keywords:** log file analysis, online learning, Web-based learning environments

## 1 Introduction

Web-based learning environments are now typically provided as a core component of tertiary teaching and learning environments. Many researchers have emphasised the need to investigate how students use and respond within these environments, and to establish linkages between student behaviour and other factors such as academic performance and the quality of their learning experiences (Nachmias & Segev, 2003; Zaiane, 2001; Zaiane & Luo, 2001). However, the developers of these websites have very few support tools to help them evaluate the behaviour of the learners within these environments. Analysis of log file data of student interactions to gain information about learning behaviour

has typically focused on frequencies of Web page visits and duration of visits (Hwang & Li, 2002; Nachmias & Segev, 2003; Sheard, Ceddia, Hurst & Tuovinen, 2003a, 2003b). Other researchers have focused on exploring navigation paths of website visits, represented by sequences of interactions, to provide deeper insights into student learning behaviour (Ford & Chen, 2000; Schoon & Cafolla, 2002). The data abstractions and analysis techniques used in these studies are useful for comparisons of behaviour; however, they are limited in their capacity to provide detailed information about the activities of the student.

A difficulty with trying to gain a view of learning behaviour from sequences of interactions is that navigation paths form a complex data set. A navigation path recorded for a single website visit can represent many different activities of the user. These activities may interweave, forming a complicated network of relationships (Card, Pirulli, Van Der Wege, Morrison, Reeder, Schraedley & Boshart, 2001). To explore learning behaviour from log file data meaningfully, it is necessary to look in detail at the patterns of interactions within navigation paths. Examination of specific navigation patterns can reveal the strategies that students use and can assist in understanding the causes of any problems that may occur during a website visit.

To gain understanding of learning behaviour from analysis of website interactions, it is proposed that it is necessary to look at sequences of interactions for particular *tasks* that the learner performs during a visit. Supporting this view, Horney (1993) argues that more attention should be paid to the fundamental activities that a user has engaged in before these actions are abstracted into strategies. An important issue with analysis of log files was raised by Berendt and Spiliopoulou (2000) who claimed that care needs to be taken when inferring relationships between pages based on linking structure. They maintain that linked pages does not necessarily mean that there is a semantic relationship. It is therefore important that the educator is involved when making inferences about sequences of website interactions, as the educator has an understanding of the educational purpose of their website.

In a search of the literature, only a few studies were found that explored log file analysis at the task level. Most micro analyses of log files focus on the discovery of frequently occurring associations or sequences of interactions rather than finding semantically meaningful sequences (Baumgarten, Büchner, Anand, Mulvenna & Hughes, 1999; Spiliopoulou, 1999). A number of sequence mining tools were found that search for

---

Copyright © 2007, Australian Computer Society, Inc. This paper appeared at the *Ninth Australasian Computing Education Conference (ACE2007)*, Ballarat, Victoria, Australia, January 2007. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 66. Samuel Mann and Simon Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

specified sequences. However, these were too general for application to this research (for example, the WEBMINER Web usage mining system developed by Cooley, Mobasher and Srivastava (1997)), or were developed for commercial applications and are limited in their application to Web-based learning environments where the interpretation of activities is often very different. The sequence mining tool developed by Zañane and Luo (2001) was the only one found that claimed to have been developed specifically for an educational environment; however, this was only developed to the prototype stage.

The review of the literature has shown that the current methods of log file data definition and analysis are limited in their potential to provide useful information about learner behaviour from log file analysis. This provided the impetus for the conceptualisation of an abstraction of log file data at the *activity* level and the development of a tool to assist in the process of analysing log file data at this level. This paper presents the concept of an *online learning activity* and describes a tool developed to ‘mine’ these activities from a log file. The results of a study, which used this tool to gain information about learning behaviour in a courseware website, are reported. The paper concludes with ideas for future work.

## 2 Learning Activities

Students working in an online environment are typically engaged in a variety of different interactions with the environment. Conceptually, each interaction can be viewed as relating to a particular activity of the student, with different types of activities defined by single interactions, or groups or sequences of interactions. We have termed this abstraction a *learning activity* and it is defined as follows: “A learning activity is a series of interactions within a website to achieve a particular outcome.” A series in this context is used to describe a contiguous sequence or group of interactions. The concept of a learning activity provides a meaningful behavioural abstraction of log file interactions for the purpose of analysis of learning behaviour within a Web-based learning environment.

Online learning activities may range from those that are undirected to those that are specific to a learning task. Viewed from another perspective, learning activities differ according to how critical they may be to the performance of a task. For example, zooming in or sorting information may be performed as part of a task but may not be essential for its completion, whereas navigation to a particular page or downloading information may be critical for the performance of a task. Using these criteria, learning activities may be organised into five categories: *presentational*, *organisational*, *explorational*, *goal-focused* and *complex*. These are described in Table 1. The categories were organised into a hierarchy of levels ranging from level 1, where activities may be seen as preparation for a task or assisting in performance of a task, to numerically higher levels where activities describe essential components of tasks or complete tasks. A learning activity may apply to

a range of applications or may be specific to a particular application. In the online learning activity hierarchy, the general activities are found at the numerically lower levels. The hierarchy thus shows categories that are organised to describe increasing levels of more deliberate, focused and complex learning activities.

**Table 1: Categories of Online Learning Activities.**

Online Learning Activities
<p><b>1. Presentational activities</b> – these activities are concerned with adjusting or changing how information is viewed, for example, maximising a window, changing font size, zooming in/zooming out etc. Students may perform these to bring information into view or gain a clearer view of information. The following are examples of descriptors for this type of activity: <i>View, Show, Highlight, Zoom in, Zoom out, Contract, Expand, Reduce, Extend, Scroll</i></p>
<p><b>2. Organisational activities</b> – these activities are concerned with organising, arranging, transferring or preparing information. Students may perform these to locate information, show the relationship between items of information, or represent information in a more meaningful or comprehensible way. The following are examples of descriptors for this type of activity: <i>Itemise, Sort, Classify, Categorise, Group, Cluster, Filter, Drag And Drop, Download, Upload, Associate, Link</i></p>
<p><b>3. Explorational activities</b> – these activities are concerned with exploring or gaining access to information. Students may perform these to experiment with learning tasks, explore a concept or gain familiarity with the learning environment and resources. These are not primarily task-oriented or deliberate learning activities. The following are examples of descriptors for this type of activity: <i>Browse, Explore, Search, Navigate</i></p>
<p><b>4. Goal-focused activities</b> – these activities are concerned with integrating, assimilating and evaluating information with the aim of gaining understanding or learning. Students may perform these as deliberate learning tasks. Alternatively, these activities may be concerned with completing a task that is indirectly related to their learning, for example, the task may be necessary to satisfy assessment requirements. Students perform these as required tasks. The following are examples of descriptors for this type of activity: <i>Add, Edit, Delete, Post, Read</i></p>
<p><b>5. Complex activities</b> – these describe activities are concerned with abstract mental activity or multiple activities. These are concerned with applying knowledge at a higher or more complicated level that occurs with the explorational or goal-focused level of activities. These may be demonstrated in open-ended learning tasks. The following are examples of descriptors for this type of activity: <i>Plan, Synthesise, Hypothesise, Generalise</i></p>



**Table 2: A sequence of log records from WIER.**

LOGID	USERID	USER_TYPE	IEGROUPID	PAGE_URL	RESOURCE	ACTION	ACTION_QUALIFIER
INTERACTION_DATA			SESSIONID			DATE/TIME	
7987	7303	Student	461	/task-tracker-v2.php	NewTimeTracker	start	
			f3f92ee54fd48dfaff88a7ad753a5287			31/10/2004	8:30:49 PM
7988	7303	Student	461	/task-tracker-v2.php	NewTimeTracker	select	task
filter_userid:-1,taskid:88174			f3f92ee54fd48dfaff88a7ad753a5287			31/10/2004	8:30:58 PM
7989	7303	Student	461	/edittask.php	NewTimeTracker	submit	delete
Taskid:88174,did_delete:true			f3f92ee54fd48dfaff88a7ad753a5287			31/10/2004	8:31:06 PM
7990	7303	Student	461	/task-tracker-v2.php	NewTimeTracker	start	
			f3f92ee54fd48dfaff88a7ad753a5287			31/10/2004	8:31:16 PM
7991	7303	Student	461	/task-tracker-v2.php	NewTimeTracker	select	task
filter_userid:-1,Taskid:88172			f3f92ee54fd48dfaff88a7ad753a5287			31/10/2004	8:31:23 PM
7992	7303	Student	461	/logtime.php	NewTimeTracker	transit	logtime
taskid:88172			f3f92ee54fd48dfaff88a7ad753a5287			31/10/2004	8:31:37 PM
7993	7303	Student	461	/logtime.php	NewTimeTracker	submit	logtime
taskid:88172			f3f92ee54fd48dfaff88a7ad753a5287			31/10/2004	8:31:45 PM

An abstraction of learner interactions within a Web-based learning environment at the activity level enables analysis of log file data to gain information about the tasks that the learner has engaged in. This process involves the capture and recording of appropriate data and the mapping of sequences of interactions to conceptualisations of learning activities. This requires the definition of learning activities in terms of the learner interactions. In order to gain meaningful definitions, it is necessary that the educator and/or instructional designer are involved as they have an understanding of the educational intent of the website. The following sections illustrate these ideas using a courseware website as a context.

### 3 The WIER Log File

The WIER website (Web Industrial Experience Resource) was used by final year computing students completing a capstone or industrial experience (IE) project. The WIER website is predominantly concerned with *goal-focused* activities reflecting the functional nature of the website.

Student interactions with the courseware website were automatically collected and recorded in a central database. Incorporation of demographic data allowed comparisons based on gender and course performance, providing insights into the work patterns of different groups and enabling any relationship of website usage and learning outcomes to be determined.

The WIER website can be viewed logically as a collection of resources within a virtual domain. Within each resource, different activities may be conducted. To enable easy identification and association of the interactions with learning activities, each page on the website needed to be identified. Each interaction was classified as belonging to a particular *resource* and according to the type of task (*action*) and particular action within the task (*action qualifier*). A subset of the fields from the WIER Log File are shown in the heading row of Table 2.

### 4 Defining Learning Activities

This section describes how learning activity abstractions of log file data can be defined. Log file data for the WIER application is used to illustrate this process. The abstractions of learning activities required definitions of learning activities in terms of interactions. As an example, the definition of a goal-focused activity will now be given. A goal-focused activity is defined by a single action or a sequence of actions. Each action sequence has a specified start action and finish action. The specification of start and finish actions means that partial or incomplete activities can be determined. The actions within the sequence may or may not be specified to occur in a set order. A goal-focused action sequence may not be contiguous as there may be presentational, organisational or explorational activities interspersed within the sequence. The sequence may also have repetitions of actions. Using Backus Naur Form (BNF) notation, a goal-focused sequence could be expressed as:

```
<goal_focused_activity> ::= <goal_focused_single> |
  (<goal_focused_start> {<goal_focused_transit> |
    <organisational_activity> | <presentational_activity> |
    <explorational_activity>} <goal_focused_end>)
```

In the sample set of seven log records listed in Table 2, (LOGID numbers 7987 to 7993) show that user 7303 is a 'student' user type in IE group number 461 and is using the 'NewTimeTracker' resource. All interactions took place in the same session (SESSIONID = f3f92...) on the 31/10/2004 at approximately 8.30pm.

The sequence beginning with log record 7987, has the student starting (ACTION = start, ACTION\_QUALIFIER and INTERACTION\_DATA = null) the NewTimeTracker RESOURCE. In log record 7988, the student selects (ACTION = select) a task (ACTION\_QUALIFIER = task) and the particular task is shown in the interaction data (INTERACTION\_DATA = filter\_userid:-1,taskid:88174). In log record 7989, the student has clicked the delete link for the previously selected task and the delete was successfully completed (ACTION = submit,

ACTION\_QUALIFIER = delete and INTERACTION\_DATA = taskid:88174,did\_delete=true). In log record 7990 the student has moved onto another action (ACTION = start, ACTION\_QUALIFIER and INTERACTION\_DATA = null) using the NewTimeTracker resource.

The log file data was analysed using learning activity abstractions derived by applying the online learning definitions. This enabled a detailed analysis of the tasks that the learners had performed. Some questions that guided this stage of the study were:

1. What activities did the students perform on the website?
2. What time did students spend on various activities?
3. What were the patterns and trends in activities over the year?

## 5 WAT- Weblog Analysis Tool

A prototype application, called Weblog Analysis Tool (WAT), was built to enable the definition of activities for a particular log file and then extract activities from that log file using those definitions. WAT currently has two phases; phase 1 is *activity definition* and phase 2 is *activity extraction*.

Phase 1 has three steps as follows:

1. Select the log file that will be reported on and the data columns (fields) that will be used. A filter can be applied to select specific records; for example, to select records for a particular semester.

2. Choose the fields that will be used to specify an activity. In Figure 1 the fields chosen are *category (resource)*, *action* and *action qualifier*.
3. Now define the activities, that is, what do the values of the selected interaction attributes actually mean. For example, in the first row of the table in Figure 2, we have defined a sequence of interactions to be an activity called *File manager-download file*. For this sequence to have been recorded in the log file, the student must have selected the File Manager from the menu, browsed to the required file and clicked the download button; all of these actions must have been performed to show the UL\_CATEGORY attribute = 'filemanager', the UL\_ACTION = 'Submit' and the UL\_ACTION = 'Download' in the logfile. In Figure 2; the defined activities are in the first four table columns and the data from the log file is in the last three columns. Activity definitions can be saved and reused later.

In Phase 2, the log file is processed to extract activities according to the activity definitions from Phase 1. The results of this are shown in the bottom half of Figure 2. This is a listing of all the activities extracted from the log file using the definitions from Phase 1. A tick in the 'completed' column for a particular activity, means that the student has actually completed the activity as opposed to abandoning it halfway through and going onto some other activity.

Figure 1 Selecting the columns that will be used to define the activities from a previously selected log file.

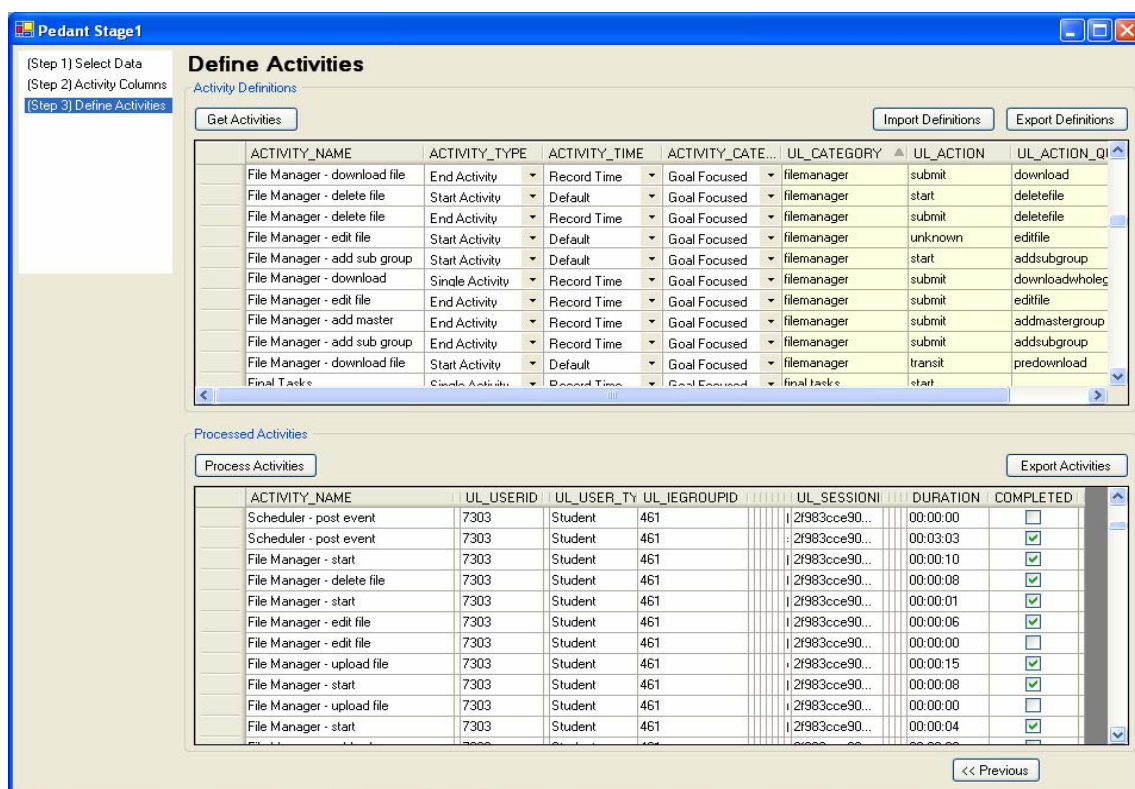


Figure 2 Processing the log file against the defined activities.

## 6 Results

The first component of the study involved an analysis of log file data collected at the WIER website for a cohort of students enrolled in the IE Project course in 2004. In this group there were 258 students organised into 53 project groups. There were 182 (71%) male and 76 (29%) female students in the group.

All online interactions with the WIER website were collected over a period of 41 weeks. The data collection period covered the entire period of time that most of the students used the website for their project work. Table 3 shows a summary of activity types and their frequency. These comprised all possible activities that could be defined at these levels.

**Table 3 Activities Extracted from the WIER Log File**

Activity category	Number of different types of activities	Total activities
Presentational	2	478
Organisational	13	201,790
Explorational	24	38,542
Goal-focused	69	299,889
Complex	4	33,920
<b>Overall</b>	<b>112</b>	<b>574,619</b>

We will now present a brief outline of the analysis techniques used on the extracted activities to gain information about learning behaviour. Measures of individual and groups of activities were analysed to provide the following information:

1. The completion rates of activities were used to determine *effectiveness* of activity performance.
2. The length of activity sequences and duration of goal-focused activities were used to determine *efficiency* of activity performance.
3. The frequency and length of sequences of explorational activities were used to investigate the *style* of learning behaviour.
4. Unusual or outlying results were used to provide indications of *problems* with the website interface or learning activity.

To illustrate the usefulness of the information that was gained, two examples of analysis will now be given.

### 6.1 Effectiveness of learning behaviour

Effectiveness in the use of a Web-based learning environment may be indicated by students achieving the objective of their visits. Measures of effectiveness in the use of WIER were indicated by the successful completion of goal-focused activities. The completion rates varied from 22% for *Delete a news item* to 96% for *Download a file group from File Manager*. The

completion rates over the course of the project increased for most activities and Mann-Whitney U tests\* indicated that this increase was significant for almost a third of the activities. The completion rates were generally higher for the high-achieving and the female students; however, in most cases these differences were not significant.

Further analysis of these activities involved examining sequences of unfinished activities to give a measure of the students' efficiency and effectiveness in performing activities. Sequences of unfinished activities either ended successfully with eventual completion of the task or ended with the student abandoning the task and attempting another task. Comparison of the frequencies of these different sequences was used to indicate effectiveness in task completion. A Kruskal-Wallis† test indicated that the completion rates increased over the course of the IE project,  $\chi^2(2, N = 752) = 84.30, p < 0.01$ . No difference was found based on gender or IE project performance.

Comparison of the lengths of the sequences ending in a successful task completion was used to give a measure of any trend in task effectiveness. A Kruskal-Wallis test indicated that the length of the sequences decreased over the course of the project,  $\chi^2(2, N = 752) = 9.41, p < 0.01$ . There were no differences found based on gender or IE project performance. A similar comparison of partially completed sequences indicated that the length of these sequences also decreased over that course of the project,  $\chi^2(2, N = 752) = 12.80, p < 0.01$ , and there were no differences based on gender or IE project performance. These changes could indicate that, in time, students became more effective with their use of WIER and/or experienced fewer difficulties.

## 6.2 Style of learning behaviour

The *style* of learning behaviour can be viewed as the way in which the students conduct their learning. The students can conduct their learning in many ways in the WIER website. They have the freedom to perform tasks in any order and tasks may be intermingled with non-task related interactions. Analysis of sequences of activities can provide an understanding of the style of their learning behaviour.

One aspect of learning behaviour is the style of navigation. To gain information about navigation, two different types of sequences of explorational activities were analysed. A *purposeful behaviour* was defined as navigation that ends with the student performing an interactive task and a *browsing behaviour* was defined as navigation that ends with the student performing a

passive activity such as the reading of a topic. The number of *purposeful behaviour* activities was 25,705 compared with 6,498 *browsing behaviour* activities. The higher proportion of *purposeful behaviour* activities reflects the functional nature of the website where students used the website for specific tasks rather than browsing. There were no significant differences in the proportion of *purposeful behaviour* to *browsing behaviour* activities based on gender or IE project performance.

Considering the nature of each type of explorational sequence, it might be expected that patterns would emerge over the course of the IE Project. It is reasonable to expect that students would become more efficient with their navigation and therefore *purposeful behaviour* sequences would become shorter. However, a Kruskal-Wallis test indicated no difference in the length of the *purposeful behaviour* sequences over the three equal time periods from the start to the end of the IE Project. In addition, it might be expected that students would have less need or inclination to browse the website as the project progresses and they become familiar with the website, and therefore the length of *browsing behaviour* sequences would decrease. In this case, this was supported by Kruskal-Wallis tests which indicated that the *browsing behaviour* sequences decreased in length over the three equal time periods from the start to the end of the IE Project,  $\chi^2(2, N = 682) = 10.42, p < 0.01$ . Furthermore, it might be expected that the nature of their visits would shift to more *purposeful behaviour* activities. A Kruskal-Wallis test indicated that the proportion of *purposeful behaviour* sequences increased over the course of the project,  $\chi^2(2, N = 32203) = 414.08, p < 0.01$ .

## 7 Conclusion and Future work

The analysis of the WIER log file data using a learning activity level of abstraction has provided valuable and previously unobtainable insights into the learning behaviour of the students using this courseware website. The WAT tool facilitated this process, enabling the mining of learning activities from the large and complex structured log file data.

Currently, WAT performs activity definition and extraction. The next step in the development of this tool is to use the activities to determine whether educational objectives of the course have been met. For example, if a student successfully completes a time recording activity twice per week then they may be deemed to have practiced, and perhaps understood, time management. The interface of an initial prototype of this tool is illustrated in Figure 3.

More metrics are needed for determining the achievement of educational objectives from activities. Currently, WAT uses frequency of completed activity per period but intuitively this seems too limited. Future work will use log files from other educational applications to explore this further.

\* Mann Whitney U test is the non-parametric equivalent of a t-test.

† Kruskal-Wallis test is the non-parametric equivalent to an ANOVA test.

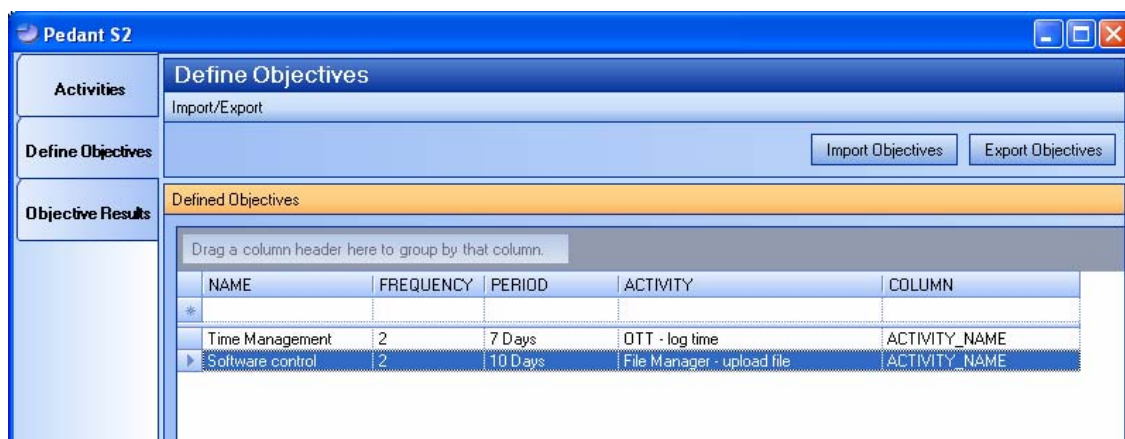


Figure 3 Defining the measure of the educational objectives using the frequency of performance of activities.

## 8 References

- Baumgarten, M., Büchner, A. G., Anand, S. S., Mulvenna, M. D., & Hughes, J. G. (1999). User-driven navigation pattern discovery from Internet data. *Proceedings of the Workshop on Web Usage Analysis and User Profiling (WEBKDD '99)*, San Diego, California, USA, 74-91.
- Berendt, B., & Spiliopoulou, M. (2000). Analysis of navigation behaviour in web sites integrating multiple information systems. *The International Journal on Very Large Data Bases*, 9, 56-75.
- Card, S., Pirolli, P., Van Der Wege, M., Morrison, J. B., Reeder, R. W., Schraedley, P. K., & Boshart, J. (2001). Information scent as a driver of Web behavior graphs: Results of a protocol analysis method for Web usability. *Proceedings of the Human Factors in Computing Systems 2001 conference*, Seattle, Washington, USA, 498-505.
- Cooley, R., Mobasher, B., & Srivastava, J. (1997). Web mining: Information and pattern discovery on the World Wide Web. *Proceedings of the Ninth IEEE International Conference Tools and Artificial Intelligence (ICTAI'97)*, 558-567.
- Ford, N., & Chen, S. Y. (2000). Individual differences, hypermedia navigation, and learning: An empirical study. *Journal of Educational Multimedia and Hypermedia*, 9(4), 281-311.
- Horney, M. A. (1993). Case Studies of Navigational Patterns in Constructive Hypertext. *Computers and Education*, 20(3), 257-270.
- Hwang, W.-Y., & Li, C.-C. (2002). What the user log shows based on learning time distribution. *Journal of Computer Assisted Learning*, 18, 232-236.
- Nachmias, R., & Segev, L. (2003). Students' use of content in Web-supported academic courses. *The Internet and Higher Education*, 6, 145-157.
- Schoon, P., & Cafolla, R. (2002). World Wide Web hypertext linkage patterns. *Journal of Educational Multimedia and Hypermedia*, 11(2), 117-139.
- Sheard, J., Ceddia, J., Hurst, J., & Tuovinen, J. (2003a). Determining Website usage time from interactions: Data preparation and analysis. *Journal of Educational Technology Systems*, 32(1), 101-121.
- Sheard, J., Ceddia, J., Hurst, J., & Tuovinen, J. (2003b). Inferring student learning behaviour from website interactions: A usage analysis. *Journal of Education and Information Technologies*, 8(3), 245-266.
- Spiliopoulou, M. (1999). The laborious way from data mining to Web log mining. *Computer Systems Science & Engineering*, 14(2), 113-126.
- Zaiane, O. R. (2001). Web usage mining for a better Web-based learning environment. *Proceedings of the Advanced Technology for Education conference*, Banff, Alberta, Canada, 60-64.
- Zaiane, O. R., & Luo, J. (2001). Towards evaluating learners' behaviour in a Web-based distance learning environment. *Proceedings of the Advanced Learning Technologies*, Madison, Wisconsin, USA, 357-360.



# Why Teach Unix?

**Bernard Doyle & Raymond Lister**

Faculty of Information Technology  
University of Technology, Sydney  
Jones St. Broadway NSW 2007

bjd@it.uts.edu.au raymond@it.uts.edu.au

## Abstract

This paper examines computing academics' conceptions of the Unix operating system, and the purpose of teaching Unix. Interview transcripts from nine academics were analysed phenomenographically. A small number of qualitatively different conceptions of Unix were identified, within two broad categories. The first broad category manifested a technical approach to Unix. Within this broad category, the conceptions of Unix were, from the least to most sophisticated – (1) Unix as a set of unrelated commands; (2) Unix as a command line interface superior to GUIs; and (3) Unix as a problem solving tool. The second broad category was a non technical conception of Unix, in which Unix was seen as a resource that is cheap, secure and robust. With regard to teaching Unix, two broad categories of reasons were identified – practical and pedagogical. These results for teachers are broadly consistent with an earlier phenomenographic study of student conceptions of Unix.

**Keywords:** Phenomenography, Unix.

## Introduction

There have been major changes in the content of IT courses over the last 15 years. The ubiquity of personal computers running Microsoft windows, the impact of the internet as well as the widespread adoption of object oriented programming languages such as Java and C++ have meant that the content of computing degrees has altered radically. There has been a trend away from subjects dealing with low level technical details in favour of those with a high level approach or a managerial perspective. Courses in assembler languages, logic and discrete mathematics, compiler construction and computer hardware are now taught as electives or have become the responsibility of engineering faculties. The teaching of operating systems in general, rather than teaching a specific operating system, is now usually an elective, if it is offered at all. The change in the content of contemporary computing studies at university level is also reflected in the fact that universities now offer students choices in degrees of Information Technology,

Computer Science, Software Engineering and Information Systems.

With the downturn in student enrolments, many Australasian universities are redesigning their degrees, in the hope of attracting more students. For academics participating in such redesigns, the stakes are high. Topics that some computing academics have loved and taught for many years are being removed as part of degree redesign, to make room for new topics.

Not all the change is one way. There is a “back to basics” movement, which advocates reversing some of the recent changes in computer education replace. For example, there has recently been a vigorous debate on the teaching of the first programming subject, with one side advocating a change back from teaching objects-early to the traditional procedural approach (Astrachan et al., 2005; Bruce, 2005; Reges, 2006). At least one Australian university has done exactly that, changing from C to Java as the first language taught, but subsequently changing back to C.

Lewis and Smith (2005) have placed these sorts of debates into a broader framework, arguing that members of the computing education community tend to debate curriculum issues from within three main conceptual frameworks – segregationist, integrationist, and synergist. Academics within the first of those frameworks argue for a traditional computer science syllabus, academics in the second frameworks argue for change while those in the third framework argue that syllabi should incorporate both traditional topics and new topics.

In the case of Unix, the debate is not so much about whether Unix should be taught at all – it appears most academics believe it should be taught – but instead the debate is about the depth to which Unix should be taught in redesigned degrees, and also the style of instruction that should be used to teach Unix.

In this paper, the authors do not argue their own position in the Unix debate. Instead, we seek to document and formalise the various views on Unix and its teaching. In particular, we seek to make explicit what often remains implicit in the heat of committee room debate. Our intention is similar to that of McCauley (2004) who, within the context of a different syllabus debate, advocated that participants need explicit agreement on terminology, so they can “*clearly and succinctly express what they had tried to say, previously, using plain English*”. We believe that most syllabus debates would benefit from scholarly analysis of the debate itself.

---

Copyright © 2007, Australian Computer Society, Inc. This paper appeared at the Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, Australia, January 2007. Conferences in Research in Practice in Information Technology, Vol. 66. Samuel Mann and Simon Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.



### 1.1 Prior study of student conceptions of Unix

In an earlier paper, the authors conducted a phenomenographic study of conceptions of Unix among students attending the authors' university (Doyle and Lister, 2006). In that study, we noted that students compartmentalized their appreciation of Unix. For example:

*"... students appear to see the superior security of Unix as an "accidental" property of Unix, not a consequence of the architecture of Unix. Perhaps, as we collect more interview transcripts, we will see students who do articulate such a connection. On the other hand, perhaps such a connection is not currently being articulated by the teachers".*

In this paper, we present a new phenomenographic study, which is similar to that prior study, but in this study we examine academic teachers' conceptions of Unix. We also study the teachers' understandings of the purpose of teaching Unix in contemporary computing degrees. This study addresses the above speculation from the earlier study, that perhaps students are not making certain connections in their conception of Unix because those connections are not commonly being articulated by their teachers

### 1.2 Phenomenography

Phenomenography is a qualitative research technique which looks at the different ways people perceive, conceptualise, approach or understand a phenomenon. (Ackerlind, 2005). Usually, phenomenographic data consists of interview transcripts. The data is analysed to identify the qualitatively different ways in which the phenomenon is conceived. These different ways of knowing are referred to as the Categories of Description. The interrelationships between the categories define what is known as the outcome space. This space is often linear. That is, there is a single aspect that varies qualitatively across the categories. In such a linear space, the categories often form a hierarchy, with higher categories being more sophisticated conceptions that subsume the lower conceptions.

Phenomenography is widely used as an education research technique. It has been used to analyse student's conceptions of various academic disciplines such as Music (Reid, 1997), Physics (Booth and Ingerman, 2002) and Statistics (Reid and Petocz, 2003). It has also been used to analyse academic's approaches to teaching (Bruce & Gerber, 1995, Trigwell & Prosser, 1997, Trigwell, 2000).

Within computing, phenomenography has been used to analyse student's conceptions of TCP/IP (Berglund, 2005), Object Oriented Information System Development (Box and Lister, 2005), Learning to Program (Booth, 1992, Booth, 2001, Bruce *et al.*, 2004, Stoodley *et al.*, 2004, Eckerdal & Thun, 2005), and Information Systems (Cope, 2003). Phenomenography has also been used to analyse approaches to teaching computing topics in general (Lister *et al.*, 2007) and the teaching of Data Structures (Lister *et al.*, 2004).

## 2 Method

### 2.1 Interviewee Background

We interviewed nine academics in the Faculty of IT at the authors' university. The faculty consists of three departments. These are Information Systems, Software Engineering and Computer Systems. In order to obtain as broad a sample of views as possible, our interviewees were drawn from all three departments. Two came from Information Systems, five from Software Engineering and two from Computer Systems. A number of other academics were approached, but declined to be interviewed. Of the nine academics interviewed, three made moderate to extensive use of Unix in the courses they taught. Of the remaining six interviewees, four used it occasionally and the remaining two never used Unix at all in their teaching. All academics interviewed had some Unix experience either as undergraduates, postgraduates, in industry or in teaching. In some cases the exposure had occurred some time ago. For example, one interviewee had used the "vi" editor and some Unix commands to teach Cobol Programming over 20 years ago.

### 2.2 Interview Structure

Following standard phenomenographic procedures, the interviews were semi-structured and used the following question set. This had been prepared prior to the interviews.

1. Tell me about your experience with Unix.
2. What does the word "Unix" mean you to you?
3. In what ways are Unix and Microsoft Windows different?
4. In what ways are Unix and Microsoft Windows the same?
5. Is there any task for which you'd prefer to use Unix over Microsoft Windows?
6. Does Unix have any role in the subject you teach? If so, what is that role?
- 7 (a). What do you think is the role of Unix in an undergraduate IT degree?
- 7 (b). What do you think is the role of Unix in an undergraduate Computer Science degree?
- 7 (c). What do you think is the role of Unix in a postgraduate IT degree?
8. What do you think is the role of Unix in computing in general?
9. What do you think is the the role of an operating system, whether it be Unix, Windows, or any other operating system?
- 10: What do you understand by the Unix term "process"?
- 11: What do you understand by the term "file system"?
- 12: What type of tasks would you prefer to use command line for instead of a GUI?
- 13: What do you understand by the term "script"?
14. In what ways are scripting languages and application level programming languages different?
15. In what ways are scripting languages and application level programming languages the same?



16. What do you understand by the term "pipe"?

As part of the semi-structured interview process, the interviewer often asked follow-up questions immediately after individual prepared questions. This was done to illuminate interesting issues arising from the answers to the prepared questions. Approximately 70% of the questions were follow-up questions.

### 2.3 Analysis Technique

The data was analysed using standard phenomenographic techniques. In terms of the categorisation of phenomenographic approaches by Ackerlind (2005) our analysis used the following approach:

- (1) We considered excerpts from transcripts.
- (2) The first author analysed the data initially.
- (3) The two authors then analysed the data jointly. This analysis focussed on attempting to resolve the initial independent interpretation of the first author and possible other interpretations of the data.
- (4) The structure of the analysis was driven by the data, but obviously the authors were influenced by their previous phenomenographic study of student conceptions of Unix.
- (5) The focus was on pragmatic validity. That is, the aim of the analysis was to provide insight into the teaching and learning of Unix

## 3 Results Part 1: Conceptions of Unix

### 3.1 Overview

Among the transcripts of the nine academics interviewed, we identified the following conceptions of Unix:

1. Unix as a command line interface.

This first conception consisted of two sub-categories:

- (a) An unrelated set of commands that is hard to learn and use.
- (b) A more powerful alternative to the Windows graphical user interface (GUI).

Two other conceptions of Unix identified were:

2. Unix as a tool for solving certain problems
3. Unix as a resource

These conceptions are discussed in greater detail in the following subsections.

### 3.2 Unix as a Command Line Interface

As summarised above, interviewees who manifested this conception of Unix tended to display one of two positions, discussed below.

#### 3.2.1 An unrelated set of commands that is hard to learn and use

The following excerpts from teachers' transcripts illustrate this first position:

"....but a command line interface which is the thing that really I think of first when I think of Unix, is just intolerable, it's something completely artificial and arbitrary. It's not even as useful as learning ancient Greek...." (A5)

"...Yes, well it also means a real pain in the neck operating system. It uses a command language. It's really annoying to use. It's really obscure. ... The kind of way that it only gives you feedback if anything goes wrong, so you are never quite sure that anything happened." (A9)

#### 3.2.2 A more powerful alternative to the Windows graphical user interface (GUI)

The second position views a command line interface as being closer to the underlying machine, and hence the command line interface has powers unavailable to a GUI:

"[The machine is] ... accessible in the sense that you when you use it, you can get to it, you can drill down to the lowest level [of the machine] if you want to" (A2)

"[Unix is] an operating system that you interact with at a command line level rather than a GUI. You interact with [the computer] more directly than using something like windows which has a GUI on top of it." (A3)

The above two interviewees were aware of the existence of Graphical User Interfaces for Unix, such as X windows, but these interviewees had a conception of Unix as a very effective operating system because it could be used at the command line level.

### 3.3 Unix as a tool for solving certain problems

Several academics saw Unix as possessing attributes that made it a useful tool for solving problems. These academics sometimes illustrated the power of Unix by describing why they chose Unix to solve problems within their own teaching. For example:

"The software that I have written for taking student submissions and stuff has probably been a lot easier to write, its command line driven, but it's probably been a lot easier to write than it would be if I had written it under some windows environment. ... from a systems administration point of view very well, [Windows is] very awkward, whereas Unix is a lot more straightforward. There are still things I've had to find out with Unix which has frustrated me at times because I've had to figure my way around them, but I suspect I would have had a lot more trouble if I had to try and do this stuff under windows." (A1)

### 3.4 Unix as a resource

The conceptions of Unix that we have described up to this point have been technical in their orientation. A non-technical but common conception is Unix as a resource. This conception focuses on useful properties of Unix that can be appreciated without necessarily having a strong technical background in Unix – a management perspective. Such properties are: robustness, speed, security, server hosting capability, networkability and cost (the last at least for open source versions).

[Unix] has a better reputation for security and performance." (A4)

"[Unix] ... for me it means reliability." (A6)

"Well, if you think about Linux, and people want to save a bit of money, maybe it's got a role in organisations that don't want to spend a huge amount of money on their software...." (A5)

### 3.5 Conceptions of Unix: The Outcome Space

In the previous three subsections we have described four categories in which the interviewees conceived of Unix. We will now look at how these categories relate to each other, to form an Outcome Space.

Three of the categories are technical in their orientation and form a hierarchical relationship. Among these three hierarchical categories, the lowest is the conception of Unix as a weakly or totally unrelated set of commands. Above that conception is the conception of Unix as a powerful command line interface which is available as an alternative to GUIs. The higher of these two conceptions differs from the lower because the higher category introduces a more unified view of the commands, as the command line interface is seen as offering better access than a GUI to the underlying machine.

The third and highest conception in this hierarchy is the conception of Unix as a tool for solving certain problems. In this highest conception, the degree of relatedness between the Unix commands is so great, the conception is of a single, unified tool kit. In this highest category, the focus has shifted away from the command line interface itself, to the problems that can be solved with the command line interface.

This hierarchy of three technical conceptions can be interpreted in terms of the SOLO taxonomy (Biggs & Collis, 1982). The type of interviewee response that illustrates the lowest conception is a unistructural response. The type of interviewee response that illustrates the intermediate conception is a multistructural response, while the highest conception manifested in a relational response.

At this stage of the project, with the interview data currently available to us, the fourth and non-technical category – Unix as a resource – cannot be related to the hierarchy formed by the three technical categories.

## 4 Results Part 2: Why Teach Unix?

All nine interviewees thought that learning Unix should be compulsory, at least for Computer Science students. Seven of the nine thought Unix should also be a compulsory part of an IT degree. In our interview script, we did not explicitly pursue the issue of just how detailed a treatment of Unix should be taught in such degrees.

The reasons cited for teaching Unix fell into two broad classes.

- Practical
- Pedagogical

The difference between these two categories is that the practical category focuses upon the computing environment in which the student will eventually work, whereas the pedagogical category focuses upon the student, and the intellectual development of the student. These understandings are not mutually exclusive, and interviewees frequently manifested both understandings.

Each understanding is described in greater detail in the following two subsections.

### 4.1 Practical Reasons

In this category, the understanding of the purpose of teaching Unix is that it is an essential skill for computing professionals:

*“Certainly I think that as a graduate student, if the company took them out and stuck them in front of a Unix terminal, the students should be able to go ‘OK, I’m not terrified of this...’ ” (A1)*

*”So I guess, at least if students have enough of a flavour of it so they don’t get out there and say ‘Oh! What’s Unix?’ when they went to a Unix shop, because that would make them look a bit silly. And I guess the other thing is... [any organization] has to have a main operating system that they use .... if it happens to be Unix in the place that they work in then they probably should know about it.” (A9)*

The fact that Unix is used widely on the internet was given as a more specific reason why Unix should be taught. For example:

*“If they are going to understand the internet and a lot of the internet functionality works, they will need to understand how Unix works. To understand a lot of the hardware, everything from routers to switches to hubs that makes the internet operational, they will need an appreciation of the kind of programming that needs to be done, the kind of operating systems that are tailored, some of them based on Unix variations that are installed on those systems.” (A9)*

### 4.2 Pedagogical Reasons

Unix, it is argued in this conception, expands the student’s horizons. Unix is seen as an alternative model to Microsoft Windows, with which the students are more familiar. The interviewees were not necessarily hostile towards Microsoft Windows or GUIs in general, but they argued for breadth in student education:

*”I think they also need to have diversity in operating systems ... [so that] ... they don’t think the world just consists of windows. That there are other operating systems” (A2)*

*”... I wouldn’t see Unix as the primary vehicle for teaching, but for developing a deeper understanding of what they are doing at that level of Graphical User Interface, and then they see the result of that under the hood. It’s important to create that understanding, - how computers work, what’s happening in there ...” (A7)*

*”I suppose there are two issues. One would be as a kind of example or historical kind of thing to say this has been a very influential type of operating system and these are what some of the features are and this is why it’s so popular at the feature level, and these are where its’ shortcomings are, that other people might want to add things in.” (A9)*

#### 4.2.1 Knowing “what’s under the hood”

One interviewee expressed the view that computer science graduates, but perhaps not IT graduates, should have an appreciation of the internals of Unix:

*“In a CS degree I think the role of Unix is still major, and it would be very important to talk about the architecture of Unix ... a famous book [see Bach, 1986] .... describes*

*everything that happens inside Unix, the Unix kernel, how it works, why it uses certain data structures, how does it pass those data structures in an efficient manner, how does it schedule processes .... In a Computer Science degree I think this would be a major role. In Information Technology sort of degree you would have to consider that students might like to use it, not necessarily understand it.” (A4)*

Another interviewee took the importance of “knowing what’s under the hood” even further:

*“[Open source software] gives students the opportunity to be exposed to a lot more software at the code level ... whereas they are unlikely to look at the code of proprietary [software].” (A3)*

We list this open source argument here, under pedagogical reasons, because this particular argument for open source does not rest upon the software being free, but on the educational opportunities that flow from having access to source code.

## 5 Discussion

### 5.1 Sample Size

Inevitably, a good portion of the readers will be troubled by the small sample size for this phenomenographic study (i.e. nine academics).

Phenomenography is a qualitative research method, not a quantitative method. From the data presented, it would not be appropriate to speculate upon the popularity among academics of any of the above categories. To make such conclusions would require significantly more data and a different research method. The aim of phenomenographic research is merely to capture the full spectrum of diversity, not quantify it.

While small sample sizes may be compatible with some qualitative methods, in the study presented in this paper we do not claim that nine interview transcripts is sufficient for final conclusions to be drawn. It is possible to perform a preliminary phenomenographic analysis with only nine interviewees, and we present our results as preliminary. We make no claim to have identified, at this stage, the full spectrum of diversity in academics’ conceptions of Unix. However, while interviewing more subjects may elaborate upon our preliminary analysis, we are confident that further data will not invalidate this preliminary study. That is, interviewing more academics will probably add more categories, add further structure to the outcome space, and perhaps refine the category definitions, but collecting more interviews is unlikely to completely invalidate the categories and outcome space as we have identified it in this paper. Having collected a subset of the data we will eventually collect, we believe our existing analysis captures a subspace of the outcome space we will eventually construct.

One strong reason for having confidence in this preliminary analysis is that the results are compatible with our earlier phenomenographic analysis of student conceptions of Unix. We compare the results of these two studies in the next subsection.

### 5.2 Relationship to prior study of students

In both this study of teachers and the authors’ previous study of students (Doyle and Lister, 2006), interviewees clearly articulated the same category “Unix as a resource”. Also, both sets of interviewees articulated a linear hierarchical set of technical conceptions. Both of these hierarchies contained three conceptions of Unix. However, while the actual categories within the two hierarchies are similar, the categories are not identical. Table 1 summarises and compares the linear hierarchical set of technical conceptions from the two studies. The remainder of this section compares these conceptions in detail.

Sophistication	Teachers	Students
High		A professional computing environment
	A tool for solving certain problems	
Low	A more powerful alternative to the Windows GUI	
	An unrelated set of commands	

**Table 1: A comparison of the linear, hierarchical technical conceptions of Unix, for the teachers in this study and the students from the prior study**

The bottom and least sophisticated conception of Unix is the same for both teachers and students – Unix as an unrelated set of commands.

In the teachers’ conceptions of Unix, the intermediate category (a more powerful alternative to the Windows GUI) is not apparent in the transcripts of the student interviews. We suspect this is because this academic conception rests on the notion of an underlying machine, but the students (who are in their first year of study) know very little about the underlying machine.

Both teachers and students share the next level in the hierarchy (Unix as a tool for solving certain problems), but that is the highest category for the academics, whereas the students show another conception, Unix as a professional computing environment. We have, for this paper, tentatively placed it as a higher category, but at this time we do not understand this category well. Some students transcript excerpts that we placed in this category are:

*“I think Unix and Linux is more powerful. .... It’s more professional than Microsoft ... I discovered a new world of computing in Unix. ....” (S01)*

*“I am a systems administrator, and I used to use Microsoft based, and we have, you know, so many problems, with Microsoft, if you are a system administrator. Unix now, opens, I think a new track for me, to deal with system administration using Unix, and now I am planning to study Unix systems administration next semester, because I would like to be a Unix systems administrator. Because I like ...*

[Unix] ... so much. I think it's powerful, and will develop my future career in systems administrator.” (S03)

It may be that these students are articulating what is really the same conception as the top academic category, but they express it this way because are focussed on their chosen future in industry.

While there may be some difference in the categories and outcome space identified in the two studies, the results are very similar. This is not surprising. The academics and students interviewed are all from the same university. Therefore, the student conceptions reflect those of their teachers.

### 5.2.1 Unix as a resource

As part of that prior study of students, the authors wrote the following:

*“At this stage of the project, it appears that students do not connect the category “Unix as a resource” to the other three categories. For example, the students appear to see the superior security of Unix as an “accidental” property of Unix, not a consequence of the architecture of Unix. Perhaps, as we collect more interview transcripts, we will see students who do articulate such a connection. On the other hand, perhaps such a connection is not currently being articulated by the teachers”.*

In our interviews with academics, with only one exception, the academics did not articulate such a connection to us. It seems likely, therefore, that these teachers also do not articulate such a connection to their students.

### 5.3 Validity and Reliability

As with all phenomenographic studies, the categories that we have inferred from our data are probably not the only categories that can be inferred from the data. However, if we presented both our interview transcripts and our categories of description to other phenomenographers, they should agree that our categories of description can be inferred from our data. This is known as communicative validity (Ackerlind 2006)

The reason why alternate sets of categories are possible is that the categories identified in any phenomenographic study are to some extent dependent on the intent of the phenomenographer. Our intent is to facilitate debate on the teaching of Unix, and we chose our categories accordingly. In formal terms, our aim is to produce results which exhibit pragmatic validity (Ackerlind 2006). Our research aim is to provide insights that may be used in the design of courses on Unix, and we believe the results we have presented fulfil the criteria for pragmatic validity.

### 5.4 Relation between Conceptions of Unix and the purpose of teaching it

It is reasonable to expect that there is a relationship between how academics’ conceive of Unix, and how they understand the reasons for teaching it. For example, it seems reasonable that an academic who tends to see Unix as an unrelated set of commands might also be drawn to pragmatic understandings of why it should be taught, and not drawn to believe that students need to understand

“what’s under the hood”. This may be the case, and further research may confirm that hypothesis, but at this time there is insufficient data to confirm or refute such a conjecture.

## 6 Conclusion

In this study of why Unix is taught, two broad categories of reasons were identified from interviews with academics – practical and pedagogical. Given the small sample size (even by the standards of phenomenographic research) we present these findings as preliminary. In future work, we will be continuing the same form of phenomenographic analysis with a larger and more diverse pool of interviewees. Furthermore, some of these interviewees will be systems programmers and other people who use Unix in their employment. If circumstances permit, we may also interview academics at other institutions. By interviewing a larger and more diverse set of people, we hope to capture a rich picture of peoples’ conceptions of Unix, and the reasons why it should be taught.

Beyond unix, this paper demonstrates how phenomenography can be used as a tool for syllabus design in general. It can be used to define various positions, before debating the pros and cons of the positions. Meetings that debate the design of new degrees are highly charged emotionally. Academics who are not inclined to join such a difficult debate can be encouraged to articulate their position as part of an early, non-confrontational, data gathering phenomenographic study. Beginning with a phenomenographic study may therefore lead to a more inclusive and comprehensive approach to syllabus design in general.

## References

- Åkerlind, G. (2005) Variation and commonality in phenomenographic research methods. *Higher Education Research & Development*. 24(4): 321-334.
- Astrachan, O., Bruce, K., Koffman, E., Kölling, M., Reges, S. (2005) “Resolved: Objects Early Has Failed”. SIGCSE’05, February 23-27, 2005, St. Louis, Missouri, USA.
- Bach, M. J. (1986) *The Design of the UNIX Operating System*. Englewood Cliffs, NJ: Prentice-Hall. ISBN: 0132017997.
- Berglund, A. (2005) *Learning computer systems in a distributed project course: The what, why, how and where*. Acta Universitatis Upsaliensis, Uppsala Dissertations from the Faculty of Science and Technology 62. ISBN 91-554-6187-5.
- Biggs, J. B. & Collis, K. F. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. New York, Academic Press, 1982.
- Booth, S. (1992). *Learning to program: A phenomenographic perspective*. PhD thesis, University of Gothenberg, Sweden.
- Booth, S. (2001) Learning to Program as Entering the Datalogical Culture: a Phenomenographic Exploration. In *9th European Conference for Research on Learning and Instruction (EARLI)*, Fribourg, Switzerland.

- Booth, S. and Ingerman, A. (2002) Making sense of Physics in the first year of study. *Learning and Instruction* **12**: 493-507
- Box, I., & Lister, R. (2005). Variation in students' conceptions of object-oriented information system development. In O. Vasilecas, A. Caplinskas, W. Wojtkowski, W. G. Wojtkowski, J. Zupancic & S. Wrycza (Eds.), *Information systems development: Advances in theory, practice and education. Proceedings of 13th international conference on information systems development (ISD 2004) Vilnius, Lithuania, September 9-11 2004* (pp. 439-451): Springer.
- Bruce, C. and Gerber, R. (1995) Towards university lecturers' conceptions of student learning. *Higher Education*, **29**, 443-458
- Bruce, C, Buckingham, L, Hynd, J, McMahon, C, Roggenkamp, M, and Stoodley, I. (2004) Ways of Experiencing the Act of Learning to Program: A Phenomenographic Study of Introductory Programming Students at University. *J. of Information Technology Education*, **3**: 143-159. <http://jite.org/documents/Vol3/v3p143-160-121.pdf> [accessed May 2005]
- Bruce, K. (2005) Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list. ACM SIGCSE Bulletin. Volume 37, Issue 2 (June 2005) 111-117.
- Cope, C. (2002) Seeking Meaning: The Educationally Critical Aspect of Learning About Information Systems, *Proceedings of the Informing Science + IT Education Conference, Cork, Ireland*. <http://proceedings.informingscience.org/IS2002Proceedings/papers/cope190seeki.pdf> [accessed Apr. 2006 ]
- Doyle, B. and Lister, R. (2006) A Preliminary Phenomenographic Study Concerning Student Experiences of Unix. *Proceedings of the 19th Annual Conference of the NACCQ*. Wellington NZ 7<sup>th</sup>-10<sup>th</sup> July 2006 pp 73-78
- A. Eckerdal, A. & Thun, M. (2005) Novice Java Programmers' Conceptions of "Object" and "Class", and Variation Theory. *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pp. 89-93.
- Kutay, C. and Lister, R. (2006) Up Close and Pedagogical: Computing Academics Talk about Teaching. *Conferences in Research in Practice in Information Technology*, **52**.
- Lewis, T., and Smith, W. (2005) The Computer Science Debate: It's a Matter Perspective. SIGCSE Bulletin. Volume 37, Issue 2 (June 2005) 80-84.
- Lister, R., Box, I., Morrison, B., Tenenberg, J., Westbrook, S. (2004) The Dimensions of Variation in the Teaching of Data Structures. *9th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Leeds, UK, 28-30 June. pp.
- Lister, R., Berglund, A., Box, I., Cope, C., Pears, A., Avram, C., Bower, M., Carbone, A., Davey, B., de Raadt, M., Doyle, B., Fitzgerald, S., Mannila, L., Kutay, C., Peltomäki, M., Sheard, J., Simon, Sutton, K., Traynor, D., Tutty, J., Anne Venables, A. (2007) Differing Ways that Computing Academics Understand Teaching. *Conferences in Research in Practice in Information Technology*, **66**.
- McCauley, R. (2004) Thinking about our teaching. ACM SIGCSE Bulletin, Vol. 36, Issue 2 (June), 18-19
- Reges, S. (2006) Back to Basics in CS1 and CS2. *Proceedings of the 37th Technical Symposium on Computer Science Education (SIGCSE 2006)*. Houston, Texas, USA. pp. 293-297.
- Reid, A. (1997), The Meaning of Music and the Understanding of Teaching and Learning in the Instrumental Lesson, in *Proceedings of the Third Triennial ESCOM Conference*, ed. A. Gabrielsson, Uppsala, Sweden: European Society for the Cognitive Sciences of Music, **3**, 200-205.
- Reid, A. and Petocz, P. (2003) Completing the Circle: Researchers of Practice in Statistics Education. *Mathematics Education Research J.*, **15**(3): 288-300.
- Stoodley, I. Christie, R. and Bruce, C. (2004) Masters Students' Experiences of Learning to Program: An Empirical Model. *Proceedings of QualIT2004: International Conference on Qualitative Research*. <http://sky.fit.qut.edu.au/~bruce/pub/papers/QualIT2004-Bruce.pdf> [accessed October 2006]
- Trigwell, K. (2000) *Phenomenography: Discernment and Variation* [http://www.learning.ox.ac.uk/files/Phenom\\_ISL\\_paper.pdf](http://www.learning.ox.ac.uk/files/Phenom_ISL_paper.pdf) [accessed Mar. 2006 ]
- Trigwell, K. and Prosser, M. (1997), Towards an Understanding of Individual Acts of Teaching and Learning (1997) *Higher Education Research and Development*, **16**(2): 241-252.



# Software Development Marketplaces—Implications for Plagiarism

Daryl D'Souza

Margaret Hamilton

Michael C. Harris

School of Computer Science & Information Technology  
RMIT University,  
PO Box 2476V, Melbourne, Victoria 3001,  
Email: {djds, mh, miharris}@cs.rmit.edu.au

## Abstract

Plagiarism of programming assignment solutions can be detected via a range of plagiarism detection tools, as long as the originally authored work is accessible electronically. For some time now, our school has used plagiarism detection software over all submissions of programming assignments in selected courses. The use of such software has provided a viable mechanism for catching such cheating by copying. However, the emergence of online software development websites now enables students to purchase solutions from software contractors. Students submit their assignment specifications to such websites and receive bids from potential contractors to develop coded solutions. In these contexts the use of copy detection software is rendered useless, as the solutions are custom written and there is usually no electronic source available against which similarities may be detected. This paper addresses the cheating problem of students purchasing solutions via websites that host software development marketplaces.

**Keywords:** Plagiarism, discipline, software development marketplaces

## 1 Introduction

Plagiarism can be defined as the practice of intentionally or inadvertently claiming someone else's work or ideas as one's own, without acknowledgement. Plagiarism by university students is a serious problem, which our school of Computer Science & Information Technology has played a leading role within RMIT University in addressing, by establishing a comprehensive set of plagiarism-management procedures (Zobel & Hamilton 2002). These procedures have been successful in increasing student and staff awareness, providing deterrents, enabling access to software detection mechanisms, introducing procedural consistency in dealing with all cases of plagiarism, and enabling appropriate linkages with university procedures dealing with appeals and exclusions. This in turn has led to a range of further initiatives by the university and schools to inform students about plagiarism.

Our school, for instance, informs students in every first lecture of each semester about plagiarism, how it may be avoided when working together, and the consequences of non-compliance with policy. Additionally, the school invites all commencing students

to workshop-oriented induction sessions (Hamilton, Tahaghoghi & Walker 2004). Such initiatives have already had an impact in terms of awareness of plagiarism, reducing the incidences of plagiarism, and in improving the image of the school. We are recognised as a school that takes the issue of plagiarism seriously.

During the time the school's plagiarism-management procedures were being established, Zobel (2004) uncovered a case in which an external private tutor was involved in selling assignment solutions to students who sought tutoring services. The discovery of this case, denoted as the *mytutor* case, highlighted a form of cheating that could not be detected during manual assessments, unless the same solution had been sold to multiple recipients in the same class and happened to be given to the same marker. As large numbers of students were enrolled in the course<sup>1</sup> and different tutors marked the assignments, without the use of plagiarism detection software and the fact that the same solution was submitted by different students, this case might not have come to light.

A similar practice has recently emerged over the Web, with students evidently accessing a range of services to seek, for example, custom solutions to their programming assignments. There have, of course, long been opportunities for students to seek online help for their assignments from third parties. Examples include postings to newsgroups, mailing lists and forums seeking answers to coding problems set as assessment, without revealing such intent. Fortunately, the purpose of these requests has often been obvious to the experts targeted, and as such they have usually been ignored or dismissed.

There is also an enormous amount of code available, free of charge, on the Internet, but unless the piece of assessment is particularly generic, finding an appropriate match is difficult, and modifying existing code to meet the requirements of an assessment specification often takes as much work and programming expertise as writing the code from scratch.

The scenario of hiring a private tutor to complete assignments is increasingly well-supported by the emergence of "eBay-like" online auction websites. Such websites allow software contractors to bid for client requests for code solutions. While many of these websites have a legitimate purpose, we have experienced their use by students to purchase solutions to assignments involving programming. Recent incidents have prompted us to examine more closely the practice of students accessing websites that provide software solutions, and to focus on their impact on plagiarism procedures in use within our school.

<sup>1</sup>In the context of our university, a "course" refers to a unit of study, commonly called a subject, toward an accredited "program", commonly called a course.

Our objective is to strengthen our school plagiarism-management procedures. Hence, this paper explores the implications to plagiarism management in the context of software development websites. In Section 2 we summarise the plagiarism-management procedures within our school and review the problems associated with the private tutor scenario in the context of these procedures. In Section 3 we report two cases that highlight the difficulties associated with identifying and penalising students who cheat, in situations where students have accessed online auction websites to buy solutions. In Section 4 we present a survey of online auction websites, in order to arrive at a comparative framework for such services. We propose additional requirements for consideration in the context of existing plagiarism management procedures, and suggestions for avoiding plagiarism. Finally, in Section 5 we review the main contributions of this paper, we summarise our findings, discuss strategies to accommodate future transgressions of this nature in existing plagiarism-management procedures, and reflect on future work directions.

## 2 Background

One approach to addressing the code plagiarism problem is to *detect* copies of work submitted by students. In our school, code plagiarism is addressed by using copy-detection software to detect copying within selected courses. For a given course, the detection software carries out pairwise comparisons of student assignment submissions, tagging each pair with a percentage similarity measure that represents the strength of duplication in each pair of submissions.

There are several software copy-detection packages available on the Web for essays and computer code. Among some of the most popular packages are *Turnitin*<sup>2</sup> and *plagiserve*<sup>3</sup> for essays and *jplag*<sup>4</sup> and *MOSS*<sup>5</sup> for computer code copy-detection. Within our school, code copy-detection algorithms have attracted research interest, such as the work by Burrows (Burrows, Tahaghoghi & Zobel 2004), which evaluates their own detection algorithm against other, well-known and widely-used systems. They define code plagiarism as “the unauthorised reuse of program structure and programming language syntax”. This reflects the typical practice of partially or completely copying others’ code and claiming it as one’s own. The definition implies that attempts to catch plagiarists need to place a greater emphasis on detection of copying of code from others. To date, code copying from peers has been the dominant form of plagiarism by our students. It is also the easiest form of plagiarism to catch, because source and copied material are both available. It is such an emphasis that drives the plagiarism management procedures within the school.

Currently, the school uses *jplag* for code copy-detection. Once code copying has been detected, other steps in the school’s plagiarism-management procedure are enacted. This procedure is a comprehensive one, and described in detail elsewhere (Zobel & Hamilton 2002). Outside the procedure itself, the school has established a regular student workshop (Hamilton, Tahaghoghi & Walker 2004), delivered at the beginning of every semester to newly enrolled students, and to returning students who have not previously attended. During the workshops stu-

dents are briefed about staff expectations for assignments and workloads, and students discuss various scenarios, based on examples of actual cases of plagiarism. Examples of the related issues addressed include time management, workload expectations of hours spent outside timetabled classes, gathering resources and becoming familiar with course requirements and assessments, possibilities and implications of hiring private tutors, and acceptable practices of tutoring requirements.

At the workshop, we also explain the range of penalties incurred for plagiarism within the school and at university level disciplinary hearings. It is recognised that there must be deterrents and penalties on the one hand, and education on the other. The workshops do not attract study credits so attendance is optional. While the workshops are only two hours’ long and our expectation is that most students will attend them, for various reasons some students do not attend.

When a student is suspected of plagiarism, they are informed of the evidence, provided with a copy of the regulations, and asked to attend a disciplinary hearing within the school. They are encouraged to bring any support person along to the hearing, as well as any supporting evidence, such as earlier versions of their work showing the progression and development of their final solution. The Head of School or their designate chairs the Hearing, which is also attended by the Lecturer, who provides the Head with any supplementary information and presents the evidence of plagiarism. The chairperson is responsible for determining and recording the outcomes of all hearings. Students are notified of such outcomes within a few working days. Only if plagiarism is deemed to have taken place, are the student details recorded and passed to the university discipline board. Procedures to manage plagiarism exist for most universities, but it is interesting to note that at our university, the second or further hearings do not occur at portfolio<sup>6</sup> level, but at the higher university level. This also covers cases where students may plagiarise in courses in various different schools within the university. Such a process allows for greater consistency of plagiarism management and outcomes across the university.

In addition to the school’s own plagiarism management process, the university subscribes to Turnitin<sup>7</sup>, as do thirty-five other Australian universities, and all the universities in the United Kingdom. According to the Turnitin website, Turnitin is:

Recognized worldwide as the standard in Plagiarism Prevention. Turnitin instantly identifies papers containing unoriginal material and acts as a powerful deterrent to stop student plagiarism before it starts.

Turnitin is an initiative to deter plagiarism from sources available on the Web, whether from essays previously written or modified from paper mill databases, or from various lecturers’ or past students’ websites. By using Turnitin it is possible to compare student text submissions with the large Turnitin database of resources, such as those available in online paper mills. Online paper mills are websites where students can download essays, either freely or for a fee. According to the Wikipedia definition of a paper mill<sup>8</sup>:

<sup>2</sup>turnitin.com

<sup>3</sup>www.plagiserve.com

<sup>4</sup>jplag.de

<sup>5</sup>www.cs.berkeley.edu/~aiken/moss.html

<sup>6</sup>At our university, portfolios are super faculties and the term “portfolio” has replaced “faculty”.

<sup>7</sup>turnitin.com

<sup>8</sup>en.wikipedia.org/wiki/Paper\_mill\_(essays)



Online paper mills usually contain a large, searchable database of essays. Most paper mills today offer customized writing services, usually charging by the page. Some sites now even offer ready-made college application essays from applicants who have been accepted.

While Turnitin does increase the ability of universities to detect plagiarism, it is not able to detect all text plagiarism, as it is now increasingly possible to purchase custom-written essays. According to a recent newspaper article<sup>9</sup>, some businesses are earning large sums of money writing custom assignments.

Hence copy-detection is applicable only where original electronic copies are available. Likewise, code copy-detection is possible only because procedures are in place to detect high similarity between students' assignment submissions. As we have observed recently in the school, code solutions are also being sourced via external sites; in this sense, *mytutor* has been replaced by its web-based equivalent: *e-tutor*. In the following section we discuss case studies, to exemplify this emerging trend.

### 3 Case Studies in Software Development Marketplaces

We report two cases of cheating that involved the purchase of software solutions for assignment work via the Internet. In both cases the RentACoder website<sup>10</sup> was used as a marketplace for the purchase of assignment solutions. Quoting from this site:

[RentACoder] is an international marketplace where people who need custom software developed can find coders in a safe and business-friendly environment. Buyers can cherry pick from a pool of ... coders ... enabling them to hire a coder across the country or across the globe ... from the comfort of their computers. Buyers who wish to hire internationally, can take advantage of favorable overseas exchange rates, resulting in work being done for 50-90% less than if the project were done in-country. Coders are also given access to a huge pool of potential work and have the ability to work independently from their homes rather than for a company.

In one online news bulletin, RentACoder is described as an "eBay for contractors"<sup>11</sup>, an online auction website at which software development services may be bought and sold. We will refer to such sites as *software development marketplaces* or SDMs. An SDM provides software contractors (developers) a forum in which to bid for software development work, in response to buyers who submit requests for code solutions. Further details appear in our survey of SDMs in Section 4. Potential buyers and contractors (or developers or bidders) register with the site. Purchasers present their requests (project descriptions) in an open market which is accessible to registered bidders. A successful bid, in which the purchaser is satisfied with the bidders rating and price, closes off the request.

The first reported incident of the use by students of the RentACoder SDM was in relation to a program

being delivered by one of our university's offshore partners.

The course manager in Melbourne discovered a request for a solution to an assignment, which formed part of the assessment in this course. The request appeared in the bidding list and, it was conjectured by the course manager, that any student with a foundation background in image processing, or who had previously passed their course, would find the assignment relatively easy. As such, there could be many potential bidders. The final bid accepted was for the sum of US\$200.

E-mail addresses of buyers are protected within RentACoder. However, it was discovered that separate requests had been posted for solutions for assignments in other courses in two previous study periods. A further investigation narrowed down the number of potential, offending students to four. At this point, an approach to RentACoder was considered. Examination of the site revealed a policy reference to copy-right, but not to plagiarism.

Several actions were pursued. Our offshore partners were alerted and the offshore lecturer for the course was asked to wait for the student submissions. They were advised to look for any similarities in the submissions received. It was also suggested that the bidding site operators be contacted, via the university's legal department, on the premise that the posting of the assignment specification represented a breach of copyright as the specification had been published without permission. Furthermore, it was suggested that attempts be made to reveal to the university, the identity of the student who submitted the request to their site, and to supply a copy of some or all of the code developed by the successful bidders. Such information would confirm that our assignments were posted, and the code could be used to establish a match with the submitted solutions.

Advice received via the registrar's office and the university legal office led to steps being taken to minimise further breaches by informing students about their responsibilities in terms of cheating, via distribution and prominent display of appropriate notices. Furthermore, the lecturers in question were advised to charge students under the university discipline regulations guidelines, if sufficient evidence was available that the students had cheated through bids to develop solutions. The discipline regulations provide for appointments of persons at offshore locations, to act in the roles of "referee" and "reviewer", to hear cases and to apply penalties.

Given that uncertainties remained, the only feasible course of action we could take was to interview the four students, who were potential offenders, as per prevailing plagiarism-management process. It was agreed that although this process was only of deterrent value, there might still be some hope that additional information about the cheating might be revealed. In terms of a future strategy for assignment development, it was suggested (by the course coordinator) that the requirements be made dependent on non-portable infrastructure, for instance, to make the solution dependent on locally developed code, situational knowledge, delivered in stages, and with key parts to be delivered only during lab sessions. It was suggested that other context dependent requirements could also be adopted.

The students who were interviewed at the offshore location all categorically denied knowledge of the website in question. They also denied knowledge of any such practice. RentACoder did not offer any assis-

<sup>9</sup>education.guardian.co.uk/schools/story/0,,1832872,00.html

<sup>10</sup>rentacoder.com

<sup>11</sup>builder.com/5100-6390-1049913.htm

tance in the matter, stating in their responses that their clients were clearly instructed via their website to use the site responsibly. They claimed that any breach of copyright law was the responsibility of the individuals who posted copyrighted documents. Our plagiarism-management procedures, which are premised on copy-detection and not on monitoring of SDM access, were not adequate in these circumstances.

In the second case, an academic from an overseas university notified a tutor in our school about the presence of an assignment specification posted on the RentACoder website. The original content of the assignment specification document had been left intact, thereby making it easy to identify source and contact details. The academic provided the link to the site and the details of who had posted the request.

Closer investigation revealed that the assignment was posted by an enrolled student, for whom the assignment represented an assessable task, and for which a successful bid had been made for a coded solution to be developed. In other words, the RentACoder site was used to pay for a completed solution to the assignment.

The student left an important clue when posting the assignment to the RentACoder site; the username under which the assignment was posted. For the purposes of this paper we will give the username as *codebuyer*. As it happened, *codebuyer* had posted in various places on the Web. These posts included a discussion of the intention to register a particular domain name, links to a website mock up, and links to a business website. Domain registration details, an email address on the mock up website, and details in the Australian business register all independently led to a student currently enrolled in the programming course where the assessment task was set.

When called to attend the school disciplinary hearing, the student was honest and open about posting the assignment specifications to the RentACoder website. *codebuyer* had not attended our school student workshop which explains some copyright issues, but did not see anything intrinsically wrong in posting the assignment specifications to the Web or in buying the solution. In fact, he was happy to explain that he had found a business niche for himself. He was quite convinced that coding was not really for him, that he would prefer to be a project manager, and outsource all his future coding to bidders from other countries who would charge much less money, and do a better job than he felt he could ever achieve. It turned out that the successful bidder, a coder in Romania, had been paid \$AU35 to write the solution to the assignment.

This case highlighted for us the need not only to clearly explain intellectual property issues about the ownership of assignment specifications, and the ownership of code, but also to request staged submissions. It also suggests the need for a SDM monitoring process that is clearly able to identify the offender. In the case study reported, de facto monitoring occurred by virtue of us being notified by the overseas academic. While evidence did exist linking the suspect to *codebuyer*, proof that the suspect was the offender was revealed by the confession of the offending student. Clearly, this highlights the difficulty in catching plagiarists who access SDMs for solutions, when compared with the copy-detection scenario.

#### 4 Monitoring of online auction sites for software development and suggestions for avoiding plagiarism

Table 1 gives a list of various websites either providing code solutions for user-supplied specifications or allowing access to expert coders. Though there are a number of different models represented, SDMs share certain general characteristics. Each site provides a mechanism for requesting custom code, some means of allowing communication between a buyer and a potential provider, and one or more methods for payment.

The process is started when a request for service is made, where a buyer posts a project's specification, often including a suggested budget. There is then a dialogue between the buyer and potential providers, with the aim of finalising the requirements of the project. SDMs may facilitate this dialogue in various ways, such as by providing public or private discussion forums, or they may simply provide contact details, such as telephone numbers or email addresses, and allow the participants free communication.

While some SDMs offer the simple purchase of coding, the majority offer an eBay-style auction process, with the bidding, which is often described as quoting for service, occurring as part of this dialogue. In these cases, buyers post a project's specification and developers respond by placing bids. When the auction expires, the buyer is able to choose the most appropriate provider, which may not necessarily be the cheapest. While the code is being completed, there may be further communication between the buyer and the provider, either through the SDM or directly.

Finally, once the solution has been completed, the SDM provides a means by which payment can be made to the service provider. In many cases, the amount agreed upon must be put into an escrow account prior to the commencement of work. Alternatively, payment may be made using PayPal, directly by credit card or by cheque.

At some point in this process, the site will have a way of making money. This may be by the use of advertising on the site, by charging a subscription fee to developers, by charging a flat fee for transactions, or by charging a commission on completed projects.

Many SDMs can be used for legitimate purposes, such as the outsourcing of programming projects, while others are clearly aimed at helping students cheat. We also note that, given these websites create a marketplace for competent coders to freelance, many students who access them may be doing so to provide coding expertise rather than to purchase assignment solutions.

As can be seen in Table 1, the list of websites is long yet it is by no means exhaustive. Within our school, we have set up a script to monitor student access to these sites. However, we are aware that students may access SDMs from home or from locations other than our school, and so we cannot monitor this access. Also, as mentioned previously, students may be accessing these sites to provide the code, not pay for code being delivered. Hence this issue of catching students inappropriately accessing SDMs is extremely problematic.

In both cases discussed above, the assignment postings on the websites were found by academics not immediately involved in the delivery of the courses concerned. They themselves were monitoring the sites and checking for their own assignments being

URL	Business Model
12freelance.com	Buyers register and post projects. Freelancers register. Unclear how projects are won.
cityitjobs.net	Buyers post projects, and can search resumes of registered providers. Providers bid on projects.
codelance.com	Buyers post projects, developers bid on projects.
computersciencetutoring.com	Hidden experts provide “help to understand, start or finish an assignment, course explanation or exam review preparation,” by telephone or email.
contractedwork.com	Post project or email job, service providers bid. Contractors pay a subscription to have access to the work.
ecodegenie.com	Hidden experts provide help with programming projects. The implied focus is on assignments.
elance.com	Buyers post projects and developers reply with quotes. Buyers select quotes, not a bidding site.
freelanceauction.com	Buyers register and post projects. Freelancers (programmers, web and graphic designers) register for small fee and pay a 10% commission on successful bids.
freelancecentral.net	Buyers post projects, developers bid on projects.
freelancersdirect.com	Buyers post projects, developers bid on projects.
freelancewebprogramming.com	Buyers post projects, developers bid on projects.
freelancewebprojects.com	Post projects and set an end date for the auction. Freelancers bid on projects. (currently down for maintenance)
getacoder.com	Buyers post projects, developers bid on projects.
www.gotprojects.com	Buyers post projects, and can search details of registered providers. Providers bid on projects.
guru.com	Buyers can post projects which freelancers quote on or search for freelancers directly and ask them for a quote.
kasamba.com	Experts (from a range of disciplines) register, posters search for and hire experts. 30% fee payable by successful experts.
outsourcetoday.net	Buyers post projects, developers bid on projects.
programmingbids.com	Buyers post a project, service providers bid.
projectspool.com	Buyers post projects, developers are charged a fee to bid on projects.
projectspring.com	Buyers post projects, developers bid on projects.
scriptlance.com	Buyers post projects, developers bid on projects.
thecentralmall.com	Buyers post projects, developers bid on projects.
webdevelopersindex.com	Buyers post projects, developers bid on projects.

Table 1: Websites providing specific code solutions.

posted. In both cases, their evidence was timely and provided the necessary impetus to follow the investigation through to the identification of the students involved. Thorough monitoring of all SDMs is desirable, but new sites emerge regularly, and it is often difficult to identify assignments. In most cases, it is only the careless students who leave identifying features which can be traced either to them, or to the assignment writers.

If we consider the reasons for which the coding assessments were originally designed, we can honestly say we would rather spend time monitoring the student progress through developing their coding skills, than sitting through disciplinary hearings, visiting SDMs, or searching for the latest addition to the SDM table. Hence though we now have very strong technical prevention procedures and tools and support available, we find they are not the complete and only answer.

There is a drive within our school to encourage a more pedagogical approach to assignment submission, for staff to design engaging assessments appropriate to the students' level of knowledge, and to encourage students to submit their work earlier in smaller regular staged submissions, as well as to use code versioning systems, such as CVS.

One particular pedagogical initiative one of the authors, D'Souza, is involved with is the adoption of a mentoring scheme, where past students of the same course encourage current students by supporting them and engaging in activities designed to instill confidence and belief in the new students. For their efforts the past students are awarded certificates commending their activities as part of the LEAD project standing for Learn, Engage, Aspire, Develop at RMIT<sup>12</sup>.

You'll meet new people, learn new skills, and be rewarded with training and official recognition on both your academic transcript and a certificate signed by the Vice-Chancellor. Through volunteering, you'll increase your knowledge base and communication skills; network with a variety of students and build employment skills along the way.

We have found many past students who are very willing to support new struggling students especially with C programming courses. Other pedagogical approaches include encouraging tutorial group work on practical projects, and discussing other possibly incorrect or incomplete solutions to problems. We have found students respond well to demonstrating their own work and to commenting on other student's demonstrations. Whether the demonstrations are marked as part of the assessment or simply part of the process of completing the assignment, students are eager to participate.

## 5 Conclusion

In this paper we reflect on a recent and emerging form of cheating by students, that of access to SDMs to obtain solutions to programming assignments on a fee-for-service basis. Bought solutions are nothing new. Private tutors have provided such services and probably still do but SDMs provide far greater opportunities to cheat and flexibility in how clients and bidders may transact with little likelihood of detection. SDMs may have emerged to serve a legitimate purpose offering a fee-for-service medium for software development. However, in a global marketplace, SDMs

represent a potentially lucrative source of income and are likely to proliferate. We suggest this will lead to a potential increase in accesses by students seeking to buy solutions to programming assignments.

We discuss two case studies dealing with cheating that involve access to SDMs for bought solutions to assignments. SDMs bring a new dimension to the way unauthorised solutions are presented as one's own. Copies of submitted work are potentially no longer available, unless by coincidence several requests are won by the same SDM bidder, who submits the same solution to clients drawn from a single cohort of students. We contend that such a scenario is unlikely. However, it may be possible that past students who have already coded similar assignments, may be inclined to try to sell solutions. Our suggestions for avoiding this are firstly to warn new students in an orientation academic integrity workshop, secondly to design fresh, new assignments each semester and thirdly, to engage past students with mentoring the current students in a supervised supportive environment.

In both case studies presented, copy-detection software could not be employed. The cases were revealed as a result of due diligence on the part of academics monitoring SDM sites to check for access by students to buy solutions to their assignments. There is only a small window of time in which the assignment specifications are available for bidding. We encourage all academics to regularly check SDMs with a view to finding any assignments posted there and gathering the necessary evidence to inform any other academics whose assignments may be involved. In our first case presented here in this paper, the absence of any technical investigation at our offshore operation made it impossible to identify the students involved. In the second case study presented, we were successful in finding the student, but only because we had the initial tip-off from the academic overseas with the evidence of the assignment posted, *and* we carried out an extensive and exhaustive technical investigation *and* the student in question admitted accessing the SDM for a solution.

We tabled a list of some commonly accessed SDMs for the interest of others and in the hope that academics might alert each other about potential cheating by students, as we were informed. An ideal scenario might be to have all SDMs monitored on a regular basis to detect the presence of assignments posted by students. Our survey of SDMs revealed that such postings are accessible to registered users, where registration is open to anyone, while they are open for contractors to do their bidding. Once a successful bid is made, the assignment is no longer visible, so the timing and frequency of such monitoring is critical. However, monitoring all SDMs is clearly not feasible if conducted by individuals. Hence we are of the view that a collegiate approach by all schools that teach programming could lead to a inter-varsity approach to addressing the SDM problem. However, we recognise this is a growing issue, and policing is not the only solution, nor are deterrents enough to stop it.

Our recommendations build on those employed to combat other forms of plagiarism, such as the *mytutor* scenario. Students should be encouraged to submit or demonstrate their work regularly, and in stages, so that clear audit (submission) trails are maintained, and authorship is able to be authenticated. Ideally students should incorporate version control systems in their submissions, and keep their own regularly updated versions. However, all programming course instructors should be encouraging staged submissions, and regular discussions between students and tutors

<sup>12</sup>[www.rmit.edu.au/lead](http://www.rmit.edu.au/lead)

or mentors about common problems encountered by all. Trying to support students along the way, before the submission of their assignments, rather than catching them at the end after submission, is the preferred solution.

## 6 Acknowledgments

The authors would like to thank colleagues Suzi Archer, Emil Mikulic, Ron van Schyndel and Steven Burrows.

## References

- Arwin, C. & Tahaghoghi, S.M.M. (2006), 'Plagiarism Detection across Programming Languages.' In Vladimir Estivill-Castro and Gill Dobbie (eds), *Proceedings of the 29th Australasian Computer Science Conference (ACSC2006)*, CRPIT volume 48, pp. 277-286, Hobart, Australia, 16-19 January 2006.
- Burrows, S. & Tahaghoghi, S.M.M. & Zobel, J. 'Efficient Plagiarism Detection for Large Code Repositories.' In G. Abraham and B.I.P. Rubinstein (eds), *Proceedings of the Second Australian Undergraduate Students' Computing Conference (AUSCC04)*, Melbourne, Australia, pp. 9-16, 8 - 10 December 2004. ISBN: 0 975 71730 8
- Burrows, S. & Tahaghoghi, S.M.M. & Zobel, J. 'Efficient plagiarism detection for large code repositories.' *Software - Practice and Experience*. To appear.
- Hamilton, M. & Tahaghoghi, S.M.M. & Walker, C. (2004), 'Educating Students About Plagiarism Avoidance - A Computer Science Perspective', *ICCE2004: International Conference on Computers in Education 2004*, pp. 1275-1284.
- Zobel, J. (2004), "Uni Cheats Racket": A case study in plagiarism investigation, Australasian Computer Education Conference, Dunedin, New Zealand, pp. 357-356.
- Zobel, J. & Hamilton, M. (2002), 'Managing student plagiarism in large academic departments', *Australian Universities Review* **45**(2), pp. 23-298.



# Measuring Improvement in Latent Semantic Analysis-Based Marking Systems: Using a Computer to Mark Questions about HTML

Debra T. Haley, Pete Thomas, Anne De Roeck, Marian Petre

Centre for Research in Computing, Department of Computing  
The Open University  
Walton Hall, Milton Keynes MK7 6AA UK

[D.Haley, P.G.Thomas, A.DeRoeck, M.Petre] [at] open [dot] ac [dot] uk

## Abstract

This paper proposes two unconventional metrics as an important tool for assessment research: the Manhattan (L1) and the Euclidean (L2) distance measures. We used them to evaluate the results of a Latent Semantic Analysis (LSA) system to assess short answers to two questions about HTML in an introductory computer science class. This is the only study, as far as we know, that addresses the question of how well an LSA-based system can evaluate answers in the very specific and technical language of HTML. We found that, although there are several ways to measure automatic assessment results in the literature, they were not useful for our purposes. We want to compare the marks given by LSA to marks awarded by a human tutor. We demonstrate how L1 and L2 quantify the results of varying the amount of training data necessary to enable LSA to mark the answers to two HTML questions. Although this paper describes the use of the metrics in one particular case, it has more general applicability. Much fine-tuning of an LSA marking system is required for good results. A researcher needs an easy way to evaluate the results of various modifications to the system. The Manhattan and the Euclidean distance measures provide this functionality.

**Keywords:** Latent Semantic Analysis, assessment, metrics, measures

## 1 Introduction

Marking essays by computer has been researched since the 1960s (Page, 1967). Educational institutions hope to save time, and therefore, money by using computerised marking systems. In addition to the possible cost savings, the computer offers some advantages over the human. Human markers may mark differently as they become fatigued in addition to being affected by the order of marking. For example, if a brilliant answer is the first read by a marker, it could cause the marker to be harsher for the remainder of the answers. Even the most scrupulous people might show bias based on personal feelings towards a student. While they may successfully avoid awarding better marks to their favourite students,

they may mark unfavoured students more highly than they deserve in an attempt to be unbiased. Automatic markers can be an improvement over human markers because their results are reliable and repeatable. They do not get tired, they do not show bias based on personal feelings towards students, their results will be the same without regard to the order in which the answers are presented, and they are able to return results much faster than humans.

This paper reports on research taking place to improve an automatic, Latent Semantic Analysis (LSA) based marking system. We are using LSA to examine the feasibility of marking short answers about HTML. The HTML questions we are marking are two-part. Part one asks the learner to correct a given fragment of incorrect HTML. Part two asks for a few sentences explaining the error(s) in the original HTML. To our knowledge, this has never been done before. Although LSA researchers have marked essays and short answers, we could find nothing in the literature describing efforts to mark short answers in the very specific and technical language of HTML.

LSA is a theory and method invented in the late 1980s (Deerwester et al., 1990) for information retrieval (IR) and was first known as Latent Semantic Indexing (Furnas et al., 1988). The IR community still refers to LSI whereas the researchers using the technique for other applications refer to it as LSA (Landauer and Dumais, 1997, Foltz et al., 1998, Miller, 2003). Automatic marking of essays and short answers is one application of LSA with great appeal for educational institutions.

Researchers using LSA-based grading systems have done a great deal of work to replicate the results of the early researchers (Landauer et al., 1998, Landauer and Dumais, 1997) with varying degrees of success (Nakov et al., 2003, Foltz et al., 1999). There are two factors that contribute to the fact that many researchers have not been able to match the results (Haley et al., 2005). One factor is that developers must make many choices that affect the results of their assessment systems. The second factor is that there is no standard way of reporting the choices made or the results of these choices (Haley et al., 2005). Thus, critical features that improve LSA-based marking systems remain unpublished and unknown to the research community. Adding to the problem, researchers have difficulty comparing various systems and modifications to the basic LSA algorithm (Wiemer-Hastings, 2000, Kanejiya et al., 2003, Foltz et al., 1999, Lemaire and Dessus, 2001).

---

Copyright © 2007, Australian Computer Society, Inc. This paper appeared at the *Ninth Australasian Computing Education Conference (ACE2007)*, Ballarat, Victoria, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 66. Reproduction for academic, not-for profit purposes permitted provided this text is included.

One of the under-researched areas in LSA-based marking systems is the amount of training data needed for good results. Training data comprises both general and specific documents (Wiemer-Hastings et al., 1999). General training data are in the form of course textbooks; specific training data are human-marked answers. This paper addresses the question of the optimum amount of specific training data as well as a related question: How do you evaluate the results of using various amounts of training data? These questions are explored in the two major strands of the paper: a description of an experiment to ascertain the required amount of training data for an LSA-based automatic marking system and suitable metrics for judging the results. The experiment used EMMA (ExaM Marking Assistant), an automatic, Latent Semantic Analysis based system we are developing to mark short answers to exam questions in the domain of Computer Science.

Section 2 addresses the first strand by summarising how LSA works and explaining why it is important to know the optimum amount of training data. Section 3 deals with the second strand; it discusses existing ways to measure the success of LSA-based assessment systems and motivates the need for new metrics.

Section 4 discusses several standard statistical tests that could be used to measure the success of automatic assessment systems and argues that none of them is suitable for our purposes. Section 5 describes the experiment to determine an appropriate amount of training data.

Sub-section 6.1 provides one of the main contributions of the paper – a suggestion of two possible metrics for comparing automatic assessment systems. These metrics are the Manhattan distance (L1) and the Euclidean distance (L2). Sub-section 6.2 presents another contribution of the paper. It uses the two metrics to draw some conclusions about the amount of training data needed for an automatic marking system. Section 7 summarises the conclusions and lists further work.

## 2 Using EMMA to mark exams

This study used EMMA, our LSA-based automatic marking system, to determine the effectiveness of automatically assessing answers to questions about HTML and the optimum amount of training data. We have exactly 1,000 tutor-marked answers for two questions and wished to evaluate EMMA by comparing the tutor marks with marks assigned by our system.

A very brief summary of LSA follows. A more thorough explanation can be found in (Landauer et al., 1998). LSA transforms training data into a term-frequency matrix  $M$ , where  $m_{ij}$  is the weighted number of times term  $i$  appears in document  $j$ . It then decomposes  $M$  by Singular Value Decomposition into three matrices such that  $M = TSD$ . The  $M$  matrix can be very large. For instance, we are using over 45,000 documents which yield over 11,000 terms. LSA reduces  $S$  to  $k$  dimensions (where  $k$  is of the order of 300) resulting in the matrix  $S'$ . The matrices  $T$ ,

$S'$ , and  $D$  can be multiplied, resulting in  $M'$ , the least squares best fit of  $M$ , where  $M' = TS'D$  and  $M \approx M'$ .

To mark a student answer, EMMA chooses the five answers in the training data that are closest (using the cosine similarity measure) to the answer being marked. EMMA assigns the weighted average of these tutor-assigned marks to the answer being marked. Although our current implementation uses five answers, we plan to experiment with varying the number of answers in the future and also with using a threshold to limit the number of similar answers to those above a certain cosine value.

The amount and type of training data are two of many areas that affect the results of an LSA-based marking system (Haley et al., 2005). The training data for EMMA comprises both general and specific documents. The general documents are the course textbooks, which were divided into over 45,000 documents, each a paragraph long. The specific training documents came from the 1,000 tutor-marked answers. We split these tutor-marked answers into two sets, a set to be marked of 333 answers and a training set. To ensure good coverage for training purposes, we allocated the remaining 667 answers to be available for the training set. The experiment used varying numbers of the 667 training data answers to find an adequate amount.

The number of required tutor-marked answers is an important criterion for evaluating the feasibility of using an LSA-based marking system. One reason to use an automatic marker is to avoid the necessity of human marking. It would be hard to justify on the basis of cost savings the use of an automatic marker that requires more effort to create the training data than it would to mark the answers manually. (This analysis ignores the other benefits of automatic markers given in the Introduction.) It takes less human effort and is thus more practical if *few* answers are required for *good results*. The rest of this paper attempts to quantify the meanings of these two terms.

## 3 The inadequacy of existing success measures for LSA marking systems

Many choices need to be made when implementing an LSA-based marking system. The LSA literature leaves many parameters unspecified including number of dimensions in the reduced matrix, amount and type of training data, types of pre-processing, and weighting functions (Haley et al., 2005). The choice of these parameters is an intrinsic aspect of building an LSA marking system. Therefore, researchers need an adequate way to measure and compare the results of the various selections, as we shall now explore.

### 3.1 A simple metric

It is tempting to determine the percentage of marks where the tutor and Emma gave identical marks and use this number as the success measure. However, this simple metric gives an incomplete picture of the results. Consider the hypothetical case illustrated in Table 1. It is debatable which is the better marking system – A or B. Although Marking System A has a higher percentage of



identical answers than does Marking System B, Marking System B has 100% of its marks disagreeing with the human by at most one point while Marking System A has only 80% of its marks disagreeing by at most one point and 20% that differ by three or more points.

Marking System A		Marking System B	
% of Questions	Point Difference between Human and Computer Scores	% of Questions	Point Difference between Human and Computer Scores
80	0	75	0
0	1	25	1
0	2	0	2
5	3	0	3
15	4	0	4

Table 1. Hypothetical results for two marking systems that show that the simple metric of the percentage of identical scores for a four-point question hides important details

## 3.2 Widely used metrics

The literature offers two techniques to evaluate marking systems – precision and recall, and correlation.

### 3.2.1 Precision and recall

The first technique is the use of precision and recall; these measures are used widely in LSI and LSA research (Manning and Schütze, 1999, Dumais, 1991, Graesser et al., 2000, Nakov et al., 2003). Precision looks at how relevant the collection of retrieved documents is; it is the ratio of correctly retrieved, i.e. relevant, documents to all retrieved documents. Recall is a measurement of completeness. It is the ratio of correctly retrieved documents to all relevant documents i.e., those that were retrieved plus those that the retrieval system failed to retrieve (Foltz, 1990). As recall goes up, precision tends to go down; in the trivial case, a system achieves 100% recall if all the documents are retrieved, which would give the lowest precision. Information retrieval (IR) researchers plot values of precision for various levels of recall to provide a good picture of the effectiveness of their techniques (Dumais, 2003).

It is important to have a good metric to measure success when tuning a marking system. Dumais, in a widely cited study (Dumais, 1991), used precision and recall to justify the use of log-entropy weighting in the term-frequency matrix. The decision of a weighting function is a critical choice to be made by LSA researchers. Nakov et al. used precision and recall figures to argue that the choice of a weighting function is the most crucial of all tuning techniques (Nakov et al., 2003). Many researchers continue to justify the use of the log-entropy weighting factor (Foltz et al., 1998) by relying on the early work of Dumais (Dumais, 1991). Although log-entropy may well be the best weighting function, it should be justified for LSA-based assessment systems on research done *with* LSA-based assessment systems instead of IR systems. Researchers need to remember that Dumais is primarily interested in information retrieval rather than essay assessment. Although precision and recall are useful for evaluating IR techniques, we believe that using them to measure automatic marking systems is irrelevant. Recall

is not important – it makes no difference how many documents are returned because the marking system looks at only a pre-determined number that are the closest matches to the document being marked. Precision, on the other hand, is very important – the documents judged by the marking system to be relevant must actually *be* relevant. Precision, however, is a binary measure. It assumes that the documents are relevant or not. EMMA uses the cosine similarity measure to rank the documents in terms of their similarity to the answer to be marked. It then awards a mark by calculating the weighted average (using the cosine measure) of the five most similar answers. This feature of LSA provides a finer-grained measure than precision and recall, which are better suited to information retrieval.

### 3.2.2 Correlation

The second technique to evaluate marking systems is statistical correlation, which is used by many researchers (Foltz et al., 2000, Wiemer-Hastings et al., 1999). The most widely known correlation measures are Pearson's  $r$ , Spearman's  $\rho$ , and Kendall's  $\tau_b$  (Dancey and Reidy, 2002). Correlation statistics indicate how well one variable can be used to predict another variable. If human-assigned marks and computer-assigned marks agree, the correlation would be perfect. Even if the human marks were always twice the computer marks, the correlation would once again be perfect. In this case, a good correlation would not mean a good marking system. Another problem with the correlation statistic is that it would be low in the case where computer marks are off by plus-or-minus one point. In this situation, the computer mark could not be used to predict the human mark even though we argue that the overall results of the marking system could be very good if all the contradictory marks differ by only one point in either direction. For these reasons, the standard correlation statistics may not be useful for evaluating automatic marking systems.

Section 4 looks at a traditional statistical test and explains why it fails to help us evaluate our automatic marking system.

## 4 Problems with the traditional t-test

Having considered and rejected the metrics described in Section 3, we turn to the field of statistics for possible success metrics.

The traditional t-test for comparing two groups is a candidate for evaluating automatic marking systems. It comes in parametric and non-parametric versions. The parametric tests are more powerful than the non-parametric versions but the data must meet three assumptions to use them: normally distributed populations, approximately equal variations of the populations, and no extreme scores.

The t-test compares the means of two groups. For marking systems, one group is the human-assigned scores and the other group is the computer-assigned scores. When all participants take place in both conditions (short answer marked by tutor and short answer marked by

EMMA), the study design is known as within-participants (also called repeated measures or related design) and the appropriate parametric statistical test for comparing the groups is the t-test (Dancey and Reidy, 2002). The output of the t-test includes the mean scores for each group, the difference between them, and the standard deviations. With these values, one can compute the effect size, which is the difference of the means divided by the mean of the standard deviations. Confidence intervals around the effect sizes are an additional tool for evaluating results (Aberson, 2002). Therefore, if the data meet the three assumptions for using parametric tests, employing effect sizes with confidence intervals is a good way to evaluate automatic marking systems.

It is highly unfortunate for us that we cannot use the t-test and effect sizes with confidence intervals because our data are not normally distributed. The marks given by tutors are highly negatively skewed because the marks tend to cluster towards the high end of the marking scale. If the marks are not normally distributed, the effect sizes and confidence intervals will be incorrect (Thompson, 2002) and the t-test is not applicable. We can, however, use the Wilcoxon signed ranks test, which is the non-parametric version of the t-test. This test statistic is calculated by ranking the differences between the two scores. But the scores with zero difference are ignored because “they do not give us any information” (Dancey and Reidy, 2002).

There are two problems with the Wilcoxon test. The first problem is the elimination of those cases where the difference is zero. Dancey and Reidy (2002) claim that these cases do not give us any information, which may be true when trying to establish that there *is* a difference between two groups. However, when evaluating marking systems, we want to establish that there is *no* difference between two groups or that the difference is very small. If, for example, a marking system produces marks that agree with the human 95% of the time, that figure *is* informative, contradicting one of the assumptions of the Wilcoxon test. We need a test statistic that takes into account the number of cases where the difference between two marks is zero.

The second problem with the Wilcoxon test statistic is that it shows *whether* two groups are different but not by how much. To solve that problem, we can look at the mean difference given by the descriptive statistics – no difference or very small differences would allow us to conclude that there is no significant difference between two groups. However, as mentioned in Section 2, tuning an LSA-based marking system is critical. How should we compare the results of tuning the system? We cannot use mean differences by themselves; we must consider the standard deviations. This requirement leads us back to effect sizes, but as mentioned in Section 3, our results will be invalid because our data are not normally distributed.

For the reasons given above, we cannot use correlation statistics, t-tests, or effect sizes with confidence intervals. Section 6 presents two possible alternative metrics and provides an example of using them to evaluate test results. Before presenting the alternatives, however, we

digress to describe the experiments carried out to tune EMMA for HTML questions.

## 5 Evaluating EMMA for assessing questions on HTML

As discussed in Section 2, we are developing EMMA to mark answers to exams given in an introductory computer science course. In this study, we investigated how much training data are needed for good results marking questions on the use of HTML. (Note that we are treating the HTML as text. We are not running the HTML through a browser to evaluate the results.) Table 2 gives the text of the two questions. Each question was worth a maximum of four points. The questions were preceded with the instructions: Correct the following fragments of HTML. For each case, write the correct HTML and write one or two sentences about the problem with the original HTML.

Question ID	HTML	The desired appearance
A	<I>It is <B>very</I> </B> important to read this text carefully.	It is <i>very</i> important to read this text carefully.
B	Things to do: Pack suitcase, </BR> Book taxi.	Things to do:  Pack suitcase, Book taxi.

Table 2. Exam questions on HTML

The training data for EMMA comprises the course textbook and tutor-marked questions. The purpose of this study was to determine the optimum number of marked answers needed for best results. We have 1,000 marked answers for each of the two questions shown in Table 2. We used EMMA to mark each of the questions in the following way. We submitted 333 of the previously marked answers to be marked by EMMA; we used none of these 333 answers for training data. We discarded 40 answers for Question A and 41 answers for Question B due to marker error. (The human marker erroneously gave an aggregate score for the entire exam rather than providing a separate mark for each question.) We used the remainder of the 1,000 marked answers (minus those with input errors) to train the system. We ran sixteen experiments (for each question) using different amounts of training data: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600 and 627 (for Question A) and 626 (for Question B) marked answers. Figures 1 and 2 show the results of marking each of the two questions.

The figures contain a lot of information, which makes them difficult to understand and interpret. How can one determine the best amount of training data by looking at these charts? (This difficulty in interpreting the data is the main motivation for finding an alternative success metric.) The y-axis shows the percentage of marks. The x-axis shows the amount of training data. The data points show the percentage of marks where the human and the computer agree, or differ by from zero to four points. The

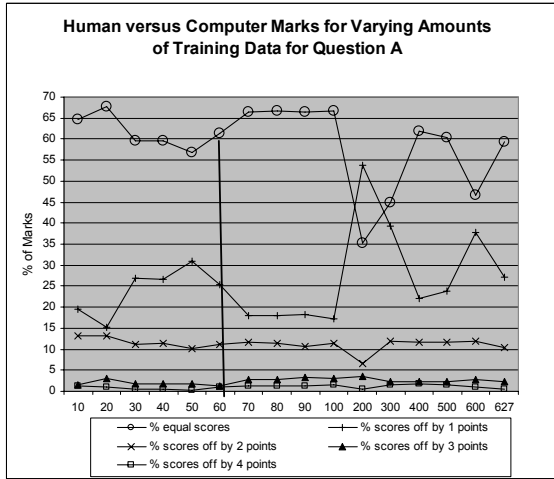


Figure 1. Comparison of tutor and computer marks for various amounts of training data for Question A

first set of data points (marked by an open  $\circ$ ) indicates the cases where there was zero difference between the tutor mark and the computer mark, i.e., they are identical. The second set of data points (marked by a  $+$ ) is where there was a difference of plus or minus one point. The fifth set (marked with an open square) indicates those questions with the worst results: either the tutor awarded four points and the computer awarded zero points, or the tutor awarded zero points and the computer awarded four points. The legend below the graph shows the correspondence between each set of data points and the amount by which the human and computer scores differ.

The viewer can see all of the results for a particular amount of training data by looking at a vertical slice of the graph. For example, the vertical line in the graph shows that when 60 training examples were used, EMMA matched the human about 61% of the time, differed by one point about 25% of the time, differed by two points about 11% of the time, differed by three points about 1% of the time, and differed by 4 points about 1% of the time.

Looking at a vertical slice of the graph shows the performance of EMMA for a particular amount of training data. Looking at a horizontal set of data points gives another point of view. The set shows how much the performance varies over different amounts of training data. For example, the set indicated with an  $x$  shows that

Manhattan distance measure (1-norm, or L1):

$$M(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^n |x_i - y_i|$$

Euclidean distance measure (2-norm, or L2):

$$M(\mathbf{X}, \mathbf{Y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Where  $\mathbf{X} = (x_1, x_2, \dots, x_n)$  and  $\mathbf{Y} = (y_1, y_2, \dots, y_n)$  are two  $n$ -dimensional vectors.

Table 3: Two metrics from vector arithmetic

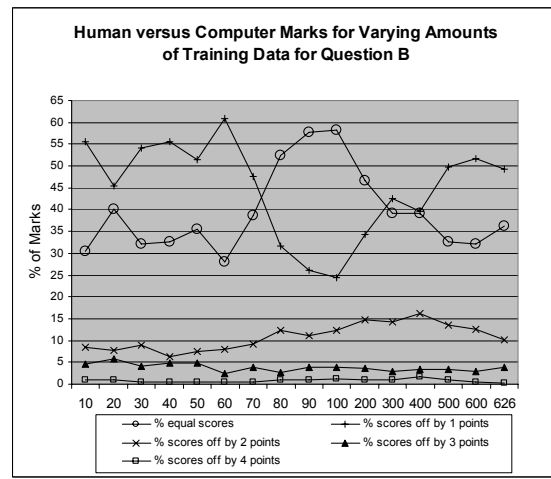


Figure 2. Comparison of tutor and computer marks for various amounts of training data for Question B

the marks that differed by plus or minus two points ranged from about 7% for 200 training data to about 13% for 10 and 20 training data.

We tried several different graphical ways to display the data – figures 1 and 2 show the clearest way we found. Even so, it is difficult to evaluate the overall effectiveness of varying the amount of training data by analysing these figures. The next section explains the metrics we used to answer the primary question of this paper: What is the appropriate amount of training data for computer assessment of short answers using an LSA marking system?

## 6 Two unconventional success metrics

We planned to evaluate the results of these 32 experiments by comparing the marks given by EMMA with the marks given by the tutor using a success metric from a standard statistical test. The inability to locate an appropriate metric, as described in Sections 3 and 4, combined with the difficulty in interpreting Figures 1 and 2 led us to look at unconventional metrics. We took two useful metrics from the field of vector space theory. Section 6 explains these metrics and shows how they can be used to evaluate the results of marking the two questions with varying amounts of training data.

### 6.1 Metrics from vector space theory

The Manhattan Distance measure (L1) and the Euclidean Distance measure (L2) are two metrics used to calculate the distance between two vectors (Gerald and Wheatley, 1970). Their application to marking exams is as follows. One vector is the list of scores given to a question by a human marker; the other vector is the list of scores assigned by EMMA. If the vectors are identical, the distance between the vectors is zero and the automatic marker would agree perfectly with the human.

The two measures are calculated using the well-known formulas shown in Table 3. These formulas compute the

Question A							
# of Marked Answers	% Equal Scores	Tutor and Computer differ by $\pm 1$	Tutor and Computer differ by $\pm 2$	Tutor and Computer differ by $\pm 3$	Tutor and Computer differ by $\pm 4$	Manhattan Distance L1	Euclidean Distance L2
10	64.7	19.5	13.2	1.5	1.2	104.8	10.2
20	67.7	15.3	13.2	3.0	0.9	109.3	10.5
30	59.6	26.9	11.1	1.8	0.6	97.0	9.8
40	59.6	26.6	11.4	1.8	0.6	97.9	9.9
50	56.9	30.8	10.2	1.8	0.3	92.5	9.6
60	61.4	25.4	11.1	1.2	0.9	94.9	9.7
70	66.5	18.0	11.7	2.7	1.2	108.1	10.4
80	66.8	18.0	11.4	2.7	1.2	106.9	10.3
90	66.5	18.3	10.8	3.3	1.2	110.2	10.5
100	66.8	17.4	11.4	3.0	1.5	113.8	10.7
200	35.3	53.9	6.6	3.6	0.6	122.2	11.1
300	44.9	39.2	12.0	2.4	1.5	132.6	11.5
400	62.0	22.2	11.7	2.4	1.8	119.2	10.9
500	60.5	24.0	11.7	2.4	1.5	116.2	10.8
600	46.7	37.7	12.0	2.7	0.9	124.3	11.1
627	59.3	27.2	10.5	2.4	0.6	100.3	10.0

Table 4. Number of answers used for training data over 16 Values ranked in order of amount of training data

Question B							
# of Marked Answers	% Equal Scores	Tutor and Computer differ by $\pm 1$	Tutor and Computer differ by $\pm 2$	Tutor and Computer differ by $\pm 3$	Tutor and Computer differ by $\pm 4$	Manhattan Distance L1	Euclidean Distance L2
10	30.5	55.7	8.4	4.5	0.9	144.0	12.0
20	40.1	45.5	7.8	5.7	0.9	142.2	11.9
30	32.0	54.2	9.0	4.2	0.6	137.4	11.7
40	32.6	55.7	6.3	4.8	0.6	133.5	11.6
50	35.6	51.5	7.5	4.8	0.6	134.1	11.6
60	28.1	60.8	8.1	2.4	0.6	124.3	11.1
70	38.6	47.6	9.3	3.9	0.6	129.3	11.4
80	52.4	31.7	12.3	2.7	0.9	119.5	10.9
90	57.8	26.2	11.1	3.9	0.9	120.5	11.0
100	58.1	24.4	12.3	3.9	1.2	128.3	11.3
200	46.7	34.4	14.7	3.6	0.9	139.8	11.8
300	39.2	42.5	14.4	3.0	0.9	141.3	11.9
400	39.2	39.5	16.2	3.3	1.8	162.6	12.8
500	32.6	49.7	13.5	3.3	0.9	147.6	12.1
600	32.0	51.8	12.6	3.0	0.6	138.6	11.8
626	35.7	50.5	8.7	4.2	0.9	137.5	11.7

Table 5. Number of answers used for training data over 16 Values ranked in order of amount of training data

distance between the vectors in slightly different ways. The L1 computes the sum of the differences between each point in the two vectors; the L2 computes the square root of the sum of the squares of the differences. The next sub-section shows how each of these two metrics evaluates the 32 experiments discussed in the previous section: the effects of 16 different amounts of training data for two different questions.

## 6.2 Example of L1 and L2 using questions on HTML

Tables 4 and 5 show the results of the experiments described in Section 5. The final two columns show the values for L1 and L2, as calculated by the formulas in Table 3. The results are sorted in increasing order of

amount of training data. By careful inspection, one can see that the smallest L2 distance corresponds to the best result. A perfect automatic marking system, i.e., one that agrees with the tutor 100% of the time would have a Euclidean distance of zero.

Figure 3 displays a graphical representation of the Euclidean distance (L2) measures from Tables 4 and 5. Recall that a lower distance is a better result than a higher distance. The graph shows that for Question A, the L2 metric ranges from about 9.5 to 11.5. For Question B, L2 ranges from about 11 to 12.8.

A few conclusions can be drawn from studying Figure 3. The most obvious, given that lower L2 values indicate a better result, is that EMMA works better for Question A

than it does for Question B for all amounts of training data tested. Another result is that the graphs look slightly different for each question. Question A shows that the

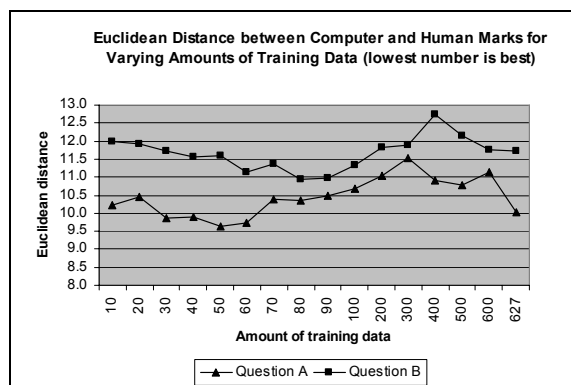


Figure 3. Euclidean distance measure for Question A and Question B

marking system works best at around 50 and gets progressively worse until about 300 where it then gets better but not as good as for 50. Question B shows a smoother curve from 10 to 80 where the results get better before getting worse from 80 to 400, and then like Question A, get better again but not as well as for 80.

A surprising result was that the largest amount of training data did not produce the best results for either of the questions.

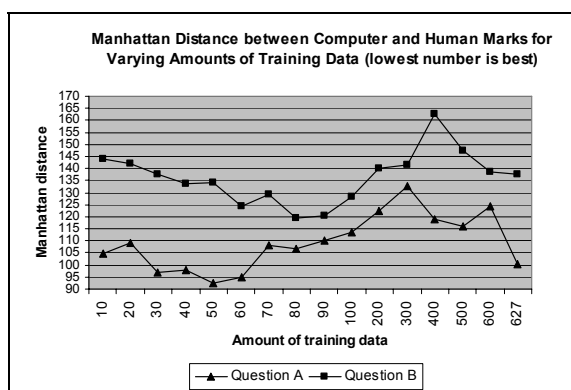


Figure 4. Manhattan Distance Measure for Question A and Question B

Figure 4 is similar to Figure 3. It displays a graphical representation of the Manhattan distance (L1) measures from Tables 4 and 5. As for L2, a smaller distance for L1 is a better result than a larger distance. The graph shows that for Question A, the L1 metric ranges from about 92 to 133. For Question B, L1 ranges from about 120 to 163. The shape of these two graphs is quite similar to the shapes in Figure 3. Thus, the L1 metric provides similar insights to the L2 metric.

The L1 and L2 metrics provided a figure that could be used to assess the results of the different experiments quickly. This evaluation agreed with a careful hand analysis of all of the figures given in Tables 4 and 5.

## 7 Conclusions and further work

### 7.1 Conclusions

This study resulted in three significant insights to the field of automatic assessment of short answers.

Contrary to expectations, more training data are not better for these questions. In fact, the better results were from around 50 to 90 for both of the questions. This is good news for those wishing to use an LSA-based marking system because it is easier, quicker, and cheaper to supply fewer human marked answers.

We had different results for the two questions. EMMA worked best for Question A at 50 marked answers used for training data; for Question B, the most effective amount of training data was 80. Therefore, fine-tuning an automatic assessment system requires analysis to be done for each question separately.

The Manhattan and Euclidean distance measures seem to be promising tools for automatic marking researchers to evaluate their systems. These two metrics take into consideration the values of agreement between human and computer over the whole range of possibilities. That is, they evaluate the results where human and computer marks are identical, where they are off by plus or minus one point, plus or minus two points, and so on until the worst result which is where the human and computer differ by the maximum point value of the question. The simple metric that uses the value where the human and computer marks are identical can lead to ambiguity, as demonstrated in section 3.1. L1 and L2 give a richer picture of the effectiveness of an automatic marking system than the simple metric and are no more difficult to analyse than the simple metric.

### 7.2 Further work

We plan further work to, on the one hand, better understand our results and on the other hand better understand how to tune our LSA-based automatic marking system:

- We plan to meticulously examine training data for these questions to determine why the results were variable.
- We want to experiment with a new hypothesis based on the results reported in this study: the quality of the training data is more important than the quantity.
- We will use L1 and L2 to compare results of varying other LSA parameters such as the number of dimensions, weighting method, stop words, stemming, and spell checking.

## 8 Acknowledgement

The work reported in this study was partially supported by the European Community under the Innovation Society Technologies (IST) programme of the 6th Framework Programme for RTD - project ELeGI, contract IST-002205. This document does not represent the opinion of the European Community, and the

European Community is not responsible for any use that might be made of data appearing therein.

## 9 References

- Aberson, C., (2002). Interpreting Null Results: Improving Presentation and Conclusions with Confidence Intervals. *Journal of Articles in Support of the Null Hypothesis* **1**(3):36-42.
- Dancey, C. P. and Reidy, J., (2002). *Statistics Without Maths for Psychology: Using SPSS for Windows*, Essex, England, Prentice Hall.
- Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. and Harshman, R., (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science* **41**(6):391-407.
- Dumais, S. T., (1991). Improving the retrieval of information from external sources. *Behavioral Research Methods, Instruments & Computers* **23**(2):229-236.
- Dumais, S. T., (2003). Data-driven approaches to information access. *Cognitive Science* **27**:491-524.
- Foltz, P. W., (1990). Using latent semantic indexing for information filtering. *Proc. Conference on Office Information Systems*. (Ed, Allen, R. B.), Cambridge, MA. 40-47.
- Foltz, P. W., Gilliam, S. and Kendall, S. A., (2000). Supporting content-based feedback in online writing evaluation with LSA. *Interactive Learning Environments* **8**(2):111-129.
- Foltz, P. W., Kintsch, W. and Landauer, T. K., (1998). The Measurement of Textual Coherence with Latent Semantic Analysis. *Discourse Process* **25**(2&3):285-307.
- Foltz, P. W., Laham, D. and Landauer, T. K., (1999). Automated Essay Scoring: Applications to Educational Technology. *Proc. ED-MEDIA '99*. Seattle.
- Furnas, G. W., Deerwester, S., Dumais, S. T., Landauer, T. K., Harshman, R. A., Streeter, L. A. and Lochbaum, K. E., (1988). Information retrieval using a singular value decomposition model of latent semantic structure. *Proc. of 11th annual int'l ACM SIGIR conference on Research and development in information retrieval*. ACM. 465-480.
- Gerald and Wheatley, (1970). *Applied Numerical Analysis*, Addison-Wesley.
- Graesser, A. C., Wiemer-Hastings, P., Wiemer-Hastings, K., Harter, D. and The Tutoring Research Group, (2000). Using latent semantic analysis to evaluate the contributions of students in AutoTutor. *Interactive Learning Environments*. [Special Issue, J. Psotka, guest editor] **8**(2):129-147.
- Haley, D. T., Thomas, P., De Roeck, A. and Petre, M., (2005). A Research Taxonomy for Latent Semantic Analysis-Based Educational Applications. *Proc. International Conference on Recent Advances in Natural Language Processing'05*. (Eds, Angelova, G., Bontcheva, K., Mitkov, R., Nicolov, N. and Nikolov, N.), Borovets, Bulgaria. 575-579.
- Kanejiya, D., Kumar, A. and Prasad, S., (2003). Automatic Evaluation of Students' Answers using Syntactically Enhanced LSA. *Proc. HLT-NAACL 2003 Workshop: Building Educational Applications Using Natural Language Processing*, 53-60.
- Landauer, T. K. and Dumais, S. T., (1997). A solution to Plato's problem: the Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review* **104**(2):211-240.
- Landauer, T. K., Foltz, P. W. and Laham, D., (1998). An introduction to Latent Semantic Analysis. *Discourse Processes* **25**:259-284.
- Lemaire, B. and Dessus, P., (2001). A system to assess the semantic content of student essays. *Journal of Educational Computing Research* **24**(3):305-320.
- Manning, C. D. and Schütze, H., (1999). *Foundations of Statistical Natural Language Processing*, Cambridge, Massachusetts, MIT Press.
- Miller, T., (2003). Essay assessment with Latent Semantic Analysis. *Journal of Educational Computing Research* **28**.
- Nakov, P., Valchanova, E. and Angelova, G., (2003). Towards Deeper Understanding of the LSA Performance. *Proc. of Recent Advances in Natural Language Processing*. Borovetz, Bulgaria. 311-318.
- Page, E., (1967). Statistical and linguistic strategies in the computer grading of essays. *Proc. 1967 International Conference On Computational Linguistics*. 1-13.
- Thompson, B., (2002). What Future Quantitative Social Science Research Could Look Like: Confidence Intervals for Effect Sizes. *Educational Researcher* **31**(3):25-32.
- Wiemer-Hastings, P., (2000). Adding syntactic information to LSA. *Proc. 22nd Annual Conference of the Cognitive Science Society*. 989-993.
- Wiemer-Hastings, P., Wiemer-Hastings, K. and Graesser, A. C., (1999). Improving an intelligent tutor's comprehension of students with Latent Semantic Analysis. *Artificial Intelligence in Education*. (Eds, Lajoie, S. P. and Vivet, M.) IOS Press. Amsterdam.

# Peer Assessment Using Aropä

John Hamer

Catherine Kell

Fiona Spence

Department of Computer Science

Centre for Flexible and Distance Learning

The University of Auckland

J.Hamer@cs.auckland.ac.nz

c.kell@auckland.ac.nz

f.spence@auckland.ac.nz

## Abstract

Aropä is a web-based peer assessment support tool that has been used extensively in a wide variety of settings over the past three years. We describe the design of Aropä and how it can be configured, and present some results from a research study into the use of peer assessment in large undergraduate courses. There is evidence to show that while students find peer assessment challenging, it can be an effective aid to learning. The study also reveals marked differences in attitude toward peer assessment between different student bodies.

## 1 Introduction

Over the past three years we have introduced peer assessment into a variety of undergraduate courses in Computer Science, Software Engineering, Pharmacology, English and Photography. The courses range in size from forty to four hundred, and in level from introductory to fourth year. The material assessed includes computer programs, technical reports, software designs, academic and mixed-mode essays, photographs, posters, team member performances, and presentations. Most material and most reviews were prepared individually, but we also have instances of team assignments and team reviewing. The assessment exercises were primarily formative, with most including a token mark for participation.

It was only possible to conduct peer assessment exercises on this scale with the help of support software, in this case a locally-developed web-based tool called Aropä (*Aropä* means “peer review” in Maōri). A detailed description of Aropä has not previously appeared in the literature, so we take the opportunity in section 3 to present an overview of the tool and its appearance from a student perspective. We then present observations on three of the courses that have used the tool. The observations were taken from anonymous surveys and interviews with both teaching staff and students. The results show that peer assessment can aid learning in a variety of ways. We also observe that marked differences are evident between different student groups.

## 2 Why peer assess?

Peer assessment is attracting increasing attention from educators looking for new ways of improving

learning outcomes in undergraduate courses. Many of the tasks associated with peer assessment are associated with Bloom’s 1956 “higher” learning outcomes of *analysis* and *evaluation*. More specifically, literature surveys by Ballantyne et al. (2002) and Topping (1998) suggests that peer assessment can:

- help to consolidate, reinforce and deepen understanding, by engaging students in cognitively demanding tasks: reviewing, summarising, clarifying, giving feedback, diagnosing misconceptions, identifying missing knowledge, and considering deviations from the ideal;
- highlight the importance of presenting work in a clear and logical fashion;
- expose students to a variety of styles, techniques, ideas and abilities, in a spectrum of quality from mistakes to exemplars;
- provide feedback swiftly and in quantity. Feedback is associated with more effective learning in a variety of settings. Even if the quality of feedback is lower than from professional staff, its immediacy, frequency and volume may compensate;
- promote social and professional skills;
- improve understanding and self-confidence; and
- encourage reflection on course objectives and the purpose of the assessment task.

Historically, peer assessment has been largely confined to small graduate courses or in tutoring contexts. However, the potential benefits for large undergraduate classes are considerable. In addition to the suggested learning benefits, time saving is also often given as a pragmatic reason in favour of peer assessment (Ballantyne et al. 2002).

## 3 Aropä

The traditional form of course assignment work involves students working independently to prepare a submission, which is then marked by a course marker who produces a grade and (perhaps) some feedback for the student to reflect on (figure 1).

Peer assessment modifies this cycle to involve the student in reviewing and possibly rating the feedback (figure 2). New “dispute” and “rating” loops are introduced by peer assessment, in acknowledgement that the reviewing will sometimes be flawed. While reviewing errors can and do arise in both kinds of activity, peer assessment involves a change in power relations between author and reviewer that legitimises questioning a review.

Copyright ©2007, Australian Computer Society, Inc. This paper appeared at Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, Australia, February 2007. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 66. Samuel Mann and Simon, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

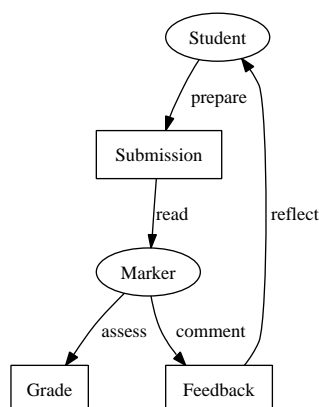


Figure 1: The traditional course assignment cycle

The main functions of Aropä are to manage: allocation of submissions to reviewers; web-entry of reviews; access to feedback, disputes and ratings; and weighted-average grade calculation.

### 3.1 The student interface

A student assumes multiple roles during peer assessment, first as a producer of material to be assessed, then as a reviewer, and finally as a receiver of feedback. Aropä presents all these roles to the student simultaneously, arranged in sections that (in a Javascript-enabled browser) can be shown or hidden by clicking on the section title.

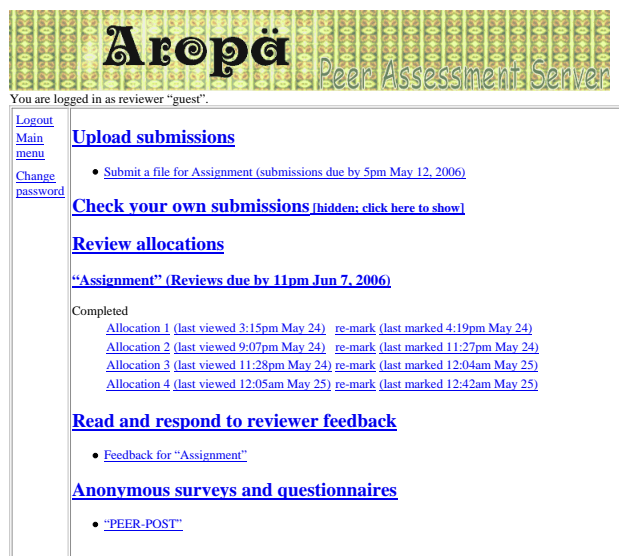


Figure 3: The student interface to Aropä

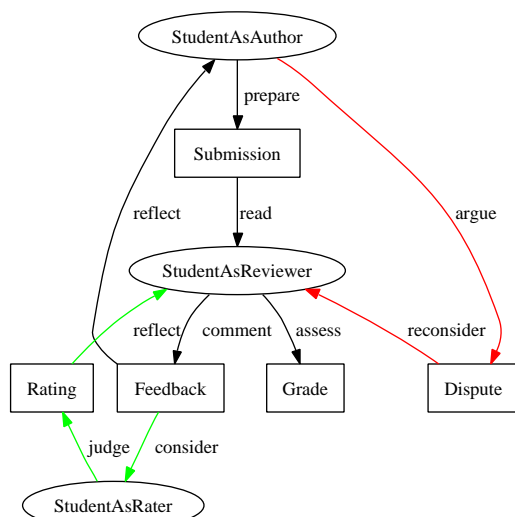


Figure 2: The peer-assessment cycle

Figure 3 shows a typical main screen after log in. The left of the screen displays links to logout, change password, and (redundantly) return to the main menu. On the right are sections to upload files for any current assignments, check that files have been uploaded correctly, review allocated assignments, read feedback, and participate in any class surveys. It is possible for a student to have more than one peer assessed assignment at a given time, in which case each assignment appears in a subsection, and an additional section is displayed at the top of the screen to allow a subset of the current assignments to be displayed.

Uploading is a straightforward operation. The student is prompted for the name of one or more files, which are sent to the web server when a “save” link is invoked. Currently no constraints are imposed on the names, types, size or number of files. Some courses (particularly in Computer Science) have their own submission system, in which case Aropä can be instructed to look for submitted files in a directory mounted on the server.

The “Check your own submissions” section is provided mainly to reassure students that their files have been received by Aropä. One of the biggest technical problems we have faced is when students submit files (from home) in a non-standard document format (typically old versions of MacWrite or Lotus Notes, etc.). This facility allows students to confirm their submissions are readable in a “standard” environment, such as a University computer laboratory.

Students spend the largest portion of their time in the “Review allocations” section. This section contains links to view and mark each of the allocations they have been assigned to review. Usually reviewing



is double-blind, in which case Aropä numbers the allocations sequentially. If half-blind reviews are used, the name of the author is displayed instead. The “view” link displays a list of all the files submitted by the author. The files can be downloaded individually, or together as a compressed “zip” archive.

Marking is done using a “grading rubric” prepared by the lecturer. The rubric can contain HTML elements (headings, paragraphs, lists, tables, etc.) together with checkboxes, groups of radio buttons, and text areas. Figure 4 shows part of a rubric used for grading a report on a pharmaceutical drug. Three groups of radio buttons are shown. This rubric included eleven such groups in total, followed by comment areas for “what did you like best about the assignment,” “how could it be improved,” and “any general comments.”

You are logged in as reviewer "guest".

[Logout](#)  
[Main menu](#)  
[Change Reviewer](#)  
[Password](#)

## Marking for allocation #1 for assignment "Assignment"

**ABSTRACT (25 Marks)**

**WORD COUNT (5)**

- ☐ The abstract is over 250 words
- ☒ The abstract is less than or equal to 250 words

**STRUCTURE (7)**

- ☐ No structure present
- ☐ Some structure apparent but largely jumbled and/or illogical
- ☐ Structure apparent but may not be entirely appropriate
- ☒ Clear, logical structure

**CONTENT (13)**

- ☐ Large amounts of irrelevant information. Little or no mention of key points.
- ☐ Important aspects ignored or receive minimal attention. Some irrelevant information.
- ☐ Appropriate weighting provided for most aspects of the article. Purpose and outcomes of study explained.
- ☒ Appropriate weighting provided for all aspects of the article. Purpose and outcomes of study clearly explained.

Figure 4: A formative grading rubric used in Pharmacology

A contrasting rubric used in an English class is (partly) shown in figure 5. This rubric is entirely formative, consisting of a series of text boxes but no “grading” elements.

In general, rubrics fall between these two extremes, with grading elements intermingled with free-form comment areas.

Facilities are provided for formatting comments. This can be done with a simple “wiki markup” or (on a modern browser) using a Javascript HTML editor<sup>1</sup>. The Javascript editor was added in the first 2006 semester, and appears to have resulted in a significant increase in the amount of written feedback. We return to this issue in section 4.2.

Feedback is provided to the student through a modified version of the grading rubric. The radio and check buttons are replaced by the number of reviewers awarding each mark. The screen shot in figure 6 shows that all three reviewers agreed that the notes were comprehensive, and two out of three approved of the organisation and conventions. Comments are shown next to a reviewer identifier (see figure 7, which lets the student reconcile comments made in different comment boxes).

The final facility provided by Aropä is an on-line survey. Teaching staff can set up one or more surveys soliciting anonymous feedback from students. The surveys use the same elements as rubrics. Surveys

You are logged in as reviewer "guest".

[Logout](#)  
[Main menu](#)  
[Change Reviewer](#)  
[Password](#)

## Marking for allocation #1 for assignment "Draft assessment exercise"

Write at least one sentence in response to each of the five questions below (making 300 words altogether) with regard to the draft essay.

1. What is the issue that the draft is addressing. Is it interesting, or do you care?

[Write your response to the issue in the text box below](#)

(you) Oct 7, 2005 the issue is will post feminism atchive what feminism in the 1970s did not. Yes it is interesting, beacuse i am a woman, and the essay brings up intesting points showing wear feminism may be leading.

The issue is will post feminism atchive what feminism in the 1970s did not. Yes it is interesting, beacuse i am a woman, and the essay brings up intesting points showing wear feminism may be leading.

2. Say what you think is the argument of the draft. If the argument is not clear, suggest what a possible argument might be.

[State the argument in the text box below](#)

(you) Oct 7, 2005 the argument is that post femisim has a better base or idea of women's rights than the 70's and that is women's choice as an individual.

The argument is that post femisim has a better base or idea of women's rights than the 70's and that is women's choice as an individual.

3. What reasons does the writer offer to support the argument? (you may like to break down the argument into quasi-syllogistic premises or to identify a Toulmin-style warrant for the argument).

[Give the reasons in the text box below](#)

(you) Oct 7, 2005 the issue of feminism is still relvant and effects all of society and has been an issue for over 100 years.

Figure 5: A formative grading rubric used in English

You are logged in as reviewer "guest".

[Logout](#)  
[Main menu](#)  
[Change Reviewer](#)  
[Password](#)

## Feedback for "Final Resources"

Based on 3 reviews.

**Notes**

Only complete this section if you are reviewing a set of study notes.

Consider the information presented in the notes.

- ☒ [3] The notes provide a comprehensive study guide to the topic. All examinable topics are covered in sufficient detail that this resource will satisfy most revision needs. I have identified aspects of the notes that I especially liked in the comments section.
- ☐ [0] Most of the information was accurate, but either some errors were made or some important details were not mentioned (or perhaps some unnecessary material was included). I have elaborated in the comments section.
- ☐ [0] The notes contain some serious errors, miss key detail, or devote excessive space to peripheral topics. I have elaborated in the comments section.

Now consider the organization of the notes.

- ☒ [2] Sections are used effectively to organise the material in a logical fashion. It is always clear what topic is being discussed, and how it relates to other sections. Each section anticipates the next, and transitions (at the end and start of each section) enhance understanding.
- ☒ [1] Sections are provided that enable the reader to follow the notes easily, but they are not polished. I have made identified specific areas that could be improved in the comment section.
- ☐ [0] Organization is lacking, and I often felt confused. I have made some suggestions for improvement in the comment section.

"Mechanics" refers to conventions for spelling, grammar, punctuation, capitalization, paragraphing, and formatting. Select the statement that most closely summarises the mechanics of the notes.

- ☒ [2] Correct conventions facilitate the reading of the notes. Conventions, used strategically, add to impact of the notes.
- ☐ [1] Correct conventions facilitate the reading of the notes.
- ☐ [0] Errors are minor but affect the reading of the notes.
- ☐ [0] Poor mechanics impede the reading of the notes.

(Parts of this rubric were adapted from <http://home.mindspring.com/~lclifton1/id6.html>)

Figure 6: Reviewer grade feedback

<sup>1</sup>We use the tinymce editor (Moxiecode Systems AB 2006)

Mandatory comments section	
Regardless of which section you completed above, please provide detailed comments to the author(s) on the following.	
<a href="#">Comment on what you liked most about the resource</a>	
R-1 Jun 4, 2006	I love the applet you recommended. Just spent 40 minutes messing around on it.. lol, the trees are pretty cool, I keep trying to create some complicated scenarios for the different trees, but they pretty much balanced everything in like 2 rotations! lol, but yes, very nice, puts everything said on the notes in perspective.
R-2 Jun 6, 2006	A lot of information of each type of tree. Unless john ask about some other type of tree in the exam, everythign should be fine.
R-3 Jun 3, 2006	You have thoroughly covered all the main topics. Clearly presented diagrams illustrate concepts well.
R-1 Jun 4, 2006	I like the structure of the notes and the images provided
<a href="#">Comment on how you think the resource could be improved</a>	
R-1 Jun 4, 2006	I feel the formatting and layout of the notes could be improved. You have some sub sub headings that are underlined.. eek.. and maybe use block centring. Need more white spaces between sections and paragraphs, kinda scary to lines and lines and lines of text, needs more seperation.
R-2 Jun 6, 2006	It all looks fine :)
<a href="#">Any additional comments you would like to make</a>	
R-1 Jun 4, 2006	Good overall, a lot of information provided here, with pictures. Just work on structure/layout a bit. Did I mention that applet is great?! Wraps everything together nicely.
R-2 Jun 6, 2006	Well done, overall a very useful set of notes :)
R-3 Jun 3, 2006	It was really good, well done.

Figure 7: Review comment feedback

can be open to all students, or limited to participants from one or more assignments or any ad-hoc group.

### 3.2 Configuration

A typology of peer assessment from (Topping 1998) is reproduced in table 1. Aropä has been designed to support all of the variations in this typology, and we have experience in applying the system to many different configurations.

We have already made mention of the range of SUBJECT AREAS that have used Aropä, and will give examples of different staff and student OBJECTIVES, STAFF ASSESSMENT, OFFICIAL WEIGHT PLACE, TIME, REQUIREMENT and REWARD in the section 4.

The rubric format described in section 3.1 supports a mixture of both quantitative and qualitative FOCUS.

The PRODUCT being assessed is most often written work of some form or other, but this is not a restriction. Aropä has also been used to assess oral presentations and for team member performance reviews. In such cases, photographs of the assessee are uploaded, to remind the reviewer of which presentation or team member they are reviewing for a given allocation.

The variables DIRECTIONALITY, YEAR and ABILITY all concern the allocation of reviews to reviewers. Aropä allocations can be manual or automatic (i.e., randomly generated). Automatic allocations can be made in one or more *streams*, where each stream involves a subset of either the reviewers or assessees. For example, a class can be grouped into top, middle and poorly performing students. Each submission can then be assigned to one or more reviewers from each group. Or, conversely, each reviewer can be allocated one or more submissions from each group. Streaming has been used in this way in some Computer Science courses.

Reviewers and assessees are independent sets, so there is no difficulty in allocating students from different courses or years. To date, this has been done with staff markers using the system to mark student

work, and with a final-year Software Engineering class reviewing work from a third-year design course.

Self-review is supported as a per-activity option. It has been occasionally used in conjunction with peer review; we have no experience with using self-review in isolation. However, positive findings on self-review have been reported in Dochy et al. (1999), and we plan to investigate this further in the future.

Submissions can also be “seeded” with pre-prepared solutions. These may be model solutions, or solutions with particular flaws. Seeded submissions are generally not identified to the reviewers. While it would be possible to use seeded submissions as a quality check, this has not been done in any of the courses using Aropä. Rather, the motivation has been to ensure each student is exposed to at least one high quality solution.

Aropä supports individual or group submissions and reviews, in any combination. Groups used for both submission and review are not required have the same membership, although this is normally the case.

### 3.3 Other features

**Review of reviews** Aropä regards each peer assessment activity as an assignment in itself, one in which the reviewers are authors of their reviews. Reviews can thus be reviewed using all the facilities available to normal assignments<sup>2</sup>. Typically, “review reviews” are carried out by a lecturer or course tutor. The reviewer “quality” grades from a review-of-the-reviews are used to automatically weight the grades assigned by the reviewer, so the marks from “good” reviewers can be made to count for more.

**Late submissions** Reviewer allocations are most often made after all submissions have been received, so that only students who submit material for the activity are included as reviewers. Aropä keeps track of the time each allocation is viewed, and can switch allocations that have not been seen to make room for occasional late submissions.

**Dialogue** Double-blind dialogue between author and reviewer is supported. Author can respond to reviewer’s comments during a “feedback” period. The administrator can decide whether this period overlaps with the review period (in which case a reviewer has the opportunity to change her comments and/or grades) or to keep the periods distinct.

**No class lists** Aropä does not connect to an enrolment database, and has no notion of a class list. For each activity, the participating students are determined either from the names on the submitted files or from a list prepared by the lecturer. This design makes it possible to include students who wish to participate in the activity, but are for some reason not enrolled, or whom the lecturer wishes to hide (such as the author of a seeded solution).

**Grade calculation** The grade for a submission is calculated using a weighted average of all the reviews. The reviewer weights can be assigned by the “review review” or using an automatic calibration algorithm (Hamer et al. 2005). A grade variance report identifies areas of significant disagreement between reviewers. The administrator can investigate, and exclude particular reviews or reviewers, or even rubric questions.

<sup>2</sup>The process can continue indefinitely. The grades for the final unreviewed reviews are computed using the algorithms described in Hamer et al. (2005)

VARIABLE	Range of variation
CURRICULUM AREA/SUBJECT	All
OBJECTIVES	Of staff and/or students? Time saving or cognitive/affective gains?
FOCUS	Quantitative/summative or qualitative/formative or both?
PRODUCT/OUTPUT	Tests/marks/grades or writing or oral presentations or other skilled behaviours?
STAFF ASSESSMENT	Substitutional or supplementary?
OFFICIAL WEIGHT	Contributing to assessee final official grade or not?
DIRECTIONALITY	One-way, reciprocal, mutual?
PRIVACY	Anonymous/confidential/public?
CONTACT	Distance or face-to-face?
YEAR	Same or cross year of study?
ABILITY	Same or cross ability?
CONSTELLATION ASSESSORS	Individual or pairs or groups?
CONSTELLATION ASSESSED	Individual or pairs or groups?
PLACE	In/out of class?
TIME	Class time/free time/informally?
REQUIREMENT	Compulsory or voluntary for assessors/ees?
REWARD	Course credit or other incentives or reinforcement for participation?

Table 1: A typology of peer assessment, from (Topping 1998)

**Persona** An course administrator can temporarily assume the persona of a student at any time, without requiring to know their password. This feature has proved suprisingly useful for investigating reported problems.

**Time-on-task** A report calculates the approximate times when a student is using Aropä and for how long. The data is approximate because it cannot measure time spent away from the computer, but it does give an indication of the workload imposed by the peer assessment activity.

## 4 Courses using Aropä

### 4.1 Pharmacology

A large year-2 class in Pharmacology used Aropä in the second semester of 2005 and again (as a year-3 class) in the first semester of 2006. In 2005, 335 student participated, and in 2006 there were 180 essays submitted. In both instances the students worked individually writing an essay on a pharmaceutical drug, chosen from a list of six. The essays were around 4–8 pages, and followed a strict academic format (abstract, references, etc.). We report here on the 2005 exercise.

Students were instructed to hand in their assignments electronically to the Aropä website, and also to submit a second version (without reference list) to Turnitin (Barrie 2006), a plagiarism detection site. Assignments were then allocated randomly for students to mark, 4 per student, over a three day period. A detailed marking rubric was provided on the website (this was available for a week prior to the deadline), and marks were submitted via this rubric. Students were awarded 4% of their grade for marking the four assignments. The assignment itself contributed 20% of their final grade.

Students were asked to provide feedback on their understanding and prior perceptions about peer assessment. Only seven students responded to these initial questions. They all had some idea of what peer assessment was, although none had previously participated in peer assessment. All of these students expressed a reluctance to participate in this peer assessment process.

#### 4.1.1 Positive feedback from the class representative and students

Positive feedback from the class representative<sup>3</sup> and the class indicated that the students had perceived and experienced two important pedagogical benefits of peer assessment.

Firstly, they had gained knowledge and understanding of the academic environment and processes. They felt they had gained a much better understanding of what is involved in marking generally and how to approach writing essays differently as a result of the experience: *“it was a good way to learn alternative ways of writing reports”*; *“I realised how poor some reports are”*; *“it was interesting to see how bad mine was compared to others.”*

Secondly, in relation to the actual content knowledge in the field of Pharmacology, students felt they had a deeper understanding of the drug topic, that it had reinforced learning and that they had been able to pick up on information about the drug which hadn’t been found while researching for their own essay. There was a suggestion that it would be useful to be able to view reports on other drugs (not to mark) to further their understanding in Pharmacology generally.

In commenting positively on the marking process two students suggested that it would be good to be provided with three model answers (excellent, average and poor) so that you could benchmark your marking. There was some sense of surprise: *“that reviewers commented on the good points and the bad,”* and *“I didn’t expect the marking to be done so fairly and consistently — students marked really well”*; *“I felt compelled to write comments...also to improve the students’ work in the future.”*

There was little feedback on the interface except for a response saying that *“comment box at the side was a really good way to learn as lecturers don’t usually have much time for comments.”*

<sup>3</sup>The University of Auckland operates a “class representative” system in which one or two students from each course volunteer to liaise with the department and raise any issues of concern.

#### 4.1.2 Problems and recommendations from the class representative and students

Some comments related to details about the interface and technical issues. One asked “*How did Aropa generate a number out of four reviewers? How does it deal with bias?*” Students reported feeling anxious and as “*needing to feel safe about loading their stuff.*” They did experience some glitches, for example, “*when an assignment was submitted but it wasn’t loaded*” and it would “*allay nerves if the system sent out a confirmation email after submission was made.*”

On pedagogical issues, the class representative’s and students’ feedback covered a number of areas, including:

- preparation
- the marking rubric
- ‘genre’ and language considerations
- the requirement for marking
- timing.

Each of these is detailed below.

The class representative and students reported that the exercise could have been improved with better preparation and more warning given to students to allay fears and anxieties. Students could have been better informed about the benefits of the approach (“*more theoretical justification of the use of peer assessment*”) and how it had been successfully used in other contexts. Better participation in reviewing could possibly have been achieved through “*more encouragement.*”

Secondly, there were a number of suggestions about the marking process and the marking rubric. Although these indicate a sense of frustration on the part of students with this particular exercise, they can be seen as positive in the sense that they engage with what peer assessment is about and make positive suggestions to improve it. The marking rubric was seen by a number of students as too limiting, and needing to be expanded to include more than three options. This seemed to be indicating that students felt that the rubric favoured short answers tailored to the rubric, rather than the answers of students who had spent more time on researching and checking information. The class representative reported that students had felt that: “*The marking rubric did not account for the effort put into understanding the drug and its mechanisms — the rubric encouraged fitting information to it.*” It was also seen as not entirely correlating to the questions given previously — it did not correspond well to what was asked for in the assignment. There was a request for “*exceptionally well done*” criteria or “*bonus marks for deep level of research,*” and frustration that it only “*allowed you to give OK or good results.*” It was suggested that the marking rubric should cover spelling and grammar, and that marks should be allocated to “*essay format and quality.*”

Thirdly, although the assignment required a strict essay format with an abstract and referencing, there were interesting responses around this. Two students indicated that the class mixed BSc students with BPharm students and that there were “*different approaches, writing styles and opinions as to which sections were more important and deserved more focus*”:

*an essay by a BSc student and marked by a BPharm student could have been perceived very differently. This could have contributed to inconsistencies.*

Responses	Time spent
(6)	2 hrs ( $\frac{1}{2}$ hr each)
(6)	6 hrs ( $1\frac{1}{2}$ hrs each)
(5)	4 hrs total
(2)	1 hr total
(2)	50 minutes each assignment
(1)	45 mins each assignment
(1)	2 hrs per assignment
(1)	read the essays 3–4 times each (no time specified)
(1)	$1\frac{1}{2}$ days
(1)	not much

Table 2: Time spent marking assignments

The class representative noted that “*students need guidance as to how to deal with writing when English is clearly a second language.*”

More importantly, a number felt that students actually marking each other was unfair, and they felt uncomfortable with this contributing to their final marks. Two students indicate that they felt it wasn’t always clear why marks were lost, one felt that “*every-one is competing so they would mark more harshly.*” One commented that: “*Real lecturers do not drag you down for minor things.*” An important comment was: “*It was difficult to tell whether what a person had written was correct if you hadn’t found that information yourself.*”

Eighteen students said they were happy with the grades they received, while ten were dissatisfied. Three did not respond to that question.

Four responded that the exercise was a waste of time being scheduled just before tests and other assignments that needed attention. A number of students felt that they had not had enough time to do the marking. Nearly all the respondents noted that they took considerable time and made an effort to provide comments and feedback in their marking. But correspondingly they nearly all said they received very little or no feedback on their returned essays. One suggested that six hours was a lot of time and that they should perhaps have had three to mark rather than four. There was a range of reported time spent marking assignments (see Table 2) from “*not much*” to 1.5 days (five did not respond).

#### 4.1.3 Mark reliability

Approximately twenty five assignments were randomly selected and cross checked by departmental staff. With one exception, all assignment marks varied by less than 12%; much of the discrepancy was due to differences in referencing interpretations. Twelve more assignments were remarked upon request of students. For the majority, the new mark differed by  $\pm 5\%$  from the original mark, although there were three where the difference between marks was more than 10%. It was common for some or all markers to have given different marks for specific sections, but typically their overall marks agreed very closely.

## 4.2 English

A medium-size class in English used Aropä in the second semester of 2005 and again in the first semester of 2006. The course was called “Rhetoric in Public Culture.” Students worked individually on a topic selected from six or seven broad subject areas. The peer assessment exercise was a formative review of a draft. A small number of marks were awarded for participating in the activity; no attempt was made to

distinguish the quality of the participation. In both cases 80 students participated. Students were allocated three reviews each, and were given the option to write a self-review. Allocations were assigned randomly.

In writing their essays students were required to think carefully about the intended audience. The course lecturer was very interested in the power relations. He mentioned that “*the student has to feel sufficiently authoritative*” to comment and that this authority was created for them by the peer assessment task, in that they were the public audience. It seemed that many of the students who enrolled in this course aspired to careers as critics, reviewers, editors, opinion columnists, etc. Many of the students therefore had an interest in the genre of the review itself.

The lecturer expressed reservations about drawing up a rubric prior to the exercise:

*as with filling in [academic performance review] forms, it organises you to mentally deal with the box. What happens if you find yourself in between boxes? If you are a creative person this is very likely to happen.*

Furthermore, he argued that:

*while I am keen to make things transparent, the thing is organic, if you get a mechanical object it's an untrue reflection. Students learn about writing over the course of a whole degree... A template can lead you to believe that there is a mechanical process involved, that writing a piece is an assemblage of things. What happens to criticality, creativity?*

In the end, he drew up the following rubric (part of this can be seen in the screen shot in Figure 5):

- Write at least one sentence in response to each of the five questions below (making 300 words altogether) with regard to the draft essay.
- What is the issue that the draft is addressing? Is it interesting, or do you care?
- Say what you think is the argument of the draft. If the argument is not clear, suggest what a possible argument might be.
- What reasons does the writer offer to support the argument? (You may like to break down the argument into quasi-syllogistic premises or to identify a Toulmin-style warrant for the argument).
- Suggest a counterargument to the argument of the draft. This comment may, alternatively, point out unexamined assumptions and/or missing or unacknowledged evidence.
- Identify a characteristic sentence of the writer. Say what you think is good about this sentence, or how this sentence can be improved (your chosen sentence may simply identify a repeated writing fault)

These questions show that one of the problems raised in the Pharmacology course had been addressed by this lecturer; i.e., the close relation between the requirements of the assignment and the organisation of the rubric.

#### 4.2.1 Survey and interviews

All students were invited to fill in an on-line survey, both before and after the exercise. Twelve students completed the survey prior to the exercise, and ten answered after (four answered both before and after). We also interviewed eight students before and again after the 2005 exercise. These volunteers were self-selected, seven female and one male.

The fact that the rubric developed by the lecturer was integrally connected with the purpose of the assignment, combined with the fact that the students were not doing summative marking of each others' work, and that they were able to improve their essays by drawing on the peer reviews, led to quite different data from the previous example. In general it would be very difficult to divide up those responses which were negative and those which were positive. We will, however, attempt to address the same themes as in the previous example.

On the issue of preparation, responses indicated that the students were well-informed about the purpose of the exercise and felt relaxed about what their role was to be. The responses that we received seemed to indicate a genuine sense of curiosity about what reviewing would offer, and how they could improve their work by reading the reviews.

However, there was considerable anxiety about having to “show” their work to others and in the surveys anonymity was frequently mentioned as being important to their comfort:

*I'm very glad my essay was anonymous. Totally hate the idea of anyone other than the lecturer/tutor reading my work and was considering losing the mark and not uploading my essay at all. Now that it's over though I guess it's good to get some feedback on ur work and to see how other people are tackling it.*

In the interviews, however, students were somewhat more open to the idea of not doing it anonymously.

There were few indications of students feeling intimidated by the technical requirements. A few software problems were encountered with the 2005 activity. Some students submitted documents in formats that were not supported by reviewers' computer systems (e.g., TIFF scanned images of hand-written notes), and there was a brief network outage. No problems were reported in 2006. Despite coming from non-technical backgrounds, the students were comfortable with using computers and with the Aropā interface.

The students were very positive about the marking rubric and its relation to the purpose of the assignment. They found the questions helpful, although a number did indicate that they would have preferred to have one open space where they could have commented generally. Having to re-state their assessee's argument and then articulate a counterargument seemed to be a valuable learning exercise. One student said that in re-stating the argument:

*I didn't just copy things out from what they'd written... but tried to read and put it into my own words so that they would have an idea of what the reader was getting out of it. That was quite important. Yeah, I felt like I read them very carefully, being able to summarise the main points, putting it in your own words, being able to identify what the writer is trying to do... seeing the argument as a piece of writing that's been written in*

*a certain way for a reason and then try to think about what that reason might be.*

She also stated that the counter-argument approach: “forces you to show that you’ve read it really carefully.” She saw value in being able to separate out her own views on the argument by having to articulate the counter-argument, and had noted that point in class:

*Yeah that’s what having the counter-argument makes you do...even though I didn’t believe in the counter argument, but I’d noted that down that we still had to do that, so I think that the exercise kind of forced you, its kind of important to do that.*

Students reported concerns about causing offence in the review but one responded that:

*I think I was a lot nicer than the people who reviewed me! Instead of saying I don’t care about your topic I said ‘This would be interesting to people who...’ or ‘I would be interested if I...’ Constructive criticism is good though.*

They largely expected to be good at reviewing, and were able to identify the skills needed for reviewing as: being critical, detached, analytical and logical. There was much interest in comparing their own performance to the rest of the class: “First essay reviewed made me feel good” (i.e., reassured); “I was relieved that others were not way better than mine.”

Students were not involved in the marking but we discussed the issue in the interviews:

JH: *You say that they were all really good. Would you be able to rank them?*

Student: *Definitely...they were all pretty good. It’d be quite hard but if I went back to them maybe ... if I look back at them now at this stage I’d give them all As.*

CK: *How did that make you feel, about your own essay?*

Student: *It made me want to look at it more carefully... ( ) about the argument I was making and see how much research other people had done. I thought well, maybe I should go back in, like I had done research but I hadn’t included it in the bibliography and everyone else had’it looked like maybe I’d hadn’t read anything and made all this up, so it made me reassess what I’d done.*

There were very positive responses in having multiple perspectives on their writing. Their lecturer had indicated the importance of this as well. While being clear that he would be allocating the mark at the end, he said that he was:

*...interested in the drafting mode... this way I’ll save myself from looking at the drafts. I can’t read 100 drafts. The idea of this is that you get a full response from other students. If it’s just one to one, you’ll get your usual range of A to D students, but this thing offers three to five responses for one paper. Across the five responses things will come up. This is a course on rhetoric in public culture. The groups of five act as the public audience.*

The students had completely ‘taken on’ this perspective. Comments on the value of the exercise were:

*“(It’s) to distribute a work to a wider public audience to test your rhetoric and better understand the audience you are trying to persuade. If you just write for a lecturer there is no feeling of audience.”*

*“The task of looking at one’s own writing from an objective perspective in order to improve it or see things that have been missed is a hard thing to do. Sometimes we tend to get into a pattern of blindspots and it is therefore possible to read something a hundred times and still miss the same basic mistake every time. The value of both getting more than one set of other eyes on the writing as well as the experience of analysing and critiquing someone else’s essay gives a much better insight in these aspects of own writing and writing in general.”*

The marks for taking part in the exercise motivated many of the students we asked, but they also mentioned: mutual benefit; receiving feedback; skill training; improving their essay; feeling obligated to provide a review for other student who had done likewise; and curiosity.

In the interviews we asked students whether doing the reviewing had given them new insights into the marking that their tutors or lecturers did:

CK: *Has it given you any more insight into the way your lectures or tutors might mark or approach the marking? Can you try and tell us what that might be?*

Student: *Well, I guess you have to almost put yourself into their shoes, so you have to think if I was going to grade this, these are the things they would look at, I guess maybe look at the argument as a whole and kind of... I guess when writing it you feel you’re really IN there and you’re paying attention to every small detail and even though it might make sense in your own head when you read other people’s and then look at theirs and then come back to yours, you’re able to look at it with a slightly broader perspective.*

In response to the feedback they received, one participant reported that the consensus view of a post-assessment tutorial group was “better to be nailed by peers than by the lecturer.” Another participant said “tutors can be ‘beyond’ where you’re at; it’s good to have reviewers at your own level.” There was also an acknowledgement that lecturers don’t always have time to give extensive feedback, and rarely look at draft work. Our study participants largely felt confident in rejecting misguided feedback, and in some cases adopted writing style or ideas from their peers.

There were no comments on the timing or the quantity of work required. In general we can say with confidence that the students reported being happy with the peer assessment process. Complaints largely related to not receiving all three reviews, due to other students in the class failing to fully participate in the exercise. The exercise was repeated in a largely identical manner in 2006. Although we did not interview students again, we did observe a significant increase in written comments. This coincided with the introduction of a WYSIWYG Javascript editor into Aropä that made it easy for reviewers to add section headings, font styles, itemised lists, smiley faces, etc. to their comments.



The web interface was easy to use	
The system responded quickly, and did not leave me waiting	
I feel comfortable with other students reading my work	
I feel comfortable assigning marks for other students	
Marking other students' work helps me spot mistakes in my work	
The comments from other students were helpful to me	
I would like more assignments to be peer marked	

Table 3: Likert scale survey questions and results for Computer Science. The graphics show the distribution of responses, Agree on the left and Disagree on the right, darker for Strongly.

### 4.3 Computer Science

Aropä has been extensively used in our introductory programming course for three years. The course enrolls around 400 students in the first and second semesters, with a further 200 students taking the course in the summer semester. Peer assessment has been made a routine component of the courses. After each programming assignment, students are expected to complete three or four reviews, one of which is now typically a sample solution. The grading rubric is the same as used by the course markers, who also use the Aropä system. Marks are awarded for participating in the peer review, with some allowance made for the quality of their reviewing. The grade for the programming task is determined solely by the course markers.

An anonymous survey asked participants to respond with (strongly) agree, neutral, (strongly) disagree to a set of questions (see table 3), and for their open-ended comments on:

- What did you like most about the peer marking system?
- In what ways could the peer marking system be improved?
- Any other comments you would like to make.

The survey attracted just under 300 responses from 155 individuals. Most of the suggestions for improvement concerned interface and functionality issues that have since been addressed. Many students commented that the workload imposed by the peer assessment activity was too high, often taking longer to complete than the assignment.

*1 or 2 assignments to mark 5 is waaay to much time. (i spent 1.5–2 hrs)*

Others felt they learned all they were going to from writing the first few reviews:

*we need to mark 6 other student's assignments is too much, waste lots of time and learning nothing, I think marking 2 assignments would be enough*

Several issues were identified with the grading rubric:

*More choices needs to be available, the "grades" that I can choose from does not include all possible errors that can be made. comments should not be necessary when all the assessment is correct*

mentions	issue
103	interface issues
47	workload is too high
28	problems with the grading rubric
18	dubious competence of peers
5	provide sample solutions
4	discomfort in writing comments
4	non-specific dislike
1	lack of feedback
1	inappropriate activity

Table 4: Summary of the open-ended improvement feedback from the Computer Science survey

*Parts of the marking schedule were quite brutal... for example... int Q2 one might perform 1 wrong spelling but the rest is perfect, yet receive no marks*

There were some concerns with the competence of their peers:

*Comments i recieved i found to be largely irrelevant and some left me questioning my peers' ability to accuratly mark my work.*

The remainder included suggestions for incorporating sample solutions in the review allocations, feelings of discomfort in reviewing, complaints about the lack of feedback received, and some non-specific indications of dislike. One student commented they felt the activity was inappropriate. The results are summarised in table 4.

The survey participants suggested a number ways in which they felt the peer assessment activity was of benefit. The most commonly mentioned was simply being exposed to a variety of coding styles:

*You get a whole lot of different viewpoints to the assignment solution and it makes you think of all the other possibilities you could have used.*

Learning occurred both from exposure to good code and exposure to mistakes:

*So that I can see the mistakes that people make so I can improve myself. Also by looking at the "tops" coding can help me learn.*

*Felt like a waste of time in the beginning, but it really is an excellent way of improving. I didn't realise the value of good comments and easy-to-understand code till I got a bunch of hard-to-understand code to mark. It really is very helpful.*

The material learnt was both specific to the assignment and more general:

*I learned very quickly the mistakes that I had made in my own programs, when marking the other people's work*

*It aided in the learning of certain aspects such as style and code conventions.*

The feedback from reviewers was variable, with some students writing detailed comments and others providing only shallow or empty responses. The students allocated conscientious reviewers were generally grateful for the feedback:

*you get feedback from fellow students, which is interesting to read*

Comparing their own performance to that of their peers was another frequently mentioned benefit. This ranged from plain curiosity, to confidence building, through to awareness of an audience (a theme noted much more strongly in the English class):

mentions	like
40	exposure to a variety of coding styles
32	learning examples of good coding
32	non-specific positive comment
35	helpful feedback received
24	the system was convenient and easy to use
20	learning to identify poor programming constructs and mistakes
19	improving (debugging) their own code for this assignment
19	comparing own performance to peers
15	helpful reading code
10	helping others by giving feedback
5	learning by marking
2	the exercise motivated them to work harder
2	gaining an insight into the marking process
2	anonymity relieved concerns about fairness

Table 5: Summary of responses to “What did you like most about the peer marking system?”

*Discovering the level of understanding of program writing of other students*

*The opportunity to gauge my work against that of other students.*

*... got me thinking more about my code that I had previously — especially towards readability and reasonably named variables. Suddenly the use of ‘foo’ as a variable name isn’t as effective when several people are looking at your code!*

The results are summarised in table 5.

One (unexpected) consequences of the routine use of peer assessment in this course was the ability to identify capable student reviewers, who were then offered employment as course markers in the following semester. The lecturers observed that complaints about marking reduced very significantly, from between twenty and forty after each assignment to virtually none. This was felt to be due to students’ increased awareness of the marking process, and more capable markers being employed. The use of Aropä by the course markers also allowed their progress to be monitored by the lecturer and tardy markers chased up.

## 5 Summary and conclusions

Hanrahan & Isaacs (2001) have noted that although the range of general studies on peer assessment is growing and some general trends can be noted, peer assessment has to be implemented on a case-by-case basis in varying subjects and contexts. They have commented that “case-based literature in this area is still alarmingly sparse.” The development of Aropä and our study of its uses in three different courses at the University of Auckland, using the same web-based tool, enable us to make some comparisons and draw some conclusions.

Aropä has been deployed in a very wide range of contexts, and has proven acceptably versatile. Students reported that peer assessment contributed to learning at many different levels, as a consequence of:

- perceiving writing as a public activity;
- reflecting on the grading rubric, which invites questions of “what is important;”

- exercising judgement in awarding marks;
- encountering examples of good style or technique;
- encountering poor solutions that reveal mistakes to avoid;
- altruistic feelings of serving a valuable community role;
- receiving accurate, constructive, timely feedback;
- selecting between worthwhile and misguided feedback.

We observed very different reactions from students in the three subjects. Pharmacology students showed the most resistance. Computer Science by-and-large accepted the activity. In English, it was positively received as a natural part of the course. The reasons for this are complex and will require further analysis, but some discussion is provided in what follows.

Boud et al. (1999) point out that “assessment is the single most powerful influence on learning in formal courses, and if not designed well, can easily undermine the positive features of an important strategy in the repertoire of teaching and learning approaches.” In summarising the literature in the field of assessment they draw up the following effects of assessment on learning.

- Individuals are emphasised

They argue that the overriding paradigm of assessment is that it is an “individual and competitive” process in most institutions. While there has been something of a shift to criterion-referencing, norm-referencing still dominates. If individualistic views of assessment are dominant, collaboration can be seen as ‘cheating.’

- Assessment exercises power and control over students

They state that assessment is the principal mechanism whereby staff exercise power and control over students. The effect of this on learning is to circumscribe it to the range of outcomes unilaterally defined as legitimate by staff. Students learn first to distrust their own judgements and then act as agents to constrain themselves.

- Assessment exerts a backwash effect on learning.

The authors cite the work of Marton et al. (1997) and they argue that “inappropriate forms of assessment appear to encourage students to take a surface approach to learning... conforming to the narrowest interpretations of assessment tasks and working to beat the system rather than engage in meaningful learning.”

- Overload of tasks discourages deep approaches to learning.

Drawing on Ramsden & Entwistle (1981), Boud et al. (1999) argue that overloading contributes to students taking a surface approach to learning tasks, and caution that overloading can lead to peer learning tasks being either ignored or falling into disrepute.

- Assessment practices need to be matched to outcomes.

They argue that outcomes need to be designed in terms of basic knowledge, understanding, communicative and competency aims being pursued in a course.



- Formal assessment processes should encourage self-assessment

They point out that assessment in higher education has a dual function of judging for the purpose of providing credentials and for the purpose of improving learning. With regard to the latter, they claim that assessment should leave students better equipped to engage in their own self-assessment.

The authors also argue that there are both pragmatic and principled reasons for the current focus on peer learning in university courses. Yorke (2003), writing about formative assessment, identifies these pressures as including:

- An increasing concern with attainment standards, leading to greater emphasis on the (summative) assessment of outcomes;
- Increasing student/staff ratios, leading to a decrease in the attention given to individuals;
- Curricular structures changing in the direction of greater unitisation, resulting in more frequent assessment of outcomes and less opportunity for formative feedback;
- The demands placed on academic staff in addition to teaching.

At the University of Auckland these pressures take the form of pressures to both intensify teaching work and at the same time improve quality, in relation to criteria like the fostering of critical and independent thought, creativity and imagination, communication and research skills.

It is clear that the students' uptake of peer assessment in our study reflects these pressures and trends in complex and contradictory ways. The design of the peer assessment tasks across the three cases provides insights into the role of peer assessment in relation to these general trends.

The Pharmacology students (who were required to provide marks for each other worth 20% of the total mark) responded (in a more aggrieved way) to what they believed were the pragmatic reasons; i.e., suggesting that they were having to do the lecturers' job, but realising the benefits for learning as they engaged with the task. The responses also suggested that they felt overloaded at the time of doing the task and became preoccupied with concerns over the fairness of the system.

Much less was at stake with the Computer Science and English assignments. Computer Science students could see some personal benefits in using the system. In English, the students reflected an awareness of the public role of their work and recognised the pedagogical value of the close matching of the assessment with the outcomes and realised the demands of the task as they internalised this matching. It must be noted that their task was formative, implicitly led to self-assessment and only counted for an ungraded 5%.

It may be, therefore, that the reversal of power relations implied in the case of the Pharmacology students was a little too much, too soon and that it is important to recognise that the dominant assessment paradigm to some extent, constructs students' responses. A lecturer in Computer Science put this point well:

*There's a difficulty primarily in marketing it. Getting students on board is difficult. It has to be done carefully, it needs to be approached not as an evaluative tool but pitched as a way of learning. The students see the potential as long as its pitched appropriately.*

The issue of the assessment exerting a backwash effect links closely with the matching of the assessment tasks and the outcomes which we suggest were seen by the lecturers in our study as dilemmas. The English lecturer's comment that writing is something students learn across the years of their time at university, not in a one-off course or task, is important. It is possible that the definition of outcomes for courses and for assessments can give out triumphalist messages about what one exercise can achieve, while at the same time the construction of, and the language used in rubrics potentially promotes inappropriate or reductionist ideas about the matching of assessment and outcomes.

Our data seems to suggest that students are very aware of these issues. We find it heartening to see students' ability to read in between the lines of these pressures on their lecturers. They respond both on the pragmatic terrain, but also indicate their desire for deep learning and reflection. The very sophisticated comment made by a Pharmacology student speaks volumes about this backwash effect and the possible constraints of the rubrics in giving expression to the matching of the assessment tasks and the outcomes:

*My assignment would be quick and simple to mark, ordered according to the marking schedule, rather than the assignment outline, and I could make sure that everything the marking schedule required was in my assignment, and nothing more. No extra research, nothing interesting; i.e., use bullet point format to create a piece of work designed only to meet the requirements of a marking schedule.*

A Computer Science lecturer specified the problems with the rubrics clearly:

*I'd recommend to others now to try and make rubrics that are reasonably vague and general. When you are too specific you eliminate some of the critical thinking and it becomes a mechanical process. Some of the early rubrics we built, we could have got a computer programme to mark them! In writing code, the style is much greater than the sum of the parts. Sometimes the code meets the specification, but it's just being done badly — it's very difficult to write a rubric to capture that. If it's very open, then it can help if the students are able to discuss the marking criteria, and we use a wiki so that they can do this.*

The English lecturer's decision to engage his students with both humorous and reflective questions provides a sophisticated solution to these dilemmas, and the students responded very well to it.

Finally, we do not have clear-cut data to back up this claim, we gained the impression that the organization of the peer assessment exercises through the automated tool of Aropä lent a certain formality and anonymity to the tasks, which gave peer assessment greater credibility. A number of students had commented on how they had done peer assessment informally in tutorials by exchanging their written work with fellow students. Many limitations to, and problems with this were indicated. Aropä 'puts' their work into a new and neutral context, and we suggest that this in itself is helpful if peer assessment is to be taken up more widely.

## 6 Acknowledgements

We would like to thank our interview subjects for sharing their views so openly. The anonymous reviewers provided extensive, thoughtful comments, for which we are grateful. Particular thanks to Paul Denny, Jeff Keelan, Andrew Luxton-Reilly and Stephen Turner for agreeing to use Aropä in their courses.

## References

- Ballantyne, R., Hughes, K. & Mylonas, A. (2002), 'Developing procedures for implementing peer assessment in large classes using an action research process', *Assessment & Evaluation in Higher Education* **27**(5), 427–441.
- Barrie, J. (2006), '<http://www.turnitin.com/>'. Accessed 10 August 2006.
- Bloom, B. S., ed. (1956), *Taxonomy of educational objectives: The classification of educational goals. Handbook I, cognitive domain*, Longmans, Green, New York; Toronto.
- Boud, D., Cohen, R. & Sampson, J. (1999), 'Peer learning and assessment', *Assessment and Evaluation in Higher Education* **24**(4), 413–426.
- Dochy, F., Segers, M. & Sluijsmans, D. (1999), 'The use of self-, peer and co-assessment in higher education: A review', *Studies in Higher Education* **24**(3), 331–350.
- Hamer, J., Ma, K. T. & Kwong, H. H. (2005), A method of automatic grade calibration in peer assessment, in A. Young & D. Tolhurst, eds, 'ACE'05 Australasian Computer Society Education Conference', Vol. 42 of *Conferences in Research and Practice in Information Technology*, Australian Computer Society, pp. 67–72.
- Hanrahan, S. & Isaacs, G. (2001), 'Assessing self- and peer-assessment: The students' views', *Higher Education Research and Development* **20**(3), 54–66.
- Marton, F., Hounsell, D. & Entwistle, N., eds (1997), *The Experience of Learning: Implications for Teaching and Studying in Higher Education*, 2nd edn, Edinburgh: Scottish Academic Press.
- Moxiecode Systems AB (2006), '<http://tinymce.moxiecode.com/>'. Accessed 10 August 2006.
- Ramsden, P. & Entwistle, N. (1981), 'Effects of academic departments on students' approaches to studying', *British Journal of Educational Psychology* **51**, 368–383.
- Topping, K. (1998), 'Peer assessment between students in colleges and universities', *Review of Education Research* **68**(3), 249–276.
- Yorke, M. (2003), 'Formative assessment in higher education: Moves towards theory and the enhancement of pedagogic practice', *Higher Education* **45**, 477–501.

# Mobile Computing, Programming, Games and Online Communities: Changes to the Communicative Ecology of Design Students Through Mobile Computing: Part 2

**Margaret Hamilton**

School of Computer Science and Information Technology

**Marsha Berry**

School of Creative Media and Design

RMIT University, Melbourne

GPO Box 2476V, Melbourne VIC 3001, Australia

{margaret.hamilton, marsha.berry}@rmit.edu.au

## Abstract

In this paper, we discuss the influence the increased mobility afforded by Tablet PCs has had on the way multimedia students learn programming concepts and practice skills and techniques through their communicative ecology. Following on from our previous ethnographic action research study, on mobile computing, visual diaries, learning and communication, (Berry & Hamilton, 2006) we allocated Tablet PCs to a new group of students and encouraged their use particularly during their programming classes. We applied many of the instruments recommended in the literature as predictors of success in a programming course to help define and explain the communicative ecology. As well as these instruments, we invited the students to focus group discussions and solicited information via email questionnaires.

**Keywords:** Communicative ecology, ethnographic action research, Tablet PCs, programming.

## 1 Introduction

Most teaching practice in Higher Education today is still typified by a transmission or absorption view of learning rather than an interaction or constructivist one, with the passive lecture still the most predominant mode of instruction. Many teachers have not yet made the move from a teaching to a learning paradigm (Barr & Tagg, 1995; Tagg, 2003) despite most institutions promoting the latter for some time now. Therefore, engaging teachers in the design of blended-learning environments holds great promise for learning and teaching, since it has been associated with an increase in the use of strategies that are student-centred and that promote active learning, including increased peer interaction. It has also been

shown to increase learner flexibility while retaining the richness of engaging with others in a social learning setting (Tutty & White, 2006).

In addition, students report greater satisfaction with participation in blended learning environments. For example, White and Tutty (White & Tutty, 2005) report consistent positive student feedback with 90% of students (N=127) reporting that they preferred this approach to a traditional classroom one. What different forms of blended learning environments have in common is an increasing reliance on Ubicomp (ubiquitous computing) and various computing devices that may signal a shift towards the Learning paradigm as conceptualised by Barr and Tagg (Barr & Tagg, 1995) and developed by Wagner (Wagner, 2005) :

*The reason for optimism is simply this: whether we like it or not, whether we are ready for it or not, mobile learning represents the next step in a long tradition of technology-mediated learning. It will feature new strategies, practices, tools, applications, and resources to realize the promise of ubiquitous, pervasive, personal, and connected learning. It responds to the on-demand learning interests of connected citizens in an information-centric world. It also connects formal educational experience (e.g., taking a class, attending a workshop, or participating in a training session) with informal, situated learning experience (e.g., receiving performance support while on the job or taking advantage of what David Metcalf has called "stolen moments for learning" while riding the train or sitting in an airport waiting for a flight).*

Our vision is the creation of a blended learning environment through the provision of Tablet PCs to multimedia students learning programming so that they may steal moments for practicing programming in their everyday routines.

## 2 Background

As part of their Program, the multimedia design students are required to learn introductory programming, in

---

Copyright © 2007, Australian Computer Society, Inc. This paper appeared at the *Ninth Australian Computing Education Conference (ACE2007)*, Ballarat, Victoria, Australia, January 2007. Conferences in Research in Practice in Information Technology, (CRPIT), Vol. 66. Samuel Mann and Simon Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

particular java programming. For this course, they join a large cohort of computer science students, and in the past, the multimedia design students have fared badly when their average results are compared with the computer science students.

In the first stage of this study, we investigated the ways the students learned their design concepts and approached their development work (Berry & Hamilton, 2006). We reported that the Tablets encouraged more frequent contributions to their visual diaries, enhancing their reflective practice within the design process and enabling them to share their preliminary designs with their group and client. In this second stage, we are expanding the investigation to include their programming concepts.

The communicative ecology of the students is a complex system of networked devices. In general, most have mobile phones and mp3 players and desktop computers connected to the internet via broadband at home. This means that while they can access data on the internet using mobile phones, they are not really able to practise programming whilst on the move. Most students do not have access to mobile computing in the form of laptop computers or Tablet PCs. The opportunities for technology mediated situated learning in a blended learning environment are restricted to listening to podcasts and accessing information on internet sites designed for mobile phones. We wanted to see whether the addition of mobile computing in the form of Tablet PCs would support the creation of a blended learning environment to improve learning programming as outlined above in the introduction of this paper.

### 3 Research

The primary objective of this phase of the study is to answer the research question “How far does increased mobility as a change to the communicative ecology of multimedia students facilitate learning programming?”

#### 3.1 Participants

The participants in this study are enrolled in the second and fourth years of a cross-disciplinary program involving Computer Science, Engineering, Creative Design and Business Schools. This program embodies many of the difficulties in creating a learning community of students who are enrolled in courses from different areas of a University. The students are diverse, with academic interests ranging from interactive media design to software development. After a common two years of core courses they can choose an area of specialisation. The fourth year students were engaged in a group project designing a website with dynamic content for a real client.

#### 3.2 Research Methodology

As part of a mobility grant to our University, HP donated twenty Tablet PCs for student use. These devices have a touch-sensitive screen, which rotates and locks in place, a tablet pen to write on the screen and enter data, and a keyboard to type data, a wireless LAN and Bluetooth configuration, standard USB keyboard, battery pack and

power connections. The Tablets have the capabilities of a laptop but are lighter and do not have the CD drive. They run under Windows XP and we installed several packages, which are licensed within the University, including Microsoft Office – Word, Excel, Powerpoint and Macromedia Studio – Flash MX 2004, Fireworks MX, Dreamweaver MX 2004, Freehand MX and Adobe Acrobat.

Following the methodology we developed in the first year of the study (Berry & Hamilton, 2006), page 36), we called for volunteers and distributed the Tablet PCs accordingly. We used a qualitative approach: ethnographic action research where the research cycle comprises four stages that make up an iterative loop: planning, doing, observing and reflecting. We drew on (Tacchi, Slater, & Hearn, 2003) and (Peterson, 2003) to arrive at our notion of a communicative ecology. Ubicomp is a dominant feature of the communicative ecology and encourages the examination of the micro-practices that make up everyday life.

*“The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.” (Weiser, 1991): 1)*

Participants were chosen randomly and were issued with a Tablet for their sole use for the semester. We administered a baseline survey at the beginning of the semester and held a focus group midway through the semester, the questions and details of both instruments are discussed in (Berry & Hamilton, 2006). We also drew on methodologies used in allied studies of computing education to extend our work, in particular the earlier study undertaken in the BRACE (Building Research in Australasian Computing Education) workshop and discussed in (deRaadt, Hamilton, Lister, & Tutti, 2005; S. Simon, Cutts, Q., et al., 2006; S. Simon et al., 2006; Tolhurst et al., 2006).

The BRACE study which was looking for predictors of success in early programming courses, found the highest correlation with the Biggs test and next with a paper-folding test (deRaadt et al., 2005). The Biggs test should be applied at the end of the course, and is a test of students’ aptitude to study as well as whether they are surface or deep learners (J. Biggs, 1987; John Biggs, Kember, & Leung, 2001). The paper-folding test (Ekstrom, J., & Harman, 1976) gives an early indication of the spatial-thinking and logic skills of our participants, which is weakly correlated with programming results (S. Simon et al., 2006). We decided that the paper-folding test would identify differences in spatial thinking and logical ability between mainstream computer science students and multimedia design students learning programming and was appropriate for our research question “How far does increased mobility facilitate learning programming?” We administered this test at the outset and followed up with the Biggs test at the end of the semester. There is evidence that suggests that increased levels of engagement with learning materials may be regarded as a measure of learning (Wagner, 2005), hence we used this as an indicator to establish

whether or not mobile computing was beneficial for students learning programming.

We also wanted to explore the degrees of engagement with other forms of networked computing such as games and open source communities in order to be able to chart directions so that educators can utilise such activities to support learning programming.

In order to further expose the participants' everyday practice and engagement with learning programming we devised virtual interview questions that we administered using email. This ensured maximum privacy for responses and allowed participants to respond individually rather than in a group situation. The questions were designed to gather descriptions of experiences and uses to which the Tablets were put. It was decided to extend the focus of the questions to include multi-user online games to explore the potential of these for learning. These virtual interview questions were as follows:

- (a) How have you used the Tablets for group work? (were you in a group with other Tablet users? did you use it for collaboration, communication between group members)
- (b) What group activities have you performed using the Tablet? ( did you use them for brainstorming, recording meetings, designing presentations, making presentations, email, connecting to the internet using the wireless network , at home, at RMIT, accessing online communities, etc)
- (c) To what extent are you engaged with open source communities for scripting and programming? Please describe situations where you would seek help from an open source community or use an open source solution to a problem (eg content management systems, databases, etc).
- (d) To what extent do you engage with online games, especially multiuser games? Please provide examples.
- (e) To what extent do you engage with online communities associated with the games?
- (f) To what extent does having a Tablet further your engagement with programming exercises (this includes Actionscript)? Would you say you find more time to practice? How often would you practice programming over a week?
- (g) What software have you downloaded and installed, for example, Alice, Eclipse, games? Please be specific. We promise this information will remain completely anonymous.
- (h) When and where do you use your Tablet? Please provide a detailed description of a typical week in Semester 1. Be sure to include entertainment activities as well as learning related activities.

The responses were coded according to categories emerging from and grounded in the participants' responses. These categories were devised post hoc and are elaborated in the results section of this paper.

## 4 Results

For this study we are interested in investigating the learning ecology of multimedia students, particularly for programming. This can be affected by their prior knowledge of programming, so at the beginning of the investigation, as part of the baseline questionnaire, we asked the students to rate their programming experience. They were offered a list of programming languages and asked to estimate their familiarity with them on a scale of 1 to 5 where 1 indicated they have never used the language to 5 meaning they felt they were experts programming in that language.

All of them have used java to some extent, as none of them wrote 1 or 5, however, none of them have used Ada (A) or Scheme (S), and either left that entry blank or filled in 1. For the category of "Other" programming languages some of them wrote html, php and even css and one marked their expertise in html with a rating of 4 while their java and C capabilities were only 2.

	Java	C++	C	A, S	Pas-cal	Vis Basic	Other
Mean	2.88	2.17	1.43	1.00	1.50	1.83	2.80
Std dev	0.64	1.33	0.53	0.00	1.00	1.60	1.30

**Table 1: Student self-assessed prior experience from scale 1 (never used) to 5 (have used a lot).**

We administered the paper-folding test to our students who scored results in the range of 6 to 18 correct answers out of the 20 questions, giving an average of 12.4 and standard deviation of 1.44. In the BRACE study, participants scored a mean of 14.1 with a standard deviation of 3.4 (deRaadt et al., 2005). In the original study (Ekstrom et al., 1976), the paper-folding scores for 46 college students resulted in a mean score of 13.8 with a standard deviation of 4.5.

This indicates that our multimedia students are within the range of the other studies, though their mean is a little lower, which could indicate that they will possibly struggle with programming courses. We were therefore very interested to hear the students volunteer the following information relevant to how possession of a Tablet PC can help learning programming at our focus group meetings.

*"I use time on the train to do programming because I can use that time now."*

*"I use Eclipse [ a platform for Java] because you don't need to be connected to the internet, I find it really useful because you can do programming anywhere."*

*"Having a Tablet is really useful in [programming] lectures. I use the keyboard because I type faster than I can write so it's really useful for making notes."*

By putting Table PCs into the communication ecology, these students have constant access to a computer capable of writing, compiling and running programs. Whether they are travelling on the train, or sitting in the cafeteria, they can open up their familiar programming environment and practice their skills.

Hence programming can become a daily habit enabled by a mobile computing device that it largely invisible, outside of tutorials and laboratory time and extra to the lecture time. Also, since the student is likely to be travelling alone, this practice can be undertaken on their own and provides an extra opportunity to reinforce their personal skills and undertake self-directed learning.

At the end of semester, we asked the students to complete a Biggs questionnaire in order to evaluate the learning environment in our classrooms. As Biggs (John Biggs et al., 2001) wrote:

*"We believe the most effective way of ensuring high quality teaching and learning is for teachers to take responsibility for ensuring that assessment and other contextual elements in the teaching and learning system are constructively aligned to promote deep approaches to learning."*

We asked our students to complete the questionnaire in relation to their approach to their programming subjects. The scores on the Biggs questionnaire are summarized in Table 2. They are included for comparison with data from the BRACE study (deRaadt et al., 2005), however, neither set of results are statistically significant. We cannot say that the students are strongly aligned with either deep or surface learning approaches, though it does appear on average that they may have a deeper learning approach and motive but more of a surface strategy to their programming studies.

	DA	DM	DS	SA	SM	SS
Mean	29.50	15.50	14.00	26.75	11.50	15.25
Std dev	4.73	2.38	2.71	7.89	3.70	4.57

**Table 2: Statistics for students on the Biggs revised two-factor study process questionnaire, R-SPQ-2F, where DA – Deep Approach, DM – Deep Motive, DS – Deep Strategy, SA – Surface Approach, SM – Surface Motive, SS – Surface Strategy represent student approaches and strategies for learning.**

Employing a surface strategy to programming might suggest that these students could try to learn by rote, rather than understand the meaning of certain lines of code, or write their own code creatively. This would certainly be disastrous practice for a programming course. However having a deep motive indicates they have an intrinsic interest and aim to achieve well in this area of learning, so that is a good sign. Perhaps the use of mobile Tablet PCs will encourage their frequent practice and

hence the realisation that programming cannot be learned by rote.

#### 4.1 Practicing programming

Increased mobility has a positive effect on practising programming outside of formal classes. Students found that ready access to text editors and compilers enabled them to get ahead with their programming exercises out of hours. Most students reported that they were more inclined to use travel time more effectively for their study. Only one student reported that he did not practise programming outside of class times.

*"I didn't really practice programming often at home. Only using the uni computers during lab times."*

*"It furthers my engagement to some extent. It's very useful in that I can do a bit of programming anywhere. I find it's great with my java lectures because for one, I don't need to go to the effort of printing the slides out and two, when the lecturer gives sample code I can quickly type them up on my tablet rather than writing it all down, which I'm very slow at"*

*"The tablet would probably help with giving me more practice, although I prefer to do my programming when I have access to the internet, only because most of the time I've already done some work during labs so I need the net to transfer them to the computer I'm using. And also I can search online for help when I need it. So I find I do most of my java programming work in labs or at home where I have the net."*

*"I probably practice java a couple of times a week? Maybe less than I should be."*

*"The tablet allowed me to use travel time to work on my programming. I was programming roughly 6 hour a week. I found the portability handy, it gave me an option to work on my programming anywhere at anytime."*

#### 4.2 Games and games communities

*"I'm not really a game person so I don't play online games."*

On the whole, the students say they do not use their Tablet for games, however, in listing the software they have installed one such respondent wrote:

*"videora ipod converter - Converts videoclips to ipod format"*

*icy towers - really old game*

*Gameboy emulator*

*cooledit pro - A sound recording program"*

as if to say that really old games don't count, and a gameboy emulator is not really a game either.

Regarding the question of software downloaded, many admitted to downloading chat software such as MSN, and some included software for images and watching movies:

*"iTunes, eclipse, photoshop, firefox, Div player, Fruity Loops, google earth, Adaware, MSN, Nero, Power DVD, winSCP, eclipse, winzip, picasa,"*

And from the same student later on:

*"Over the holidays i used it quite often to watch downloaded tv shows and movies (my computer at home does not have a graphics card so it is laggy where as the tablet PC gives a clear picture and the sound is in synch with the video) It is good because it is portable."*

Other students listed software downloaded as including:

*"Eclipse, WinSCP, putty, MSN Messenger, Editpad Lite" and*

*"I've downloaded msn messenger, winamp, itunes, apache server, and someother free-source programs. The programs needed for work already come together with the tablet."*

### 4.3 Engagement with open source and communities

We found that the students were not really looking for ways to collaborate or engage with online professional communities. Rather they are forging ahead to solve a problem before unpacking it and finding out if there was previous knowledge that they could exploit and extend. Hence they often jump into finding solutions without really understanding the problem solving process itself, nor what works and what does not work within a typical problem solving process.

*"I do not engage with open source communities for scripting and programming."*

*"I use the blackboard [the University's learning management platform] for help with java."*

*"when developing the website for the bdms course, i did not have any prior knowledge about it therefore everything i learnt is from open source community, mostly regarding database and php scripting."*

*"I used it for brainstorming ideas and recording a meeting for a group presentation. I connected to the Internet using the [University's] wireless network where possible, including Melbourne Central's food court."*

*"I used the tablet once in a group situation for Marketing last semester. It was used to show the rest of the group some designs I came up with for a particular project. There were no other tablet users in the group. It was solely used for communicating my designs to the other members."*

### 4.4 Typical week

We asked the students to describe a typical week to ascertain the effect the Tablet PCs have on the communicative ecology. The descriptions below indicate that the Tablet PC enabled Ubicomp where the technology enabled networked activities in a non intrusive manner. In other words, students no longer needed to seek out computers connected to the internet to undertake their studies.

*"Last semester I brought my tablet PC to the majority of my classes."*

*"I would use it during lectures to access lecture slides and for some note taking."*

*"I brought it to design class to transfer files I had been working on and also to show updates of my work."*

*"I've used it a lot in lectures. Probably brought it to every lecture. Also in tutes. I like typing things better than writing, only because I'm a slow writer and I like being able to change something or add something in without making things too messy, like if I were to write my notes down for example. During breaks it's been really good because my friends and I would usually go to Melbourne Central upstairs food court for lunch and we have access to wireless there so I could go online without needing to go to a lab and I could eat my lunch too =) I use it in bed. On the train sometimes. Mostly at uni though."*

#### Typical week.. (semester 1)

*"Mon: brought it for java lec."*

*Tues: design lab and lec - sometimes I brought it. Not every week though."*

*Wed: day off*

*Thurs: java lec, tute, lab. Used it all day."*

*Fri: marketing class and web pg tute/lab. Used it all day too."*

*"I was using my tablet in the programming labs as it was easier to access and continue on with projects I was working on at home. I found it easier to demonstrate my programs on the tablet because I was accustomed to the Eclipse version I was using."*

*"I used it in the cafe to browse the internet, usually to read news websites like The Age and BBC."*

*"I used it on the train to read some .pdf books i downloaded."*

*"-i used the tablet almost everywhere. I'm pretty mobile about the tablet, including sitting down in the park and drawing stuff using it. It just feels cool using it in public places."*

The responses to where and when do you use your Tablets were as wide and varied as the students themselves:

*"Last Semester, I would at times, use my tablet PC to practice a bit of Java programming in bed, when I didn't feel like using my desktop PC downstairs. Other than that, I don't use the tablet a lot."*

*"I use my tablet every day and for most assignments except those that require lots of graphics (hi res photoshop). I watch DVD's on it (portable DVD player), connect it to my phone and iPod, download music from iTunes, check my email, use the internet, connect to my wireless network, surf the internet at uni and at home"*

*"i used the tablet almost everywhere. I'm pretty mobile about the tablet, including sitting down in the park and drawing stuff using it. It just feel cool using it in public places."*

And all the following from one student:

*"Last semester I brought my tablet PC to the majority of my classes."*

*"I would use it during lectures to access lecture slides and for some note taking."*

*"I brought it to design class to transfer files I had been working on and also to show updates of my work."*

*"During lunch breaks at Melbourne central to access the internet"*

*"I used it on the train during exam time to study past exams (so i didn't have to print out pages and pages of exams)"*

*"Used it in state library to take down study notes and work questions from."*

*"Used to do random drawing in the library."*

*"I used my tablet PC to do my entire flash animation assignment. It helped save alot of time (although in flash the pen does not give an immediate response while drawing)"*

## 5 Conclusions and Future Directions

Computing education has typically focussed on individual skills development. The sharing and development of other people's code is generally discouraged in assessment. Yet within the ICT and multimedia industries, teamwork and collaboration is the dominant professional practice paradigm. Developers and designers are constantly building on the work of others to implement new solutions to business problems. Often, they turn to online open source communities for assistance. Software engineering is also shifting towards open source models (Hiltzik, 2004) (Foundation, 2004) (Taft, 2004), Linux is preferred over Microsoft and MySQL and SQL are the preferred databases over Access.

Evidently there is a gap between programming education and industry practice. This gap is not as wide in design education as the studio model encourages collaboration and reflection on the efficacy of problem solving strategies throughout the production process, including pre-production. We would like to argue that the introduction of mobility into the communicative ecology of students along with technologies such as blogging facilitate reflective practice and sharing of ideas in design but this effect did not flow through to their programming.

Before students can fully take advantage of the benefits of mobility and connectivity, they need to shift their thinking about learning to a more active mode where they take responsibility for unpacking problems into constituent elements and seek input from outside expert sources and communities who have already encountered similar situations. Such a shift in thinking about knowledge would also elevate problems to more abstract levels where the focus is about, for example, object manipulation in general rather than on specific lines of code or scripting to make the object perform a particular task or action.

It would need to be supported by an overhaul of assessment with more emphasis on mapping the stages in the problem solving process. For example, programming students could use blogs to articulate abstractions of a programming problem and the various approaches one could use to solve the problem. Such an innovation with regard to assessment would also help build integrity in students as they will feel ownership of the thinking behind a specific coding solution. Risks of plagiarism would also be reduced because the thought process is highlighted in the production of code.

Further, students would realise that for every problem there are numerous solutions, depending on how one approaches it: there is no one absolute way to write code anymore than there are absolutes in design.

## 6 Acknowledgements

This research is partly funded by HP through their HP Mobile Technology for Teaching Grant Initiative which provided the twenty Tablet PCs for student use.

## 7 References

- Barr, R. B., & Tagg, J. (1995). From Teaching to Learning - A New Paradigm for Undergraduate Education. *Change Magazine*.
- Berry, M., & Hamilton, M. (2006). Mobile Computing, Visual Diaries, Learning and Communication: Changes to the Communicative Ecology of Design Students Through Mobile Computing. *Conferences in Research in Practice in Information Technology*, 52, 35-44.
- Biggs, J. (1987). *Student approaches to learning and studying*.: Australian Council for Educational Research.
- Biggs, J., Kember, D., & Leung, D. (2001). The Revised Two-Factor Study Process Questionnaire: R-SPQ-2F. *British Journal of Educational Psychology*, 71, 133-149.



- deRaadt, M., Hamilton, M., Lister, R., & Tutti, J. (2005). *Approaches to Learning in Computer Programming Students, and its Effect on Success*
- Ekstrom, R., J., F., & Harman, H. (1976). Cognitive Factors: Their Identification and Replication. *Multivariate Behavioural Research Monographs*, 79(2).
- Foundation, M. (2004). *Mozilla Home Page*, from <http://www.mozilla.org>
- Hiltzik. (2004). Building a better browser at mozilla. *Los Angeles Times*.
- Peterson, M. A. (2003). *Anthropology and Mass Communication: Media and Myth in the New Millennium*. New York: Berghahn Books.
- Simon, S., Cutts, Q., Fincher, S., Haden, P., Robbins, A., Sutton, K., Baker, B., et al. (2006). The Ability to Articulate Strategy as a Predictor of Programming Skill. *Conferences in Research in Practice in Information Technology (CRPIT)*, 52, 181-188.
- Simon, S., Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., et al. (2006). Predictors of success in a first programming course. *Conferences in Research in Practice in Information Technology (CRPIT)*, 52, 189-196.
- Tacchi, J., Slater, D., & Hearn, G. (2003). *Ethnographic Action Research.*, from [unescoedelhi.nic.in/publications/ear.pdf](http://unescoedelhi.nic.in/publications/ear.pdf)
- Taft, D. K. (2004). Eclipse director takes on challenge of growth. *eWeek*.
- Tagg, J. (2003). *The Learning Paradigm College*. Bolton, MA: Anker Publishing Company.
- Tolhurst, D., Baker, B., Hamer, J., Box, I., Lister, R., Cutts, Q., et al. (2006). Do Map Drawing Styles of Novice Programmers Predict Success in Programming? A Multi-National, Multi-Institutional Study. *Conferences in Research in Practice in Information Technology (CRPIT)*, 52, 213-222.
- Tutty, J., & White, B. (2006). Tablet Classroom Interactions. *Conferences in Research in Practice in Information Technology (CRPIT)*, 52, 229-233.
- Wagner, E. D. (2005). Enabling Mobile Learning. *EDUCAUSE Review*, 40(3), 40-53.
- Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, 94-104.
- White, B., & Tutty, J. (2005). *What, no lectures!?: Experiences from a blended tablet PC Classroom*. Paper presented at the UniServe Science, University of Sydney.



# Quantitative Peer Assessment: Can students be objective?

Nicole Herbert

School of Computing,  
University of Tasmania  
Private Bag 100, Tasmania, 7001, Australia

[Nicole.Herbert@utas.edu.au](mailto:Nicole.Herbert@utas.edu.au)

## Abstract

Team work can have a positive impact on student learning and commitment, but it is challenging to determine a method of assessment that does not require lecturers to involve themselves intimately with each team. Team members are often the best source of meaningful data, and as a result, lecturers are including self and peer assessment. One method of peer assessment is to have team members quantify their own contribution and that of team members. Concerns have been raised in the literature about distribution patterns with this method of peer assessment. An online peer assessment system has been capturing data from a capstone project course for three years with over 24 teams and 100 students each year. This paper analyses the following questions: do students take the easy option of equal distribution to avoid conflict, are students honest about their own contribution, are females treated fairly and are international students unfairly discriminated against.

**Keywords:** peer assessment, teamwork, capstone project.

## 1 Introduction

Rewarding a student with an individual grade is challenging in a capstone project course due to the team and project structure – the work products vary between projects and an individual's contribution can be hard to identify. As stated by Wilkins and Lawhead (2000) many instructors shy away from team project situations, partly because of the challenge to devise a way to assign grades to individual team members. Gates, Delgado, Mondragon (2000) identified the importance of structuring individual accountability to ensure that all members of a team contribute to the project concluding that team members are often the best source of meaningful data.

Software Engineering Project is a 26-week capstone program divided into two consecutive 13-week units; the students get two grades. The students undertake real industry projects suggested by local businesses in teams of 4-5 students, occasionally 3. Students form their own teams and choose a project from the list of suitable projects (Clark 2005). Table 1 summarises the team and student configuration data from 2004, 2005, and 2006.

---

Copyright © 2007, Australian Computer Society, Inc. This paper appeared at the *Ninth Australasian Computing Education Conference (ACE2007)*, Ballarat, Victoria, Australia, January 2007. Conferences in Research in Practice in Information Technology, Vol. 66. Samuel Mann and Simon Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

	2004	2005	2006
Total Teams	27	25	24
Teams of 5 students	21	19	20
Teams of 4 (or less) students	6	6	4
Teams with gender mix	11	10	16
Teams > 2 of both genders	2	2	7
Teams all-domestic students	20	14	10
Teams all-international students	3	5	4
Teams with domicile mix	4	6	10
Teams > 2 of each domicile	1	3	3
Total Students	129	118	115
Females (mixed gender teams)	13	17	24
Males (mixed gender teams)	41	30	52
Males (not mixed)	75	71	39
Domestics (mixed domicile)	11	13	22
Domestics (not mixed)	96	65	47
Internationals (mixed domicile)	8	16	26
Internationals (not mixed)	14	24	20

**Table 1: Team and student data 2004 , 2005, 2006**

The number of female students has increased steadily over the years. The number of international students doubled in 2005 as students from China, who had completed the first 1 – 1.5 years of their degree in China, came to Tasmania to complete the final 1.5 – 2 years of their degree. All international students are temporarily in Australia to study and the majority are Chinese (71%).

In Software Engineering Project students receive a grade that reflects their input into the project. One of the main sources of assessment data is peer assessment. A range of tools are used to allow students to indicate their own contribution and that of team members, all of these are described in detail in Clark, Davies and Skeers (2005):

- Timesheets – an online version of the timesheets described by Humphrey(1997).
- Individual Contribution Reports – a personal report detailing their contribution to a work product. Each student also has to indicate agreement or disagreement (including explanation) with the reports written by each of their team members.
- Self/Peer Evaluation Surveys – students rate each team member (including self) on a list of behavioural characteristics of good team work.
- Work Product Pay Packet – each team member gives a quantitative opinion of how much each team member contributed to a work product.

The timesheets are assessed weekly. The other peer assessments are conducted at the conclusion of a major work product (eg design report). Once the work product has been assessed a meeting is held with the lecturer to receive feedback on the work product and discuss the peer assessment data.

An analysis of the numbers input into the Work Product Pay Packet (WPPP) is the focus of this paper. The approach is similar to that described by Hayes, Lethbridge and Port (2003) and Kennedy (2005) and the numbers are used similar to the way Brown (1995) used the numbers from the Autorating system. Clark (2005) gives a detailed description of the evolution of the tool and how it is used in the assessment process to determine a final mark. In first semester 2004 each student had to distribute 20 marks amongst their team members, since second semester 2004 the students distributed a virtual \$100, using a work product pay packet form, Figure 1. They are told to distribute the amount between team members based on the quality and quantity of work contributed by each individual. In 2004 the WPPPs were used 7 times and in 2005 they were used 10 times and in first semester 2006 they have been used 3 times. There is a pay packet associated with each of the design reports (3 a year) and at least one for each software release (2 a year). Pay packets are also used to quantify contribution to documentation and marketing work products.

Kaufmann (2000), Layton and Ohland (2001) and Hayes et al (2003) all used a process of having students quantify the contribution of team members, but they expressed some concerns about undesirable distribution patterns. Kaufmann (2000) raised concerns about team members forming a pact to give equal amounts to avoid conflict, and concerns about individuals inflating their own performance. Hayes et al (2003) raised concerns that team members may “gang up” on another member (form partial team pacts). Kaufmann (2000) and Layton and Ohland (2001) both raised concerns about gender and ethnicity bias. Kennedy (2005) asked whether peer assessment was worth the effort, concluding that marks awarded to individuals based on peer assessment differ only slightly from equal allocation.

This paper focuses on the data contained in the WPPPs in

	Pay	Comments
Jim	30	Both Claire and I undertook the majority of the release 2 implementation while the others concentrated on the documentation
Claire	30	
Ben	15	
Max	10	
Colin	15	
<b>Total</b>	<b>\$100</b>	

	Jim	Claire	Ben	Max	Colin	Pay
Jim	30	25	30	30	25	<b>140</b>
Claire	30	25	25	25	25	<b>130</b>
Ben	15	15	15	15	15	<b>75</b>
Max	10	15	15	10	15	<b>65</b>
Colin	15	20	15	20	20	<b>90</b>

**Figure 1: Work Product Pay Packet and Work Product Pay Packet Team Summary**

2004, 2005 and first semester 2006, and analyses the following questions: do students take the easy option of equal distribution, are students honest about their own contribution, are females treated fairly and are international students unfairly discriminated against.

## 2 Do students take the easy option of equal distribution?

As shown in Table 2, in 2004 the students gave equal amounts to all their team members 50% of the possible times (number of students 129 x 7 tests). In first semester it was 59% of the time, whereas in second semester it was only 44%. In 2005 the students gave equal amounts 43% of the time. Again there was a drop from first to second semester, but only 8%. In first semester 2006 the students gave equal amounts 23% of the time - a dramatic reduction on the previous two years.

In 2004, females gave equal amounts 37% of the time – considerably less than males, but this was reversed in 2005 and then reversed again in 2006. This indicates that equal distribution is not influenced by gender.

In 2004, international students gave equal amounts 70% of the time considerably more than domestic students. In 2005 this was reduced by 10% and in 2006 this was reduced by nearly 50%. The data does indicate that international students are significantly more likely to distribute amounts equally than domestic students.

There are a number of reasons why students may distribute amounts equally:

1. They genuinely believe the contribution was equal.
2. They are lazy or doing the form in a hurry.
3. They are trying to disguise their own contribution – be it too much or too little.

		N	Possible	% of Possible
2004	Females	34	91	37
	Males	419	812	52
	Domestics	345	749	46
	Internationals	108	154	70
	Semester 1	228	387	59
	Semester 2	225	516	44
	Total	453	903	55
2005	Females	92	170	54
	Males	415	1010	41
	Domestics	269	780	35
	Internationals	238	400	60
	Semester 1	279	590	47
	Semester 2	229	590	39
	Total	508	1180	43
2006	Females	13	72	18
	Males	67	273	25
	Domestics	33	207	16
	Internationals	47	138	34
	Semester 1	80	345	23

**Table 2: Individuals who gave equal amounts to all team members**

4. They disagree with peer assessment and refuse “to do the lecturers work for them”.
5. They have a pact and believe other team members will also distribute equally.

Students are able to write comments on the WPPPs and many students do confirm that they mean to give an equal distribution because they believe this was the contribution pattern. Reasons 2, 3 and 4 are fairly easy for the lecturer to identify after reading the Individual Contribution Reports, Timesheets and the WPPPs from other members. The offenders are quizzed at the meeting with the lecturer and tend not to be repeat offenders. Though students doing it for reason 4 can be hard to dissuade.

Teams that have a pact to distribute equally are a concern. Often teams form a pact to all contribute equally (which is great) and agree to do the distribution equally – sadly the contribution is not equal but members still distribute equally. There are indications that when a team makes a pact that it is hard to break – with 4 teams in 2004 and 3 in 2005 hardly wavering at all. Though in 2004 on only 5 (out of 26) pay packets and in 2005 on only 4 (out of 34) pay packets was the lecturer able to discern a different contribution by each member based on the Individual Contribution Reports indicating that these team members did do an equal contribution most of the time.

As shown in Table 3, in 2004 an entire team gave equal amounts to all their team members 22% of the possible times (number of teams 27 x 7 tests each). In 2005 it was reduced to just 15 % of the time. In 2006 it was only 6% of the time. Teams consisting entirely of international students are most likely to distribute equally. In 2004, the all-international teams gave equal amounts 76% of the time. In 2005, it was reduced to 58% of the time – but a staggering eleven times more often than the all-domestic

teams. In first semester 2006, the all-international teams gave equal amounts only 25% of the time – but all-domestic teams have been reduced to 0%. In 2004 there

		<i>N</i>	<i>Possible</i>	<i>% of Possible</i>
2004	Teams of 5	24	161	15
	Teams of 4	18	28	64
	All-international	16	21	76
	All-domestic	26	140	19
	Mixed domicile	0	28	0
	Total	42	189	22
2005	Teams of 5	26	190	14
	Teams of 4	11	60	18
	All-international	29	50	58
	All-domestic	7	140	5
	Mixed domicile	1	60	2
	Total	37	250	15
2006	Teams of 5	4	60	7
	Teams of 4	0	12	0
	All-international	3	12	25
	All-domestic	0	30	0
	Mixed domicile	1	30	3
	Total	4	72	6

**Table 3: Entire team gave equal amounts**

were three teams that gave the same amount every time, two of them were entirely domestic teams, the other was entirely international. There was another international team that rewarded evenly 5 out of 7 times. In 2005 there were three international teams that gave the same amount 9, 8 and 7 times out of 10. The most an all domestic team rewarded evenly was 3 times in 2005.

The most interesting revelation from all years is that mixed domicile teams (teams with both international and domestic students) nearly never (once only in 2005 and 2006) gave equal amounts to everyone.

Table 3 shows that teams of 4 (or less) are more likely to distribute amounts equally than teams of 5. In both 2004 and 2005 only one team of 4 was entirely international students and none were in 2006. In 2004, teams of 4 gave equal amounts 64% of the time – four times more often than the teams of 5. In 2004 there was one international team of 4 and one domestic team of 4 that gave equal distribution every time. In 2005, there was a significant reduction in the number of times teams of 4 gave equal amounts, but there was one international team of 4 that gave equal amounts 8 times out of 10. In first semester 2006 teams of 4 never gave equal distribution.

Since the teams stay the same and continue to work on the same project it is interesting to consider the four factors that influence the students to not distribute amounts equally.

Firstly, in 2004 a change was made to the amount the students had to distribute between semesters – in first semester they had only 20 marks to distribute but in second semester they had a virtual \$100. The reasons for this change are discussed in Clark et al (2005). This allowed students to be more discerning, a large number of students only differentiate by a \$1 between team members. In 2004, 25 of the equal distributions by teams were done in first semester, and 17 in second semester when 4 of the 7 tests were conducted.

Secondly, students have been advised in course materials and in lectures not to make a pact, the following has been in the Project Manual (the unit bible) since 2004:

“It is tempting to have a pact with your team members to always give high ratings. You are advised *not* to do this. This encourages individuals not to do their share of the work and you will end up carrying them or submitting sub-quality work. You should respond based on *your* opinion of each person’s contribution. You should find that if you are honest with each other you will all learn more and improve, as students are often in a better position to provide one another with meaningful feedback regarding both technical and interpersonal performance.” (Herbert, Ollington 2006, pg 9).

When an entire team distributed amounts equally they were quizzed at a follow-up meeting with the lecturer as to whether this was really a true reflection of the work completed. In 2005 teams were advised at the follow-up meeting that this indicated that the team felt they all deserved the same mark for that work product but that they would all get the same lowest mark determined by the lecturer based on the Individual Contribution Reports

and timesheets. The impact of this is indicated by the number of teams that only reward evenly once. In 2006 from the very first follow-up meeting with the lecturer teams were given this warning when only a single member of the team distributed amounts equally. This appears to have had a dramatic effect in 2006. In 2004 there were five teams that rewarded evenly once in first semester only, and four other teams rewarded evenly 4 or less times with only one of these teams doing it again once in second semester. In 2005 there were seven teams that all rewarded evenly 4 or less times, with five teams doing it once in first semester only. In 2006 there were four different teams that all rewarded evenly once – two on the first pay packet and one each on the second and third pay packet.

Thirdly, students receive two formative marks during first semester that indicate the influence of the pay packets. The impact of this is also indicated by the number of teams that only reward evenly once. Also at the end of first semester students receive a grade and are thus made aware (in some cases painfully) of the impact of carrying other team members on their own overall results. The impact of this is indicated by the number of teams that reward evenly consistently in first semester but rarely in second semester (4 teams in 2004, and 2 in 2005).

Finally, in first semester students have to allocate work equally during all phases of the lifecycle (particularly design and implementation), but in second semester they are advised to allocate work based on students particular strengths while ensuring everyone does the same overall amount of work. If teams do this you would expect the pay packets to be less equal in second semester.

### 3 Are students honest about themselves?

A major concern with having students quantify contributions is that they will exaggerate their own contribution. In 2004 students gave themselves the

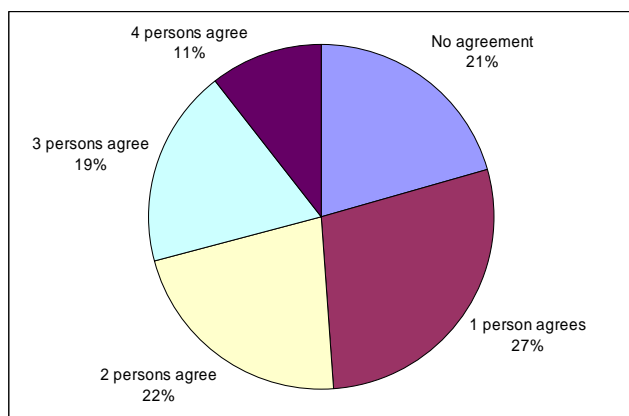
highest amount 191 times. In 134 cases they gave the same amount to at least one other member of their team, but not the entire team. From table 4 you can see that in both 2004 and 2005 students gave themselves the highest amount around 21% of the time and in 2006 this has increased to 26%. International students gave themselves the highest amount less often than the other categories, but the earlier section demonstrated that they are much more likely to give equal amounts to everyone.

Do students give themselves the highest amount when they shouldn't? In 2004 of the 191 cases where a student gave themselves the highest amount other team members gave them their highest amount 142 times, 74%. In 2005 203 of the 257 cases had agreement, 79%. In 2006, 77 of the 91 cases had agreement, 84%. Graph 1 shows that 79% of the time team members agreed that a member should have had the highest amount. It also shows that over 52% of cases had the agreement of at least two other team members (meaning more than half the team agreed on who should get the highest amount). Graph 2 shows how often the people who agreed gave the student the same, more or less money than they gave themselves. 21% of the time they weren't willing to give them as much money as students gave themselves. This analysis indicates that giving yourself the highest amount is justified in the majority of cases, but there are some students giving themselves the highest amount when the rest of the team does not agree.

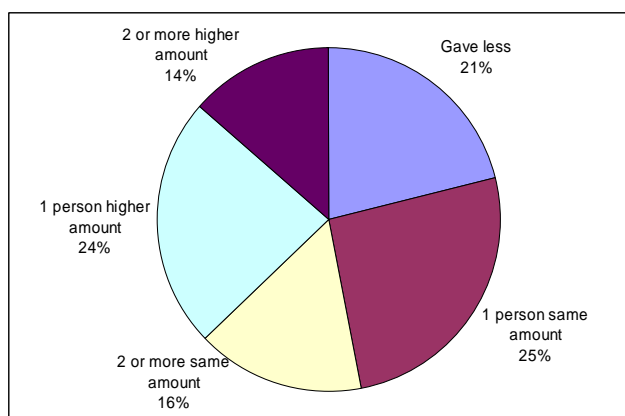
Do students give themselves the highest amount when they should? Over the three years, when at least two members agreed that another member should be given the highest amount in only 59% of these cases did the member agree. When at least three people agreed that another member should get the highest amount the member agreed only 70% of the time. So it seems some individuals do not give themselves the highest amount when they should.

		<i>Possible</i>	<i>Highest to themselves</i>	<i>Others equal high</i>	<i>% of possible</i>	<i>Lowest to themselves</i>	<i>Others equal low</i>	<i>% of possible</i>
2004	Females	91	2	23	27	9	7	18
	Males	812	55	111	20	29	39	8
	Domestics	749	51	117	22	29	42	9
	Internationals	154	6	17	15	9	4	8
	Total	903	57	134	21	38	46	9
2005	Females	170	11	20	18	4	28	19
	Males	1010	92	134	22	65	123	19
	Domestics	780	74	127	26	61	93	18
	Internationals	400	29	27	14	8	58	17
	Total	1180	103	154	22	69	151	19
2006	Females	72	0	15	21	10	15	35
	Males	273	33	43	28	23	33	21
	Domestics	207	29	36	31	24	28	25
	Internationals	138	4	22	19	9	20	21
	Total	345	33	58	26	33	48	23

**Table 4: Individuals that gave themselves their highest/lowest amount (but NOT equal amount to all)**



**Graph 1: How many agreed a person should be given the highest amount**



**Graph 2: How often the agreeer gave equal, more or less money than student gave themselves**

As shown on the right side of table 4, in 2004 students gave themselves the lowest amount 84 times. In 46 cases they shared an equally low amount with at least one other member of their team, but not the entire team. From 2004 to 2005 there was a significant increase in the number of students who gave themselves the lowest amount. This change possibly occurred as a result in the change of amounts being distributed in 2004 (20 marks in first semester versus \$100 in second semester) which meant in 2004 many more students distributed equal amounts. Strangely there was no change in the number of students who gave themselves the highest amount.

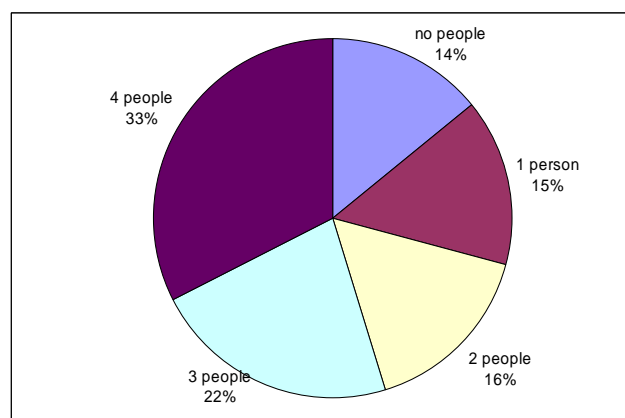
In 2004 and 2006 female students were more likely to give themselves the lowest amount, but this was not indicated in 2005. In 2005 there were two gender mixed teams where the majority of students were female (only one male). There has only been one other team where females have outnumbered males and the ratio was 3:2.

Do students give themselves the lowest amount too often? In 2004 of the 84 cases where a student gave themselves the lowest amount other team members gave them their lowest amount 66 times, 78%. In 2005 175 of the 220 cases had agreement, 80%. In 2006, 60 of the 81 cases had agreement, 77%. This indicates that giving yourself the lowest amount is justified in the majority of cases, but there are some students being too hard on themselves.

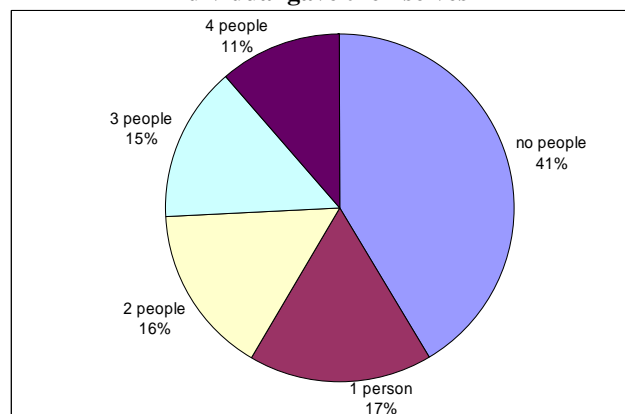
Do students give themselves the lowest amount often enough? Over the three years, when at least two members agreed that another member should be given the lowest amount in only 44% of these cases did the member agree. When at least three people agreed that another member should get the lowest amount the member agreed only 50% of the time. It seems many individuals do not give themselves the lowest amount when they should.

It is worth noting that over 71% of the time at least 2 people gave the same or more than the person gave themselves as shown in Graph 3, which is at least half the rest of the team. Again it is encouraging that 58% of the time at most 1 person in a team thinks a person should get less than they gave themselves, as shown in graph 4. These two graphs are indicating that in the majority of cases an individual is giving themselves an amount that the majority of the team agrees with.

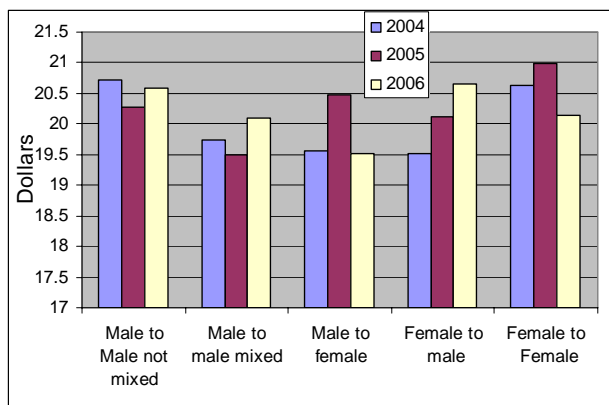
This evidence suggests that student quantification of contribution should not be used in isolation and that it is necessary to have other forms of evaluating the contribution of individuals to confirm the quantitative opinions of the students. Two other methods utilized in Software Engineering Project are timesheets filled in by an individual and individual contribution reports written by a team member about their contribution to a work product and agreed with by other team members, these are further described in Clark et al (2005).



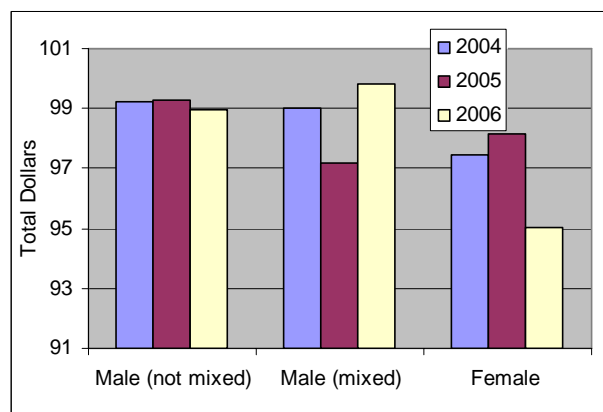
**Graph 3: How many people gave same or more than individual gave themselves**



**Graph 4: How many people gave less than individual gave themselves.**



Graph 5: Average individual amounts



Graph 6: Average total amounts

#### 4 Are female students treated fairly?

Graph 5 shows the average amounts given to team members based on gender. Graph 5 indicates that males gave fellow males in a mixed gender team lower amounts than males do to fellow males that are in all male teams. Males in 2005 gave higher amounts to females than males, in 2004 and 2006 they gave slightly lower amounts to females. Male students gave female students lower amounts than female students gave to fellow female students. In 2004 and 2005 females gave higher amounts to females than males. In 2006 females gave higher amounts to males than they gave to females. In 2005 and 2006 females gave higher amounts to male students than male students gave each other. So in conclusion, the individual amounts are not showing any consistent gender bias.

Table 5 is showing the significance of differences in average individual allocations by genders for 2005 and 2006. All levels of significance are determined using a Mann-Whitney nonparametric test for significance between the distributions of two independent samples,

with statistical significance defined by  $p < 0.05$ . Table 5 is showing that all the differences in 2005 were not significant but that the difference between what males gave to females and what males gave males was significant in the first semester of 2006.

Graph 6 shows the average total amounts (the accumulation of all the WPPP amounts) and indicates that in 2004 and 2006 female students averaged less than their male students, but in 2005 females were averaging slightly higher than males in mixed teams.

Table 6 is showing the number of times the maximum and minimum total amounts were given to different groupings of students in the mixed teams. In 2005 and 2006 female students were given the minimum total amount more often than their male team members. In 2004 and 2006 female students were given the maximum total amount less often than their male team members.

Since the distribution patterns have not been consistent over the three years it is not possible to conclude that there is a gender bias.

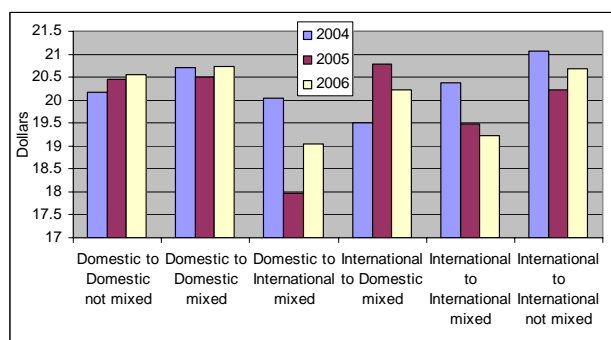
		2005			Semester 1, 2006		
		N	Average amounts	p	N	Average amounts	p
1	Male to Female	377	20.48	0.1515	219	19.53	<.0001
	Male to Male (mix)	932	19.5		534	20.10	
2	Male to Female	377	20.48	0.0735	219	19.53	0.0526
	Female to Female	160	20.98		126	20.15	
3	Female to Male	375	20.13	0.4483	219	20.66	0.0436
	Female to Female	160	20.98		126	20.15	
4	Female to Male	375	20.13	0.2483	219	20.66	0.4207
	Male to Male (mix)	932	19.5		534	20.10	

Table 5: Significance of differences in average individual allocations for 2005 and 2006

		N	Possible	Minimum	% of poss	Maximum	% of poss
2004	Females	13	91	33	36	30	33
	Males in mixed	41	287	117	41	123	43
2005	Females	17	170	89	52	84	49
	Males in mixed	30	300	138	46	125	42
2006	Females	24	72	30	42	19	26
	Males in mixed	52	156	38	24	54	35

Table 6: Minimum and Maximum total amount distribution

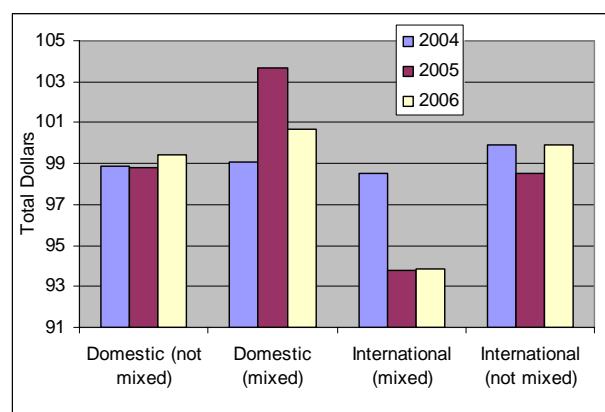




Graph 7: Average individual amounts

## 5 Are international students unfairly discriminated against?

Graph 7 shows the average amounts given to team members based on domicile. Table 7 is showing the significance of differences in average individual allocations by domicile for 2005 and 2006. Graph 7 indicates that domestic students in both mixed and non-mixed teams average around the same amount for each other in all years, with domestics in mixed teams being fractionally higher. Domestic students gave international students lower amounts than they gave fellow domestic students and this difference was significant (Table 7, row 1). Domestic students gave international students lower amounts than international students gave each other in all years and the difference was significant (Table 7, row 3). International students gave domestic students significantly more than domestic students gave domestics students in 2005 (Table 7, row 2), interestingly international students gave domestic students less than domestics students gave each other in 2004 and 2006! Internationals students gave lower amounts to fellow international students rather than fellow domestic students in 2005 and 2006, the reverse happened in 2004,



Graph 8: Average total amounts

the difference in 2005 and 2006 was significant (Table 7, row 4). Internationals in mixed teams gave significantly lower amounts to each other than internationals in non-mixed teams (Table 7, row 5), though all-international teams distribute amounts equally often, whereas no (or nearly no) mixed teams did.

Graph 8 indicates that domestics in mixed teams regularly received average total amounts higher than domestics in non-mixed teams. Internationals in mixed teams consistently got lower amounts than internationals in non-mixed teams. Domestics in mixed teams got higher amounts than internationals in mixed teams in all years. There was no indication of a problem in 2004, but in 2004 the amount of equal distribution was the highest. 2005 data indicated a potential problem that is also being seen so far in 2006. In 2005 and 2006 the number of international students increased significantly on 2004.

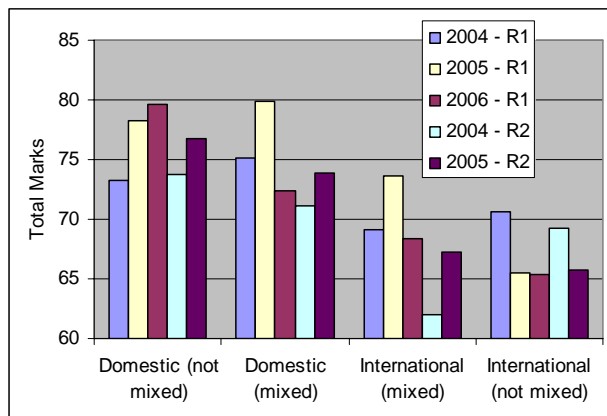
Table 8 indicates that in all years internationals in mixed teams were twice as likely to be given the minimum total amount. Likewise in all years the international students were less likely to be given the maximum amount.

		2005			Semester 1, 2006		
		N	Average amounts	p	N	Average amounts	p
1	Domestic to Domestic (mix)	220	20.51	<.0001	186	20.74	<.0001
	Domestic to International	290	17.98		132	19.05	
2	International to Domestic	290	20.78	<b>0.0143</b>	132	20.22	0.2676
	Domestic to Domestic (mix)	220	20.51		186	20.74	
3	International to International (mix)	320	19.47	<b>0.002</b>	246	19.21	<b>0.0375</b>
	Domestic to International	290	17.98		132	19.05	
4	International to International (mix)	320	19.47	< <b>0.0001</b>	246	19.21	<b>0.0001</b>
	International to Domestic	290	20.78		132	20.22	
5	International to International (mix)	320	19.47	< <b>0.0001</b>	246	19.21	<b>0.0031</b>
	International to International (not mixed)	920	20.22		300	20.69	

Table 7: Significance of differences in average individual allocations for 2005 and 2006

		N	Possible	Minimum	% of poss	Maximum	% of poss
2004	Domestics in mixed	11	77	13	17	29	38
	Internationals in mixed	8	56	19	34	13	23
2005	Domestics in mixed	13	130	25	19	39	30
	Internationals in mixed	16	160	65	41	35	22
2006	Domestics in mixed	22	66	12	18	27	41
	Internationals in mixed	26	78	28	36	11	14

Table 8: Minimum and Maximum total amount distribution



**Graph 9: Average Final Marks**

So all this analysis indicates that there maybe an issue of domicile bias, but maybe the differences in allocations are justified. An analysis of the final marks is necessary to determine if there is unfair discrimination.

The entire software development lifecycle is repeated each semester. In the first semester teams complete release one (or a third of the project), in second semester they complete release two (the remaining two-thirds). Graph 9 shows the average final marks for release 1 in all years and for release 2 in 2004 and 2005.

In 2004 and 2005 domestic students in mixed or non-mixed domicile teams were achieving around the same results in each semester. In both years in first semester domestics in mixed teams got slightly higher results than domestics in non-mixed teams but this was reversed in both years in second semester. In 2006 domestics in non-mixed teams have got slightly higher marks than domestics in mixed teams.

Overall graph 9 indicates that international students whether in mixed or non-mixed teams are averaging around the same grade but less than domestic students. Though in second semester 2004 international students in mixed teams got significantly lower results. This result is partially explained by the fact that there were only 8

international students in mixed teams and 4 of them were in the one team and that team did not do well. In 2005 international students in mixed teams achieved significantly higher results in first semester to international students in non-mixed teams (8%), but in second semester the difference was only 1.5%, and table 9 shows this difference was not significant.

In all years international students in mixed teams consistently achieved lower average final marks that domestic students in mixed teams. Table 9 shows that the difference was significant in second semester 2005.

The final marks are calculated using the WPPPs and so they are not completely independent. The teams have to produce three design reports, two in first semester and one in second each worth 10% for the team. The teams developed a release of the software in each semester. The team marks for these work products are not influenced by peer assessment.

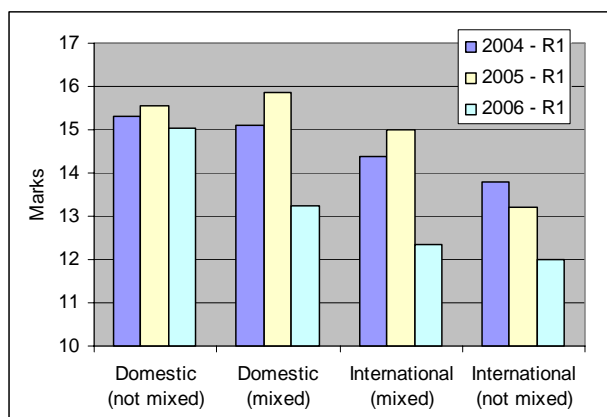
There has been no significant differences in marks for design reports for domestic student whether in a mixed or non-mixed team, in the majority of cases the differences have been less than 2% except for design report 2 in 2006 when the difference was 10%. An analysis does indicate though that internationals in mixed teams do achieve higher results than internationals in non-mixed teams for the design reports (over 20% for design report 1 in 2005 and 2006), and table 9 confirms that the difference was significant for design report 3 in 2005 where the difference was only 11%.

Graph 10 shows the team software results for release 1. Domestic students averaged around the same in both mixed and non-mixed teams though the difference in 2006 is larger than the other two years. International students in mixed teams averaged slightly higher than those in non-mixed teams. The difference between international students in mixed and non-mixed teams was significant in second semester 2005, as shown in table 9.

So the analysis indicates that there is no significant difference in the results of domestic students whether in a

		N	Average Mark	p
Domestic (mix) International (mix)	Semester 2 individual mark	13	73.85	<b>0.0314</b>
		16	67.31	
	Design report team mark (max 10)	13	9	0.102
		16	8.13	
International (not mixed) International (mix)	Software team mark (max 20)	13	15.77	0.0869
		16	15.13	
	Semester 2 individual mark	24	65.75	0.496
		16	67.31	
Domestic (not mixed) Domestic (mix)	Design report team mark	24	7	<b>0.0244</b>
		16	8.13	
	Software team mark	24	14.42	<b>0.0322</b>
		16	15.13	
Domestic (not mixed) Domestic (mix)	Semester 2 individual mark	65	76.75	0.1131
		13	73.85	
	Design report team mark	65	9.2	0.4325
		13	9	
	Software team mark	65	15.42	0.1814
		13	15.77	

**Table 9: Significance of differences in 2005 semester 2 marks**



**Graph 10: Team software results for release 1**

mixed or non-mixed team. This indicates that the results of the domestic students in mixed teams are not suffering and they gain many generic skills from the experience.

The analysis of the amounts given in WPPPs did indicate a significant difference in both average individual amounts and total average amounts given to international students in mixed teams compared to international students in non-mixed teams and domestic students in mixed teams, indicating that there is a possible bias against international students in mixed teams. But the fact that there is a significant difference in the marks given to internationals in mixed teams versus internationals in non-mixed teams for team items such as design reports and software, and no significant difference in the overall marks indicates the allocations are reflecting actual performance. It is important to remember that both domestic and international students in mixed teams were giving higher amounts to the domestic students.

## 6 Conclusion

This paper investigated distribution pattern concerns about quantitative peer assessment. Equal distribution is a concern, if it is done for the wrong reason. The analysis has shown that individual students will distribute amounts equally, particularly international students. In Software Engineering Project four factors were influential in reducing equal distribution such that it rarely happens on a team wide basis. Consideration needs to be given to the amount that is distributed, releasing grades that show the impact of equal distribution and explaining to students the interpretation of distributing equally. Lecturers are concerned that some students are not honest about their own contribution. The analysis has shown this to be true, students are likely to give themselves the highest amount both too often and not enough, likewise, students give themselves the lowest amount both too often and not enough. This evidence suggests that the tool should not be used in isolation and that it is necessary to have other forms of evaluating the contribution of individuals to confirm the quantitative opinions of the students. Gender bias is a real concern but there was no conclusive consistent evidence that there was a gender bias. There was concern that international students consistently received lower amounts in mixed teams (from both domestic students and fellow international students) but

an analysis of the results data concluded that there was no domicile bias and that peer assessments were reflecting actual (or at least perceived) performance.

## 7 References

- Brown, R. (1995): Autorating: Getting Individual Marks from Team Marks and Enhancing Teamwork. *Proceedings of Frontiers in Education Conference*.
- Clark, N. (2005): Evaluating student teams developing unique industry projects. *Proceedings of the seventh Australasian Conference on Computing Education*. Newcastle, Australia
- Clark, N., Davies, P., Skeers, R. (2005): Self and Peer Assessment in Software Engineering Project. *Proceedings of the seventh Australasian Conference on Computing Education*. Newcastle, Australia
- Gates, A.Q., Delgado, N., Mondragon, O. (2000): A Structured Approach for Managing a Practical Software Engineering Course. *ASEE/IEEE Frontiers in Education Conference*, October, Kansas City. 1:21-26
- Hayes, J.H., Lethbridge, T.C., Port, D. (2003): Evaluating Individual Contribution Toward Group Software Engineering Projects. *Proceedings of International Conference on Software Engineering*, Portland, Oregon. 622-627
- Herbert, N., Ollington, R. (2006), *Software Engineering Project: Project Manual version 9*, Accessed July 31, 2006, <http://www.comp.utas.edu.au/units/kxa351/2006ProjectManualv9.pdf>
- Humphrey, W. (1997): *Introduction to the Personal Software Process*. Addison-Wesley.
- Kaufmann, D.B., Felder, R.M., Fuller, H. (2000): Accounting for individual effort in cooperative learning teams. *Journal of Engineering Education*. **89** (2), 133-140.
- Layton, R.A., Ohland, M.W. (2001): Peer Ratings Revisited: Focus on Teamwork, Not Ability. *Proceedings of American Society for Engineering Education Annual Conference*, Albuquerque.
- Wilkins, D.E., Lawhead, P.B. (2000): Evaluating Individuals in Team Projects. *Proceedings of the 31<sup>st</sup> SIGCSE Technical Symposium on Computer Science Education*, Austin, Texas, 172-175



# Student timesheets can aid in curriculum coordination

Nicole Herbert, Zhong Wang

School of Computing,  
University of Tasmania  
Private Bag 100, Tasmania, 7001, Australia

[Nicole.Herbert@utas.edu.au](mailto:Nicole.Herbert@utas.edu.au), [zhongw@utas.edu.au](mailto:zhongw@utas.edu.au)

## Abstract

For many years lecturers have been encouraging students to complete timesheets to help them manage their time and prepare them for the process when they enter the workforce. As well as aiding the students in time management, the data contained in a timesheet can be used for curriculum planning. In 2004, 2005, and 2006 students used a web-based timesheet system during a capstone project course. After considering the accuracy of the timesheet data the focus of this paper is an analysis of the data from the timesheet system to identify student's behavioural patterns and concludes that students work to deadlines, that students do not spend too much time on a capstone project, that the time spent on a task relates to the marks allocated to that task, that more time available for a task does not mean more time is used and that students can be induced to do tasks early.

**Keywords:** PSP, Timesheets, Capstone Projects

## 1 Introduction

Humphrey (1997) defined the Personal Software Process (PSP) to help software engineers manage their time and to estimate the duration of future tasks based on historical data collected. PSP is also a useful tool for students and lecturers to quantitatively evaluate the effort associated with a given learning experience (von Konsky, Ivins, Robey 2005).

Software Engineering Project (SEP) is a capstone program provided by the School of Computing at the University of Tasmania. Students work in teams of 4 or 5 students and collaborate with a real industry client to produce a major piece of software. SEP is a 26-week program offered on two campuses divided into two consecutive 13-week units; the students get two grades. Each semester the students produce a release of the software.

During the first semester, each team has to submit two design reports: design report 1 contains the analysis documents for the entire project; and design report 2 contains the design documents for release 1 (e.g. UML

diagrams and Prototyping). In the second half of the first semester, each team implements release 1 which is about a third of the final software product. At the end of the semester, students give a formal presentation to demonstrate their software to the client and staff.

At the beginning of the second semester, each team has to complete another design report containing the changed analysis documents and the new design documents for release 2. The major outcome for second semester is the implementation of release 2 which is around two thirds of the final product. The students also write a user manual and a reference manual. At the end of semester, students demonstrate their software at a public demonstration day.

SEP utilises the ideas of time management from the Personal Software Process defined by Humphrey (1997). Students enrolled in this unit are required to submit their weekly timesheets in order to demonstrate and improve their use of time. Logging actual effort helps students to understand how they are spending their time, where their time is wasted and how it can be used more efficiently (von Konsky et al 2005). The timesheet system uses a web-to-database model that was developed in 2004, prior to that it was paper-based. The evolution of the timesheet system is described in Clark, Davies and Skeers (2005).

For each time entry students have to enter the date they did the job, the start time and the finish time of the job, and the job code (see Figure 1). Students also record the total duration of any interruptions and a detailed description of the job. Students can make entries for a week at any time during the week and students can modify or delete the current week entries. Students are permitted to add timesheet entries for previous weeks, but they can not edit or delete entries from a previous week, as timesheet entries are used for assessment. The job codes have varied slightly over the three years. The 2006 job codes were design, implementation, prototyping, manuals, meeting, marketing, testing, study, and admin.

Since 1998 the students have been required to complete weekly timesheets. Similar to Carrington (1998) student reaction to paper timesheets was mostly negative, they saw them as pointless and many openly stated their records were not accurate, but participated because it was a requirement. In a survey conducted in 2002 only 55% of the students thought the timesheets were useful.

2004 saw the introduction of an online version of the timesheets. The online system improved the accuracy as peer pressure had an impact. In the program feedback survey conducted during 2004, 68% of students indicated that their time recordings were 90% accurate, and a further 27% thought they were 70% accurate. Over 90%

---

Copyright © 2007, Australian Computer Society, Inc. This paper appeared at the *Ninth Australasian Computing Education Conference (ACE2007)*, Ballarat, Victoria, Australia, January 2007. Conferences in Research in Practice in Information Technology, Vol. 66. Samuel Mann and Simon Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.



Date	Start (hh:mm)	Finish (hh:mm)	Interruptions (mins)	Job code	Comments
Mon 07 Aug	09:15	11:10		Meeting	Team Meeting
Mon 07 Aug	14:30	15:30		Design	Wrote RTM
Tue 08 Aug	12:00	15:00	35	Design	UML Diagrams
Wed 09 Aug	09:15	09:45		Admin	Worked on scheduler
Wed 09 Aug	15:40	18:50	65	Prototyping	Author Info screen

**Figure 1 Timesheet System Screenshot**

of students found it useful to see how much time they were spending on their project and over 85% of students found it useful to see where their team members were spending their time. This represented a marked change in attitude to the value of monitoring time.

The timesheets are worth 5% of the final grade each semester. Each weekly timesheet is worth 0.5% of the final grade. To get the 0.5% they must enter a minimum number of entries (5 in 2005) for that week and they must average over 8 hours per week over the entire semester or for each week they average less than 8 hours they get 0. The students do 13 timesheets each semester and the 3 with the lowest score are not counted – allowing students to have weeks where they can do minimal project work. Each timesheet is due Monday midnight of the following week. For example, week 1 timesheet entries are due on midnight Monday of week 2. Any entries added after that time are not counted towards the number of entries for that week but are counted towards the average time.

The data in the timesheets is also used to do assessment of their work products. For example, the amount of time an individual spends on design is used in the calculation of their individual mark for a design report. Late entries count towards their assessment for a work product (up to the assessment date). Clark (2005) describes how timesheet data is used in the assessment process.

The timesheet entry data, which was collected from 2004, 2005 and semester 1, 2006, was first analysed for accuracy and then analysed to identify student's behavioural patterns and provide answers to the following research questions:

- Do students work to deadlines?
- Do students spend too much time on a capstone project?
- Does time spent on a task relate to the marks allocated to that task?
- Does more time available for a task mean more time is used?
- Can we induce students to do tasks early?

## 2 How accurate are the timesheets?

For timesheets to produce meaningful data they need to be accurate. Johnson and Disney (1998) found that using PSP in both industrial and academic settings revealed problems both in collection of data and its later analysis.

In SEP missing entries were believed to be common. As shown in Table 1 the average number of timesheet entries per student per week reduced slightly from 2004 to 2005 but rose in 2006. In 2006 the lecturer really emphasised the importance of timesheets and told students, 'if it is not on your timesheet you didn't do it', and explained the negative impact this would have on their assessment (equivalent of working in the real world for no pay).

To analyse the missing entries problem further students were divided into different groups according to their average weekly timesheet entries, shown in Table 2. In 2004, the first year of use of the electronic system, the lecturer had not established a minimum number of entries that they should enter each week for assessment purposes. It is interesting that this year had the greatest percentage of students entering more than 10 entries a week. However, the lecturer told students in 2005 that they had to make at least 4 timesheet entries each week to gain the 0.5% for that week's timesheet. As a result, the percentage of students who made less than 6 entries per week increased from 11% to 21%. In particular, the proportion of students who just entered 4 to 5 timesheet entries on average each week increased from 2% in 2004 to 9% in 2005. In 2006 this negative impact has been combated by changing the minimum requirement to 5 entries each week and waging the "if its not on your timesheet" campaign. The proportion of students who entered the minimum requirement or below reduced from 9% in 2005 to 6% in 2006. The campaign also increased the number of timesheet entries made, reducing the number of missing entries and therefore increasing their accuracy. In 2004 and 2005 69% of students averaged more than 7 entries, but in 2006 this number has increased to 79%. This analysis indicates that the majority of students are recording enough data for the timesheets to be useful for further analysis.

	2004	2005	Semester 1, 2006
Number of teams	27	25	24
Number of students	129	118	115
Total hours	40860	35109	18192
Average hours per student per week	12.18	11.44	12.17
Number of entries	29498	25385	13338
Average entries per student per week	8.79	8.27	8.92

**Table 1 Summary of timesheet data**

	2004	2005	Semester 1, 2006
4-5 entries	2%	9%	1%
5-6 entries	9%	12%	5%
6-7 entries	20%	10%	15%
7-8 entries	16%	21%	21%
8-9 entries	13%	15%	25%
9-10 entries	11%	9%	12%
>10 entries	29%	24%	21%

**Table 2 Average weekly timesheet entries**

In SEP because team members can see other team member's entries there is very little problem with fictitious or overly exaggerated entries. In general students do so much work on the project that fictitious entries are not necessary, the bigger problem is missing entries. Team members are asked to provide feedback to the lecturer via peer assessment every 4 weeks about whether they believe team member's entries are accurate. Timesheets are also discussed at the team management meetings with the lecturer every 3 weeks, and any dubious timesheet entries are discussed and corrected. Timesheets have been used in the program for over 8 years and the lecturer has enough experience to know which timesheet entries are abnormal. Imbalances between the amount of work submitted and what is recorded in the personal contribution reports and the timesheets provide a mechanism for ensuring minimal fictitious or overly exaggerated entries. 'Rounding up' of time taken is very common as students tend to record things as starting and ending on the hour. To minimise the impact of this for assessment, ratios (percent of total time) or ranges (10-12 hours) are used rather than exact Figures such as 10 hours per week. There is no evidence to suggest that using timesheet entries for assessment or increasing the number of required entries has increased the number of fictitious or exaggerated entries and that the data collected is useful enough for further analysis.

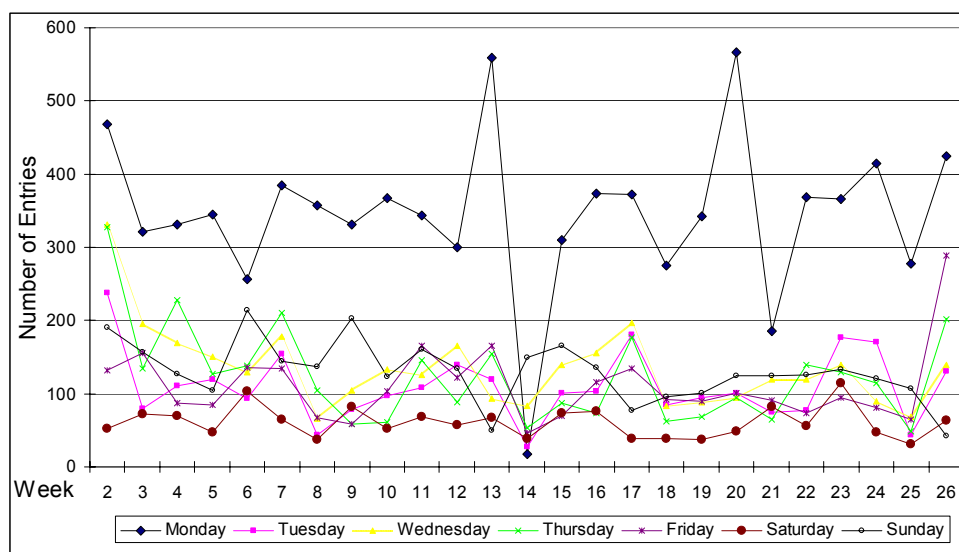
### 3 Do students work to deadlines?

It is strongly suspected that student's do the majority of their assignments when they are due, this is based on anecdotal evidence such as overflowing computer labs and the number of questioning students queuing at the door. Mathews, Haughton, Pisupati, Scaroni, DiBiase (2004) analysed how often their students accessed online data and found that most of the accesses were performed on the day of the deadline.

To confirm this for project students timesheet data was analysed for peak activity times. Figure 2 shows the number of timesheet entries entered on each week day for each week in 2005. The graphs for 2004 and semester 1, 2006 are very similar. In all years Monday was the day of the week when students entered the most number of timesheet entries, except for week 14. The timesheet entries were due on Monday midnight for the previous week. Week 14 was the first week of the second semester and students had just recommenced their project so Monday of week 14 was not a due date. In contrast, the day of week with the least number of timesheet entries was Saturday in all years.

Figure 3 shows the amalgamated data from all three years for the number of entries per entry day, the number of entries per start date (the day the work was done), and the number of entries that were entered within 24 hours of doing the work. There were almost three times as many entries made on Monday than the other days. On every day, except Monday and Sunday, the number of entries made was less than the amount of work started on that day. Only about 50% of entries were entered within 24 hours of doing the work, except on Sundays. This is because when students worked on Sunday, they had to submit it within 24 hours (or it would be a late entry and not count towards timesheet assessment).

To further analyse when students actually submitted their times, the due date was divided into eight 3-hour time slots and the timesheet entries proportion for each slot was analysed, shown in Figure 4.



**Figure 2 Number of timesheet entries on each day for each week in 2005**

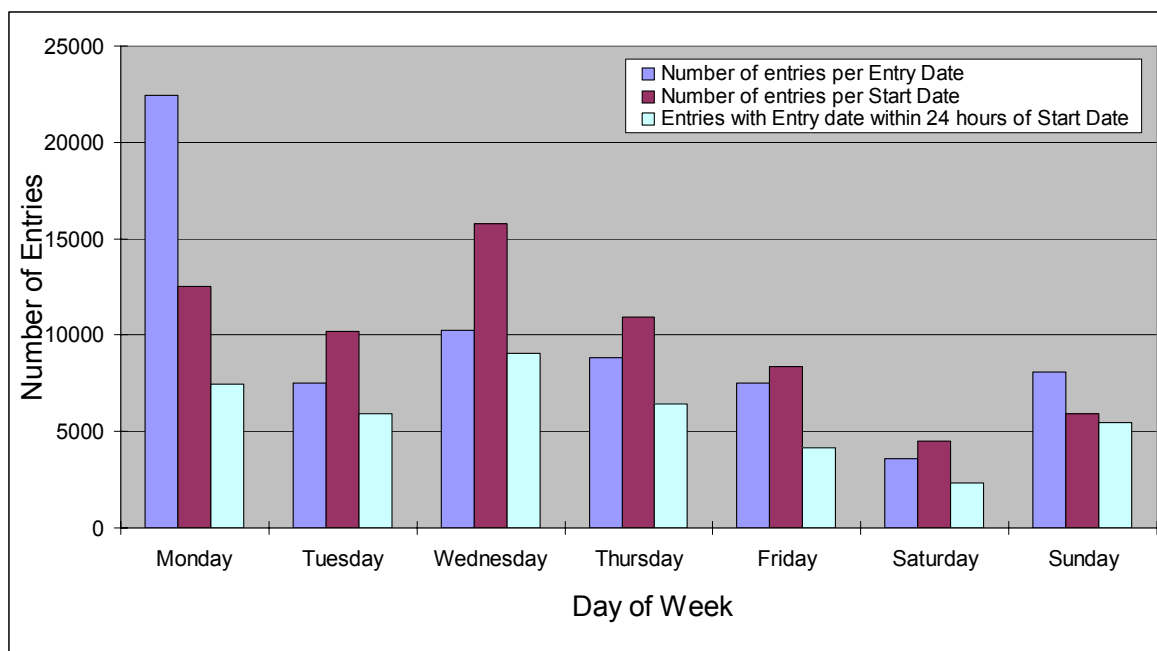


Figure 3 Timesheet entries on each week day

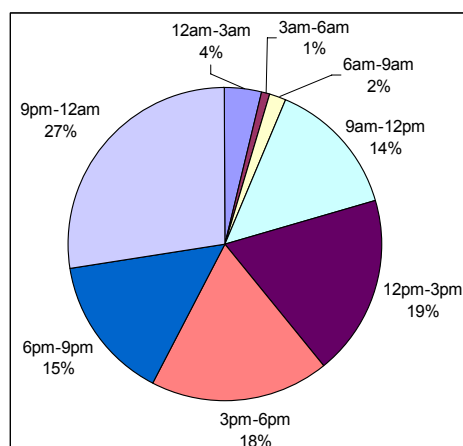


Figure 4 Proportion of timesheet entries for each 3 hour period on Monday

The biggest proportion of entries was made during 9pm to midnight. Indicating students submit their entries at the last minute, literally.

This evidence suggests that students are not recording their times as they do the work. Anecdotal evidence suggests that the students are recording their times on a “paper” timesheet judging by the fact that the blank paper timesheets that are available are continually disappearing and a ‘show of hands’ survey taken during a lecture. Obviously, further research needs to be conducted to confirm how many people are actually writing down their times at the time they do the work. To encourage students to do their entries earlier, next year we are considering introducing the rule that a timesheet entry only counts towards assessment of the timesheet if it was entered within 24 hours of doing the work – daily deadlines!

Figure 5 shows the amount of time that was spent on the project on a week by week basis, and there are some obvious peaks. Week 7 is obviously a very busy week in all years (this is the week design report 2 is due). Other key weeks are weeks 17/18 (design report 3), week 23 in 2005 (demonstration day was in week 24), week 24 in 2004 (demonstration day was in week 25) and week 26 in both years (the software is assessed and manuals are due).

All this evidence confirms what lecturers have known anecdotally for a long time – students do the majority of work for a task close to the deadline.

#### 4 Do students spend too much time on a capstone project?

It is extremely common that when students are doing a capstone project that they focus all their attention on the project to the detriment of their other subjects. Lecturers of capstone projects can suffer the ire of their colleagues who say their subjects are suffering as a result of students putting most of their time into the capstone project. When students use PSP, lecturers can evaluate just how much time the students are actually using (von Konsky et al 2005). As shown in Table 1, students averaged 12.18 hours on the project per week in 2004, compared to 11.44 hours in 2005. There were several possible reasons for this reduction. Firstly, the lecturer warned the students during lectures and meetings to reduce the time they spent on the project, particularly students who were averaging more than 15 hours a week. Secondly, there were changes made to the assessment procedures requiring less work to be completed, see Clark (2005) for more details. In 2006, the average was 12.17 hours per student. One reason for this increase was that the average weekly entries increased above the 2004 level, as discussed in the previous section.



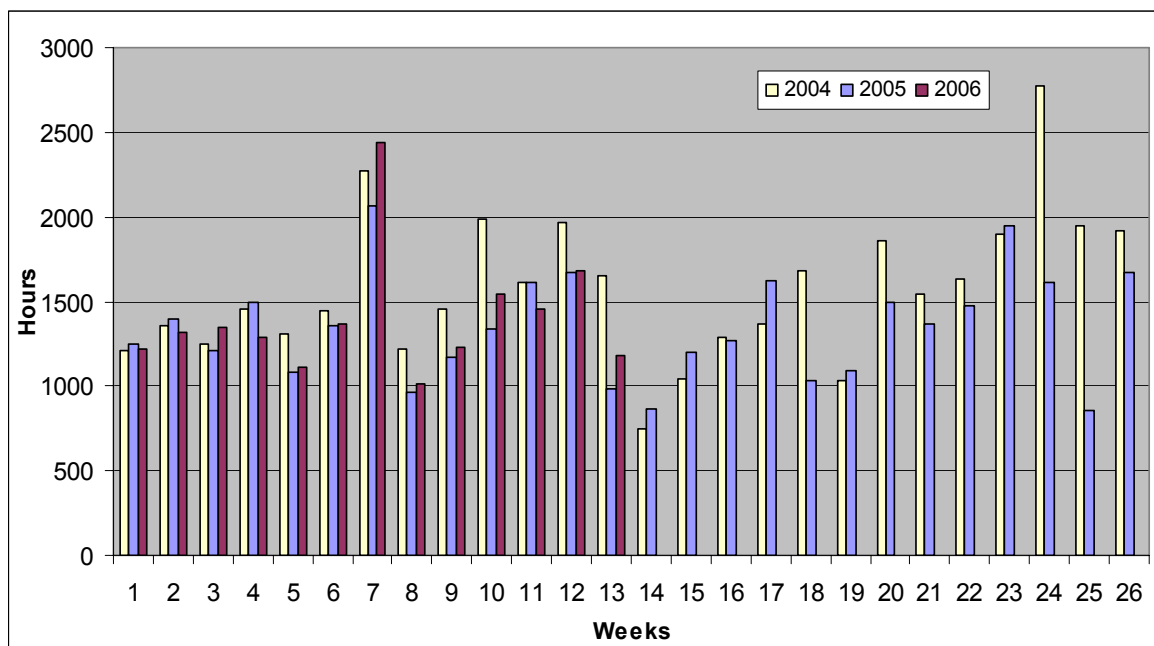


Figure 5 Summary of hours per week

Students are told that to pass they must spend at least 8 hours a week on average on the project. The University advises students to spend 10 hours a week on each unit. The analysis indicates that over the three years students are averaging around 12 hours a week on project, leaving plenty of time for their other units or commitments. Similar to Carrington's (1998) results some of the overload reported by students can be attributed to poor time management skills.

The day that most work was undertaken was Wednesday with Monday being the second busiest day, as shown in Figure 6. The reason for more work being completed on Mondays and Wednesdays was that these are the "project days" for each campus. The computer labs are reserved in the morning and the lectures are occasionally scheduled in the afternoons. When there is no lecture students are encouraged to work on their project. Classes for other final year School of Computing units are not scheduled on these days. This indicates that students are using the time set aside to do the project work. Other weekdays do

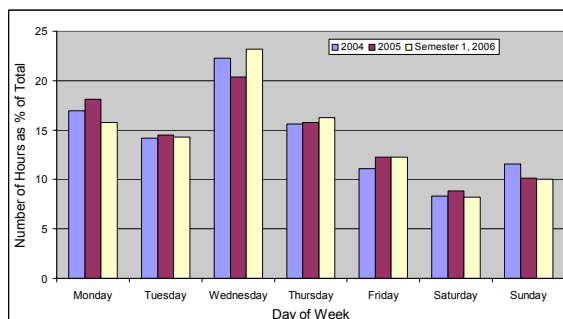


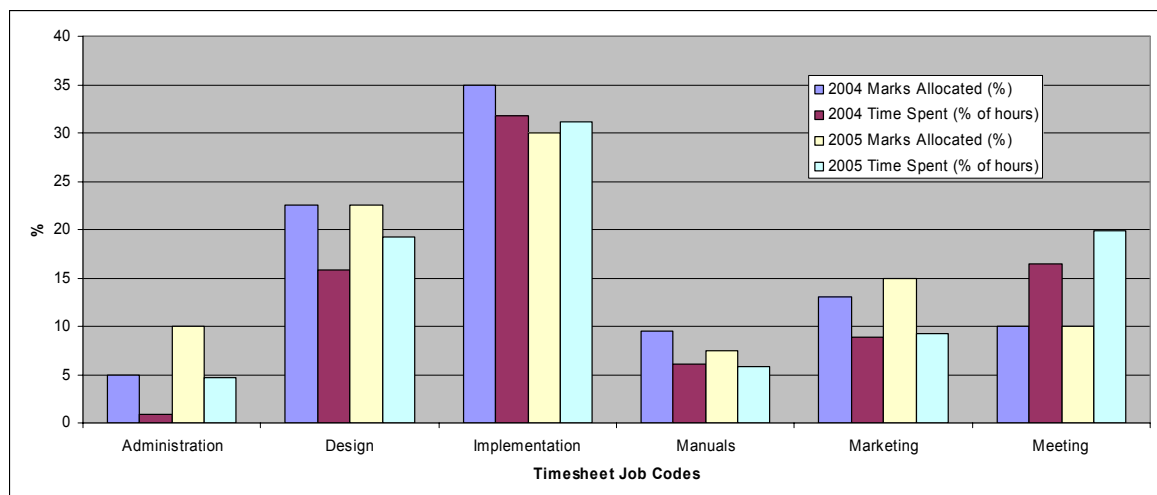
Figure 6 Summary of hours on each week day

contain a substantial amount of project work but in contrast, the amount of work undertaken on Saturdays was the least in all years. This is possibly a result of the change in student lifestyle. Many students are now working as well as studying, making it difficult for a team to get together on the weekend.

As already seen from Figure 5 students work to deadlines resulting in key weeks where there are peaks of project activity. It is likely that these weeks coincide with the submission week for assignments in other subjects, particularly week 7 as this is the middle of a semester. Complaints from colleagues and students are probably based on student availability in a given week. Based on this data it is obvious that if lecturers cooperated and organised it so other units had their submissions due in the week after a key project week it is likely that there would be less complaints.

## 5 Does time spent on a task relate to the marks allocated for that task?

Most units involve assessment, for many it is a combination of an exam and some assignments. How much each assignment should be weighted is a difficult decision. Generally the decision is made based on how long it will take for the student to do one assignment in comparison to another or how much time is available for one assignment versus another. Von Konsky et al (2005) conducted a study using PSP data to quantitatively evaluate effort against learning outcomes and adjusted the curriculum accordingly. The PSP data was analysed to see if there was a correlation between how much time students spent on each of the assessed components and the marks allocated for that component.



**Figure 7 Time spent versus marks allocated in 2004**

Figure 7 shows the percentage of the marks allocated for a component versus the percentage of time students spent on that component during 2004 and 2005. For the purposes of this analysis the time spent and marks allocated have been combined for the two semesters in a year, but the assessment is performed on a semester basis. Clark (2005) describes the entire assessment process. Design, implementation and meetings were the most time-consuming jobs in all years.

Some job codes have a direct assessment component (e.g. design and implementation) but some others do not. Meetings are not assessed, but individuals get marks for professionalism at team meetings (assessed by peers) and professionalism at client meetings (assessed by client). Administration is also not assessed, but timesheets (in both years) and using the task scheduler and completing a diary (in 2005) all formed part of the administration job code.

The time spent was very similar to the marks allocated. In the majority of cases the differences did not exceed 7%. In particular, the difference for implementation was just 3% in 2004 and 1% in 2005. This is interesting because student often comment that the software should be worth more of the final grade!

One possible reason for any discrepancy is the job code “study” (researching project management process) which was not allocated marks, occupied 9-10% of the time consumption. Much of this time would have been spent on study that resulted in improved design and implementation marks. This is confirmed when each semester is looked at in isolation as the amount of time spent on study in semester 2 has decreased by 50% on semester 1 each year and the design and the implementation times are much closer to the marks allocated. The reason for the sharp decrease was that the number of lectures was halved in semester 2 and that students are more familiar with the project management process so they spend less time reading project management materials.

Another reason for any discrepancy is meeting time spent was more than what was allocated by marks for professionalism but many meetings were held to deal with design and implementation issues as stated in the comment field of the time entry.

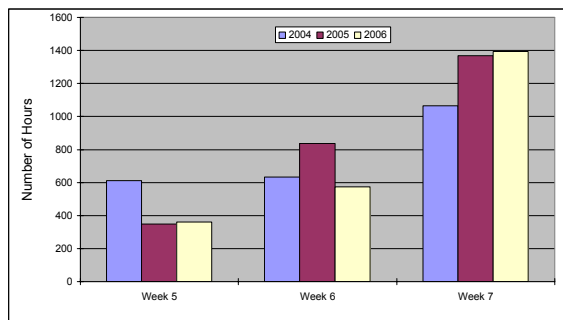
So having the PSP data is an excellent way of confirming that the distribution of marks is closely related to the time spent on an assessed task, rather than using the less meaningful time available statistic. PSP data is also useful for making educated adjustments. For example, based on the accumulated data, the weighting for many components does not need to change but the weighting for marketing and administration should be reduced or these tasks should be made to involve more work.

The correlation between marks and time spent could also explain why students spend so much time on project tasks in comparison to other assignments as they are generally worth more to a final grade. In the School of Computing the average assignment weighting for an assignment for final year students is 12.5%, ranging from 5-25%. Based on this analysis students are going to devote more time to a 30% implementation mark rather than a 5% assignment.

## 6 Does more time available for a task mean more time is used?

When lecturers set an assignment it is always an issue deciding how much time to allow. We are faced with the dilemma that it takes time to come to terms with the problem, to learn the required skills and then perform the task. Unfortunately, as already shown, the majority of students will leave the task until it is due or shortly before.

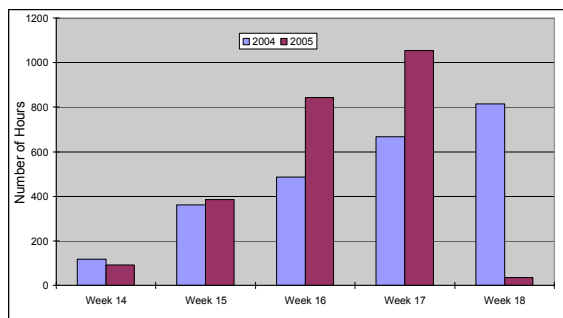
One interesting event during the design report 2 phase (weeks 5 to 7) is the impact of the Easter break – which means students have two weeks in real time but it is only counted as one project week. In 2004, week 7 (the week it was due) was the Easter break week and compared to week 5 and week 6, the design time almost doubled, as shown in Figure 8. In 2005 the Easter break week was week 5 (the first week in the design process), however, in



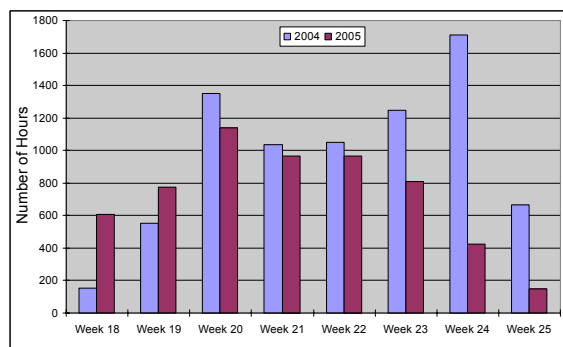
**Figure 8 Design time from week 5 to 7**

comparison to other years students spent the least time on design during this week. The Easter break week was back to week 7 in 2006, and again the amount of design work doubled in this week. From this one can only conclude that students do the most work in the week it is due and that the extra time does not result in more time used.

In semester 2, students start design report 3 in week 14. In 2004, students had 5 weeks but in 2005 they only had 4 weeks to finish the design report, as shown in Figure 9. In 2005 the average hours spent on design was 593 hours per week, much higher than the 489 average hours per week in 2004. In 2005 118 students spent a total of 2371 hours doing design (20 hours per student) and in 2004 when they had an extra week 129 students spent 2445 hours (19 hours per student). In both years the students did the most work in the week it was due, but more time meant that the load was more evenly spread over the allowed weeks, whereas the shorter time resulted in more dramatic increases each week.



**Figure 9 Design Time from week 14 to 18**



**Figure 10 Implementation time from week 18 to 25**

Release 2 was completed from week 18 to week 25, Figure 10 shows the implementation time for 2004 and 2005. At the end of semester there is a public demonstration of the final software for clients and members of the Tasmanian IT community. In 2004, the demonstration day was in week 25 and the peak of the implementation time was achieved in week 24. Another peak of the implementation time was reached in week 20 (the semester 2 mid-semester break, two weeks in real time). The time spent on implementation was more evenly spread in 2005 and the peak appeared in week 20 which was again the mid-semester break week, but this peak does not indicate that significantly more work was completed this week, as the other weeks are all very similar.

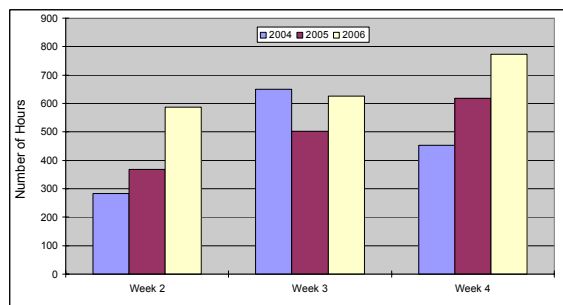
This analysis shows that making more time available does not mean that more time will be used, and that most work is completed near when the task is due. This suggests that when setting assignments it would be better to give each assignment a short available time and coordinate across subjects to ensure minimal overlap.

## 7 Can we induce students to do tasks early?

Most lecturers are concerned that students appear to leave their assignment work until it is due, this is one of the reasons that PSP is included. They would like to find a way to encourage students to spread the load over the allocated time, not only to increase their learning but to distribute the use of the available resources (e.g. computer labs). The data was analysed to see what events resulted in students doing tasks early.

Each project is tested at various times during the semester by people other than members of the development team. Each student enrolled is required to develop one project and test three other projects at different times during a semester. All teams are developing different projects, so there are no concerns about plagiarism. The testing process was assessed but the actual items made available for testing were not assessed by the lecturer until after the development team had fixed them based on feedback from the testers – see Clark (2004) for more details.

After students have a client meeting to acquire requirements in week 1, they write the analysis documents (design report 1). As shown in Figure 11, the first peaks of time spent on design in all three years were achieved around week 3 or week 4 because students had to submit their design report in week 4.



**Figure 11 Design time spent from Week 2 to 4**

In 2004, the peak of the design time in these three weeks was in week 3. One reason was students were required to distribute two of the documents from design report 1 to their testers by the end of week 3 to be tested early in week 4. In week 4, they modified these documents according to the feedback they received and completed two more documents. In 2005, students were not required to send the documents in week 3 and could have their testing meeting anytime in week 4. Thus, the design time rose linearly from week 2 to week 4 this year. The situation changed in 2006, students had to hold their testing session in week 3 but they only had to test one document. The time spent on design was much higher than the previous two years in week 2 and the design time spent in weeks 2 and 3 was almost the same.

Figure 12 shows the implementation time from week 8 to week 13 when the students were implementing release 1. There were two testing sessions held in week 11 and week 12 in 2004 and the peak of the time spent on implementation was reached in week 10. In 2005, only one testing session was held in week 12 and the peak of the time spent on implementation was moved to week 11. In 2006, students held one testing session either in week 11 or in week 12 and the peak was in week 10, but it was much more evenly spread from weeks 9 to 12.

This analysis indicates that students can be induced to do work prior to a deadline by introducing some pseudo earlier deadlines (even just getting them to distribute the work to fellow students).

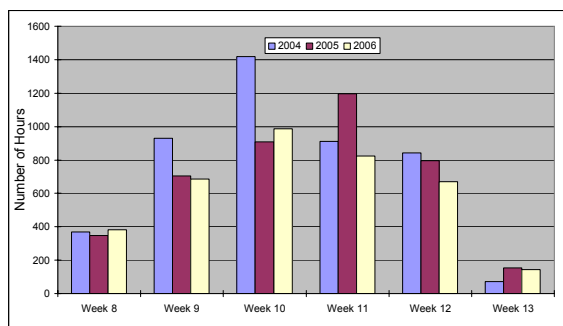


Figure 12 Implementation time from Week 8 to 13

## 8 Conclusion

The timesheet data was considered accurate enough to do some meaningful analysis and this analysis has identified common student behavioural patterns which most lecturers knew to be true based on anecdotal evidence: students do work close to deadlines, students will do the minimum requirements for assessment unless encouraged by the assessment system to do otherwise, students will focus their attention to areas that have the most marks associated with them.

Students do not spend too much time on a capstone project, but they do spend more time on projects than their other subjects, possibly because it has more deadlines or possibly because the assessed components are worth more than the assignments in other subjects.

The time spent on a task does relate to the marks allocated to that task. PSP data could be used to tweak an assessment weighting to ensure the marks are distributed where the students are spending the most effort (time) rather than using a weighting based on time available. To get students to put more effort into their assignments they should be weighted higher (and if necessary made to include more work to justify the weighting).

Giving the students more time to complete an assignment does not mean more time will be used to complete the assignment. The most work will be completed close to the deadline though students can be induced to do work early if there are marks associated with doing it early. When assignments are set they should be given minimal available time but coordination across subjects must ensure minimal overlap and deadlines should avoid key weeks in the project process.

In conclusion, collating PSP data over a three year period has provided some meaningful insights into student behaviour and these insights could be very useful for curriculum coordination.

## 9 References

- Carrington, D. (1998): Time Monitoring for Students. *Proceedings of Frontiers in Education Conference*, Tempe, Arizona, 8-13
- Clark, N. (2004): Peer Testing in Software Engineering Projects. *Proceedings of the sixth Australasian Conference on Computing Education*. Dunedin, New Zealand., 41-48
- Clark, N. (2005): Evaluating student teams developing unique industry projects. *Proceedings of the seventh Australasian Conference on Computing Education*. Newcastle, Australia, 21-30
- Clark, N., Davies, P., Skeers, R. (2005): Self and Peer Assessment in Software Engineering Project. *Proceedings of the seventh Australasian Conference on Computing Education*. Newcastle, Australia, 91-100
- Humphrey, W. (1997): *Introduction to the Personal Software Process*. Addison-Wesley.
- Johnson, P.M. and Disney, A.M. (1998): *Investigating data quality problems in the PSP*, Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering, Lake Buena Vista, Florida, 143 - 152
- Mathews, J.P. Haughton, N. Pisupati, S. Scaroni, A.W. DiBiase, D. (2004): For an online course encompassing "traditional campus students": how, where, and when students work and engage with the course material, *34<sup>th</sup> ASEE/IEEE Frontiers in Education Conference*, Savannah, GA, F1D-12-16
- Von Konsky, B.R., Ivins, J., Robey, M., . (2005): Using PSP to Evaluate Student Effort in Achieving Learning – Outcomes in a Software Engineering Assignment *Proceedings of the seventh Australasian Conference on Computing Education*. Newcastle, Australia, 193-202

# Difficulties experienced by students in maintaining object-oriented systems: an empirical study

**Amela Karahasanović**

Simula Research Laboratory  
P.O. Box 134, NO-1325 Lysaker, Norway  
amela@simula.no

**Richard C. Thomas**

Computer Science & Software Engineering,  
M002, The University of Western Australia,  
Crawley, Western Australia 6009, Australia  
richard@csse.uwa.edu.au

## Abstract

It is widely accepted that software maintenance absorbs a significant amount of the effort expended in software development. Proper training of both university students and professional developers is required in order to improve software maintenance. Understanding cognitive difficulties the students have while maintaining object-oriented systems is a prerequisite for improving their university education and preparing them for jobs in industry. The goal of the experiment reported in this paper is to explore the difficulties of students who maintain an unfamiliar object-oriented system. The subjects were 34 students in their third year of study in computer science. They used a professional Java tool to perform several maintenance tasks on a medium-size Java application system in a seven-hour long experiment. The major difficulties were related to understanding program logic, algorithms, finding change impacts, and inheritance of the functionality. Based on these results we suggest teaching the basics of impact analysis and introducing examples of modifying larger object-oriented programs in courses on object-oriented programming.

## 1 Introduction

Software maintenance has been widely recognised as a dominating cost factor in most software organisations. Although the reported figures differ, most researchers on software maintenance agree that more than 50% of programming effort is constituted by changes made to the system after the implementation (Coleman *et al.*, 1994; Holgeid *et al.*, 2000; Lehman and Belady, 1985; Lientz, 1983; Nosek and Prashant, 1990; Pfleeger, 1987; Zerkowitz, 1978).

Whereas difficulties the students have during the design of object-oriented systems are relatively well understood, rather less attention has been paid to their difficulties during maintenance. The study reported in this paper explores difficulties of programmers when conducting maintenance tasks.

The program used in this study was a 3600 lines of code (LOC) large library application system written in Java and can be considered to be a medium-size application according to the classification given by von Mayrhauser and Vans (1995). The participants were 34 students in their third year of study in computer science at the University of Western Australia (UWA). The experiment lasted seven hours and the participants conducted three maintenance tasks on the given application system, using JBuilder. The participants were provided with documentation that describes the application system and the JBuilder documentation. They had access to the Java online documentation.

## 1.1 Background

In spite of their complexity, maintenance tasks are often given to beginners and less experienced developers (Gunderman, 1988). To improve maintenance proper training of both university students and professional developers is required (Kajko-Mattsson *et al.*, 2002). Furthermore, it requires the provision of meaningful feedback to maintainers (Jørgensen and Sjøberg, 2002).

Object-oriented programming has become a *de facto* standard and therefore we need to understand the problems of maintaining such systems. Object-oriented programming is increasingly being taught in computer science courses. A survey conducted by Dale (2005a; 2005b) shows that 65% of the participating educational institutions teach object-oriented programming as a part of introductory courses in computer science education. Some of these students are going to have to maintain object-oriented systems when they start work.

It is therefore important to understand the cognitive difficulties the students have while maintaining object-oriented systems. It is a prerequisite for improving their university education and preparing them to enter the maintenance workforce.

Several studies have been conducted on the cognitive consequences of the object-oriented approach in the context of software design (Détienne, 1997). In her survey of empirical research on object-oriented design, Détienne (1997) gives a comprehensive list of difficulties experienced by individuals (novices and experts) and teams during the design of object-oriented systems. Novice designers have been found to have problems with class creation and with articulating the declarative and procedural aspects of the solution. They also had

misconceptions about some fundamental object-oriented concepts.

In a previous study Karahasanovic *et al.* (2006), explored the strategies and difficulties of programmers when conducting maintenance tasks. The participants were 38 students in their third or fourth year of study in computer science at either the University of Oslo or Oslo University College, and can be considered as advanced beginners according to the classification of Dreyfus and Dreyfus (1986). They conducted three maintenance tasks on a Java library application system, using JBuilder. The results showed that two major groups of difficulties were the comprehension of the application structure (identifying GUI components affected by a change, identifying classes affected by a change, identifying impacts of a change within a class) and using the inheritance of functionality. Furthermore the subjects had difficulties with the GUI, understanding and using a given Java API class, and algorithms.

The ability to generalize these results to the target population of advanced beginners, i.e., external validity of this study can be questioned. It is recommended that a way to identify potentially important factors that affect the process under investigation is to replicate the study with variations in the context variables (Basili *et al.*, 1999). Thus, our earlier findings need to be tested through replications with subjects from a slightly different educational background.

The present investigation is a replication of the study conducted with students in Norway by Karahasanovic *et al.* (2006) with a major difference that the subjects were from a university in another country and had a slightly different syllabus. It aims to identify the difficulties that students have while conducting maintenance tasks on a medium-size object-oriented application.

The remainder of this paper is organised as follows. Section 2 describes the methodology. Section 3 presents the results. Section 4 discusses the limitations of this research. Section 5 concludes and suggests avenues for further work.

## 2 Research Method

The main goals of the experiment were:

- To identify students' difficulties in conducting changes on a medium-size object-oriented application,
- To conduct an analysis of students' comprehension strategies on a medium-size object-oriented application, and
- To replicate and extend the earlier investigation by Thomas *et al.* (2005) into keystroke latency metrics as an indicator of programming performance.

This paper reports results regarding the first goal. The results regarding other goals are outside the scope of this paper.

The experimental material from the original experiment conducted by Karahasanovic *et al.* (2006) was translated from the Norwegian by the authors. This experiment had two treatments (Feedback Collection and Control Silent), whereas the original experiment had two more treatments (Concurrent Think-Aloud and Retrospective Think-Aloud) as it aimed to evaluate different think-aloud methods. A short copy typing test was introduced in this experiment to be used in the keystroke latency investigation. Otherwise, the experimental design and the material were the same in both experiments.

### 2.1 Experimental Design and Participants

A randomised design was employed: participants were randomly assigned into one of two treatment groups. There were 34 participants, half in each group.

All the participants were students at UWA. They were mainly in their third year of study in the School of Computer Science & Software Engineering and were asked to volunteer via an email sent to everyone taking a third or fourth level unit. The normal minimum attainment was to have passed two second level units: *Data Structures and Algorithms* and *Object Oriented Programming*. They will also have passed *Java Programming* from level one. About half the participants were on double degrees, such as Bachelor of Computer Science and Bachelor of Engineering; these have high cut off grades for entry. The remainder were taking single degrees, such as Bachelor of Computer Science. Everyone was male, aged 19-47, mean 22.

The protocols reported in this paper were approved by the university's Human Research Ethics Committee prior to the commencement of the recruitment of participants.

### 2.2 Treatment

The two treatments were Feedback Collection (FC) and Control Silent (CS). In Feedback Collection the participants were asked every 15 minutes "What are you thinking now?" This was delivered through the feedback collection screen that appeared for two minutes during the change tasks (Karahasanovic *et al.*, 2005). Participants could write whatever they wanted in that period and if they did not close the window it would automatically disappear after two minutes. In Control Silent, the tasks were the same but there was no feedback collection.

### 2.3 Procedure

Sessions were organised for 7 separate days, each testing 2-7 students, and held in the computer science laboratories. Sessions would start at 09:15 with an information meeting; continue through to lunch, provided around 12:30-13:00, and finish about 16:15. Participants were paid an honorarium of A\$100. Two observers were present in the laboratory during the experiments to answer questions and to provide help if any technical problems arose.

The first task was a short copy typing test, followed by a background questionnaire administered over the web using the Simula Experiment Support Environment (SESE) (Arisholm *et al.*, 2002). Next, everyone solved a simple training task and then a calibration task. Following this, those in the FC group were trained on providing written feedback, while CS members started on the change tasks. Everyone attempted three change tasks and then an exit questionnaire. Lastly there were group interviews.

## 2.4 Tasks

The subjects were asked to conduct a small training task and a pre-test task. The purpose of the training task was to make subjects familiar with SESE and the experimental situation. The participants downloaded the task, created a Java program to write a string in reverse order and uploaded their solutions. The pre-task was to extend the functionality of a bank teller machine program. This application was a small Java program consisting of seven Java classes and about 400 LOC. The task was to extend the program to provide a printout of all successfully performed transactions (deposits and withdrawals) for a given bank account. The purpose of the pre-test task was to provide a basis for comparing the programming skill level of the subjects. These tasks were taken from (Arisholm *et al.*, 2001).

The tasks of the experiment were to modify a library application system given in Eriksson and Penker, (1998). A library lends books and magazines. The books and the magazines are registered in the system. A library handles the purchase of new titles for the library. Popular titles are bought in multiple copies. Old books and magazines are removed when they are out of date or in a poor condition. The librarians can easily create, update, delete and browse information about the titles in the system. The borrowers can browse information about the titles. They can reserve a title if it is not available. The application consists of four packages with a total of 3600 LOC in 26 Java classes. This application system was used because we assumed that the application domain is very familiar to students. The subjects were asked to conduct the following changes on the library application system:

- Task1 Delete functionality related to ISBN number
- Task2 Extend the system to handle customer e-mail address
- Task3 Introduce the functionality to inform a person when a loan is due

The tasks were ordered by complexity. The subjects were provided with documentation describing the library application system in addition to the normal JBuilder documentation. They also had access to the Java online documentation. We emphasized that subjects should give higher priority to the quality of solutions rather than to shorter development time.

## 2.5 Data Collection and Supporting Tools

A Web-based tool, the Simula Experiment Support Environment (SESE) (Arisholm *et al.*, 2002) was used for logistics support. The subjects used this tool to answer the background questionnaire, to download the documents and code, to upload their solutions and to provide feedback (FCM group only). The tool recorded start-time and end-time for each task. The typing test was distributed by the observers. Keystrokes, mouse-clicks and window focus events were logged with timestamps in milliseconds by the GRUMPS-Lite software (Thomas *et al.*, 2003). The programming environment was Borland JBuilder, chosen as it was used in prior experiments at Simula. The pretest questionnaire confirmed that most students had little or no prior experience with JBuilder. Observers made notes during the experiment.

## 2.6 Data Preparation and Analysis

Cognitive difficulties the participants had while solving the given tasks were identified from the collected feedback and from their solutions.

### 2.6.1 Collected Feedback

Information collected by the feedback-collection tool was first categorised in four broad categories: experimental context, subjects' perception, experimental conduct and interaction. The feedback-collection data was then analysed and the information about the comprehension problems and background knowledge was used to make a list of difficulties for each participant. The coding took about 16 working hours.

### 2.6.2 Participants' Solutions

The assessment of the participants' solutions (correctness and problems) was done by a PhD student who was not involved in this research. She was provided with task specifications, correct solutions and guidelines for giving scores. All solutions were compiled, executed and thoroughly tested for functionality. The source code was also manually inspected. Questionable cases were resolved through discussion with the researchers. Based on this analysis we made a list of problems for each participant. The assessment of the solutions took about 80 working hours.

## 3 Results

The participants experienced different difficulties while conducting change tasks. We first give an overview of the difficulties identified as a result of examining the participants' solutions and collected feedback. We then describe the major difficulties in greater detail.

To identify the difficulties participants had, we first analysed the assessor's report and made a detailed list of



errors for each participant. We then extended this list with the difficulties that we found in the collected feedback.

As in the original experiment conducted by Karahasanovic *et al.* (2006), the difficulties were then categorised within a refinement of the model of von Mayrhauser and Vans (1995). Von Mayrhauser and Vans describe two types of knowledge: (i) general knowledge, which is independent of the specific software application that the programmers are trying to understand, and (ii) software-specific knowledge, which represents their level of understanding of the software application. They suggest that difficulties and errors in comprehension arise from a lack of either general, or specific knowledge, or both. Among the difficulties caused by the lack of general knowledge, we identified the following sub-categories: difficulties concerning program logic, graphical user interface (GUI), object-oriented programming, algorithms and programming environment. Among the difficulties caused by the lack of the specific knowledge, we identified the following three sub-categories: difficulties concerning the GUI, object-oriented programming and testing procedure.

Table 1 gives an overview of the difficulties. A difficulty is presented only once per participant per task. As described in Section 3, Task 3 was given as an extra change task and the majority of participants did not complete it. Consequently, the numbers of difficulties per category reported for Task 3 will not give us a complete picture. However, we present them here because they provide additional information about the difficulties that the participants experienced.

Two major general difficulties that prevented the participants from completing the task or caused a significant delay were related to program logic (23 occurrences) and algorithms (10 occurrences). The participants had two types of problems related to program logic (category 1.1). The first one was related to an if-then-else statement. This statement implements a book search on title, author or ISBN. The participants removed either too much or too little from this statement and, as a result, the library application did not work properly. It might be that the participants interpreted this task as a pure text editing task (finding all ISBN occurrences and deleting them) and did not try to understand the logic of the program. However, it is also possible that the participants felt time pressure and were not sufficiently careful. Another was related to finding a given title. Two participants explicitly reported that they had problems understanding the logic of this part of the application.

Another difficulty that was reported relatively frequently concerned knowledge of algorithms or the application domain (category 1.4). Ten participants failed to calculate the expiry date correctly, or did not try to calculate it at all. They also reported in the collected feedback that this is difficult. One usually uses library classes for different conversions and calculations. Therefore, it might be that making their own calculations was difficult for these students. Furthermore, the array index started from zero in this library class, which might be unusual for the

students, who mostly programmed in the programming language Java.

Two specific difficulties that frequently occurred were related to adding or removing GUI components (category 2.1.2, 17 occurrences) and removing label declarations (category 2.1.3, 19 occurrences). The participants either forgot to add or remove different components of the GUI-like radio buttons and text fields or failed to remove all label declarations of ISBN in Task 1. It was clearly stated in the task description that all references to ISBN should be removed. However, leaving some of these ISBN declarations had no effect on the functionality of the application.

The participants in the present study appeared to have particular difficulty with the GUI components (category 2.1.2). It should be noted that many of the present people would not have studied the design and implementation of GUIs by the time they participated in this experiment; this is covered towards the end of the degree. The given tasks required no special proficiency in GUI programming, but they required basic understanding of impact analysis. Familiarity with a domain (GUI in this case) could make impact analysis easier for the students.

Two major specific difficulties that prevented the participants from completing the task or caused a significant delay were related to finding impacts on classes (category 2.2.2, 14 occurrences) and inherited functionality (category 2.2.4, 12 occurrences). To conduct change tasks, the participants had to comprehend the structure of a medium-sized application object-oriented application. They had to comprehend relationships between the classes and to find the classes affected by a change. It seems that this was difficult for them.

Furthermore, the participants needed to understand inheritance of the functionality to solve the tasks. Classes in the library application that needed to be persistent had to inherit an abstract class called Persistent. The subclasses of the Persistent class had to implement the methods *read()* and *write()*, which reads/writes from/to a file. The failure to make data persistent occurred relatively often thus indicating that this was difficult for the participants.

Compared with the previous experiment, these students were more challenged to understand impacts on classes but performed better on inherited functionality. The latter is a specific topic in the Object Oriented Programming unit considered as a core attainment (section 2.1). In contrast comprehending the structure of a medium-sized object oriented system may not have been studied by some students and this may have reduced their comprehension of impacts.

The present cohort has relatively few problems with attributes and methods in the wrong class (2.2.2.2-3) or their removal (2.2.3.1). This may have been because of the importance placed on understanding the fundamentals of classes, objects and methods in the Java Programming



foundation unit. The BlueJ environment <sup>1</sup>, used for this unit, is especially good for illustrating these concepts, perhaps at the expense of practice with larger programs.

	Task1	Task2	Task3
1 General			
1.1 Program logic	20	3	
1.2 GUI			
1.2.1 Forgot to expand the window		6	
1.2.2 Little experience with GUI programming	2	1	
1.3 Object-oriented programming			
1.3.1 Initialise objects			2
1.3.2 Instantiate a class			1
1.3.3 Understand and use a Java API class			2
1.3.4 Reuse of methods		2	
1.4. Algorithms			10
1.5 Programming environment	1		1
2 Specific			
2.1 GUI			
2.1.1 Changing interface		1	
2.1.2 Adding or removing GUI components	10	4	3
2.1.3 Removing label declarations	19		
2.2 OO comprehension and programming			
2.2.1 Overall program structure	2	2	
2.2.2 Impacts on classes	1	10	3
2.2.2.1 Self-reported problems			
2.2.2.2 Attributes in the wrong class			2
2.2.2.3 Methods in the wrong class			
2.2.3 Impacts within class	1		
2.2.3.1 Removing variables and methods	3		
2.2.4 Inherited functionality	2	6	4
2.3 Testing procedure	1	1	1

**Table 1: Number and type of difficulties per tasks**

## 4 Limitations of this Study

The data on difficulties experienced by the participants are partly qualitative and subjective. A detailed list of errors for each participant made by the independent assessor (difficulties that the participants did not overcome) was combined with the difficulties identified in the collected feedback. However, one should be aware that there might be differences among participants. Some might have forgotten to report their difficulties. Hence, some of the difficulties the participants had during the experiment that they managed to overcome might be missing from our list.

One should also be aware that the majority of the participants did not finish Task 3. Hence, the list of difficulties for this task is not complete.

The experiment lasted only seven hours, which might be too short a time to become familiar with the application. Future work should therefore include case studies that last longer.

## 5 Conclusions and future work

This paper provides further empirical evidence with respect to the difficulties students had while conducting maintenance tasks on a medium-sized Java application. It allows generalisation of the results of the previous study by Karahasanovic *et al.* (2006) to the population of advanced beginners. The results revealed the difficulties the participants had due to a lack of knowledge that is independent of the specific application (general knowledge) and a lack of knowledge of the specific application (specific knowledge). The major general difficulties that prevented the participants from completing the tasks were related to program logic and algorithms. These difficulties were also identified in the previous experiment. These findings can be used for improving courses on data algorithms.

The major specific difficulties that prevented the participants from completing the tasks were related to finding impacts of changes (removing label declarations and impacts on classes) and inheritance of functionality. The same difficulties were identified in the previous experiment. However, there were some differences. UWA students were more challenged to understand impacts on classes than students in Norway. On the other hand, UWA students performed better on inherited functionality. This can be explained by a different syllabus within a broadly similar degree. Findings on differences between universities in different countries can be used for further improvement of their syllabuses.

Based on these results we recommend introducing examples of modifying larger object-oriented programs in courses of object-orientation. Students should learn the basics of impact analysis earlier in their computer science education. However, this does not mean that the training in understanding the fundamentals of classes, objects and

<sup>1</sup> [www.bluej.org](http://www.bluej.org)

methods as provided at UWA should be reduced. The BlueJ environment can be recommended for illustrating the fundamentals of object-orientation.

What needs to be taught to improve effectiveness at maintenance is a complex question. Efforts have been made by the community to introduce the theory and practice of maintenance in computer science education (Austin and Samadzadeh, 2005; Postema *et al.*, 2001). Nevertheless, large number of students would leave their universities without any maintenance experience. Furthermore, their understanding of object-oriented concepts gained through the introductory programming courses affects their ability to maintain such systems later on. We thus believe that students should obtain some experience in understanding and modifying larger programs earlier in their education. Identifying difficulties the students had while conducting maintenance tasks is only a first step towards improving their education. We intend to further explore interactions between programming and general problem-solving knowledge, and the effects of these interactions on the students' ability to maintain larger object-oriented applications.

## Acknowledgements

Amela Karahasanovic acknowledges support from the University of Western Australia for the Gledden Visiting Senior Fellowship.

The authors are grateful to Kesaraporn Techapichetvanich and Junisilver Taij for valuable contributions to this paper. We thank Gunnar Carelius, Ashley Chew, Ryan McConigley and Laurie McKeaig for their considerable technical help, and Louise Bolitho and Jeff Pollard for administrative help. We are grateful to Kaja Kværn and Gøril Tømmerberg for preparing experimental materials. We thank the students of UWA for participating in the experiment.

## References

- Arisholm, E., Sjøberg, D.I.K., Carelius, G. and Lindsjörn, Y. 2002. A Web-based Support Environment for Software Engineering Experiments. *Nordic Journal of Computing*, 9, No. 4, 231–247.
- Arisholm, E., Sjøberg, D.I.K. and Jørgensen, M. 2001. Assessing the Changeability of two Object-Oriented Design Alternatives – a Controlled Experiment. *Empirical Software Engineering*, 6, No. 3, 231–277.
- Austin, M.A. and Samadzadeh, M.H. 2005. Software comprehension/maintenance: an introductory course. *18th Int. Conf. on Systems Engineering (ISCEng'05)*, IEEE, 414–419.
- Basili, V.R., Shull, F. and Lanubile, F. 1999. Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering*, 25, No. 4 (July–Aug.), 456–73.
- Coleman, D., Ash, D., Lowther, B. and Oman, P. 1994. Using Metrics to Evaluate Software System Maintainability. *IEEE Computer*, August 1994, 44–49.
- Dale, N. 2005a. SIGCSE members survey. <http://www.cs.utexas.edu/users/ndale/ContentResults.html>
- Dale, N. 2005b. Non SIGCSE members survey. <http://www.cs.utexas.edu/users/ndale/ContentResults2.html>
- Détienne, F. 1997. Assessing the cognitive consequences of the object-oriented approach: A survey of empirical research on object-oriented design by individuals and teams. *Interacting with Computers*, 9, 47–72.
- Dreyfus, H.L. and Dreyfus, S.E. 1986. *Mind over Machine*. New York: The Free Press.
- Eriksson, H.E. and Penker, M. (1998). “Case Study”. In: *UML Toolkit*. (editors), New York, John Wiley & Sons, Inc.,
- Gunderman, R.E. (1988). “A glimpse into program maintenance”. In: *Techniques of program and system maintenance*. Parikh, G. (editors), Wellesley, MA, QED Information Sciences Inc., 55–59.
- Holgeid, K.K., Krogstie, J. and Sjøberg, D.I.K. 2000. A Study of Development and Maintenance in Norway: Assessing the Efficiency of Information Systems Support Using Functional Maintenance. *Information and Software Technology*, 42, No. 10, 687–700.
- Jørgensen, M. and Sjøberg, D.I.K. 2002. Impact of experience on maintenance skills. *Journal of Software Maintenance and Evolution: Research and Practice*, 14, 123–146.
- Kajko-Mattsson, M., Forssander, S. and Andersson, G. 2002. Developing CM3: Maintainers' Education and Training at ABB. *Computer Science Education*, 12, No.1–2, 57–89.
- Karahasanovic, A., Anda, B., Arisholm, E., Hove, S.E., Jørgensen, M., Sjøberg, D.I.K. and Welland, R. 2005. Collecting Feedback during Software Engineering Experiments. *Empirical Software Engineering*, 10, No. 2, 113–147.
- Karahasanovic, A., Levine, A.K. and Thomas, R. 2006. Comprehension strategies and difficulties in maintaining object-oriented systems: an explorative study, *Journal of Systems and Software*. Forthcoming.
- Lehman, M. and Belady, L.A. 1985. *Program Evolution – Processes of Software Change*. Academic Press, London.
- Lientz, B.P. 1983. Issues in Software Maintenance. *Computing Surveys*, 15 (3), 271–278.
- Nosek, J.T. and Prashant, P. 1990. Software Maintenance Management: Change in the Last Decade. *Journal of*

- Software Maintenance Research and Practice*, 2, No. 3, 157–174.
- Pfleeger, S.L. 1987. *Software Engineering – The Production of Quality Software*. Macmillan.
- Postema, M., Miller, J. and Dick, M. 2001. Including practical software evolution in software engineering education. *14th Conference on Software Engineering Education and Training*, IEEE, 127–135.
- Thomas, R., A. Karahasanovic, and Kennedy, G. 2005. An Investigation into Keystroke Metrics as an Indicator of Programming Performance. Proc. of the 7th Australasian Computing Education Conference 2005 (ACE 2005), Conference in Research and Practice in Information Technology, Newcastle, Australia, Australian Computer Society, Inc., 42, 127–134.
- Thomas, R., Kennedy, G., Draper, S., Mancy, R., Crease, M., Evans, H. and Gray, P. 2003. Generic Usage Monitoring of Programming Students. *ASCILITE 2003 Conference, University of Adelaide, Australia*, Adelaide, Australia, ASCILITE, 715–719.
- Von Mayerhauser, A. and Vans, A.M. 1995. Industrial Experience with an Integrated Comprehension Model. *Software Engineering Journal*, 171–182.
- Von Mayrhauser, A. and Vans, A.M. 1995. Program Comprehension During Software Maintenance and Evolution. *Computer*, 28(8), 44–55.
- Zelkowitz, M.V. 1978. Perspectives on Software Engineering. *ACM Computing Surveys*, 10, No. 2, 197–216.



# Learner Reflection in Student Self-assessment

Judy Kay, Lichao Li and Alan Fekete

School of Information Technologies  
The University of Sydney, NSW 2006, Australia

{judy, lli1, fekete}@cs.usyd.edu.au

## Abstract

Learner reflection is critical to effective, deep, transferable learning, especially in cognitively demanding areas, such as learning programming. This paper presents Reflect, a programming education system, which aims to facilitate student self-assessment and promote learner reflection through scrutable learner models. Reflect sets tasks and presents a set of example answers to read and assess. We report students' use of the system over one semester. Overall Reflect appears to help students understand the teacher's goals for the tasks set and to refine their own work in relation to these goals.

## 1 Introduction

Learner reflection is "a generic term for those intellectual and affective activities in which individuals engage to explore their experiences in order to lead to a new understanding and appreciation" (Boud, Keogh et al. 1985). There is a large body of evidence suggesting that learning effectiveness can be enhanced when learners pay attention to their own learning experiences by reflecting on the state of their knowledge and the learning process (Schön 1983; Boud, Keogh et al. 1985; Schön 1987). Learner reflection is especially important for cognitively demanding learning topics, such as learning programming. Schön identified three types of learner reflection, namely "reflection-in-action", "reflection-on-action" and "reflection-on-reflection" (Schön 1983; Schön 1987), that may occur in any learning experience. We explore ways to support the first two of these in an intelligent programming education system named Reflect.

There are several examples of systems that capture a model of the learner and make it available to support learner reflection: for example, the skill meter in ACT programming tutors (Corbett and Anderson 1995), the course progress chart in ELM-ART (Weber and Brusilovsky 2001), as well as in (Bull and Pain 1995) and (Hartley and Mitrovic 2002). Our system is unique in several aspects. First of all, Reflect aims to promote learner reflection through showing the learner a model of their progress in the process of student self-assessment.

Student self-assessment refers to "the involvement of students in identifying standards and/or criteria to apply to their work, and making judgments about the extent to which they have met these criteria and standards" (Boud 1991). It is generally acknowledged that student self-assessment contributes to the development of meta-cognitive skills, in that it can help learners develop the capacity to identify their strengths and weaknesses and direct their study to areas that require improvement. (Boud, Keogh et al. 1985) Such learning skills often characterize effective learners (Schön 1983; Boud, Keogh et al. 1985; Schön 1987). Student self-assessment has two key elements: the learner's identification of criteria or standards to be applied to their own work and the making of judgments about the extent to which work meets these criteria.

To make the student's self-rating in line with the teacher's judgment of the student's work, explicit criteria for satisfactory and unsatisfactory performance need to be established (Boud 1989). Teachers should design learning tasks to meet learning goals. In Reflect, the teacher makes these explicit by defining evaluation criteria for each task, and students need to self-assess themselves with criteria supplied by teachers. Although it is common practice in classroom teaching to encourage students to review grading criteria (Fekete, Kay et al. 2000), a system like Reflect is novel. Another distinctive feature of Reflect is that it lets students study examples and assess them according to criteria.

The work reported in this paper was conducted in the context of learning programming in C in an undergraduate subject, Software Construction I, which is a second year programming course that aims to teach programming in C in a UNIX environment. Reflect is one of the learning support tools designed to help students improve their learning and helping them pass the course. The system has evolved considerably since it was first built in 2002. Its original objectives were to enable students to self-assess their knowledge, monitor their learning progress and judge if they have achieved the required learning outcomes. This paper describes the latest version of Reflect with a learner modelling component that supports more intelligent feedback on learner's progress, thereby promoting learner reflection more effectively.

The following section gives an overview of Reflect. Section 3 describes the way that students used the system and Section 4 has further work and conclusions.

## 2 System Description

<b>Counting Elements</b> You are required to iterate through a singly linked list of nodes and return the number of occurrences of a value. The lists data structure is provided below: <pre>typedef struct node {     int n_value;     struct node *n_next; } Node;</pre>	This task aims to improve your understanding of the following learning objectives: <div style="display: flex; justify-content: space-between;"> <div style="width: 48%;"> <b>1. Coding Style</b> - Good Use of Indentation  <b>3. Coding Style</b> - Comments  <b>5. Dynamic Data Structure</b> - Linked List Traversal         </div> <div style="width: 48%;"> <b>2. Coding Style</b> - Useful Identifier Names  <b>4. Similar concepts in both C and Java</b> - Control Flow         </div> </div>
<b>Step 1: Examples</b> Read and assess example solutions: <a href="#">Example 0</a> <a href="#">Example 1</a> <a href="#">Example 2</a>	
<b>Step 2: Fill in the routine body</b> <pre>int count(Node *start, int val) {</pre> <div style="border: 1px solid #ccc; height: 150px; margin: 5px 0;"></div> <pre>}</pre> <p>Alternatively, you can <b>upload</b> your solution here (The texts in the above textarea will be ignored if you upload a file):</p> <div style="border: 1px solid #ccc; width: 100%; height: 20px; margin-bottom: 5px;"></div> <div style="text-align: right;"> <input type="button" value="上传"/> </div>	
<b>Step 3: Self-assess</b> <div style="text-align: right;"> <input type="button" value="Save and assess"/> </div>	

Figure 1. Problem Statement

To accomplish our research goal in the context of teaching programming in C in UNIX, domain knowledge of the subject was analyzed and learning objectives as well as students' common misconceptions (Fekete, Kummerfeld et al. 2003) were identified and summarised. Based on this information, a pragmatic learner model was defined. This was used to create a model for each student to monitor their knowledge state and learning progress. Students' interactions with the system are recorded in their learner models. Thus, these models always hold the system's beliefs of how well the students are performing in the subject. Making this information available to students by scrutinizing their user models, can help them become aware of their learning progress, thereby promoting learner reflection (Zapata-Rivera and Greer 2001).

Reflect has a collection of tasks for students to self-assess. As Figure 1 shows, there are three steps in the student self-evaluation process:

1. Students read and assess example solutions provided by the teacher;
2. They provide their own solutions to the problem and;
3. They self-assess that solution using criteria the teacher has defined for that task.

On the problem statement page, at the upper right in Figure 1, there is a list of learning objectives that the teacher would want students to learn from the task. The system keeps all these learning objectives in a dictionary, which represent the domain knowledge. When reading and assessing example solutions as well as writing their own answer to the problem, students need to pay special attention to such learning objectives. For example, in Figure 1, the task is intended to help students learn about

1. Coding style: Good use of indentation, useful identifier names and comments;

2. Similar concepts in C and Java: Control flow and;
3. Dynamic data structure: Linked list traversal.

To begin the self-assessment process, students are required to read and assess all examples the teacher provides. These examples are not necessarily 'perfect' solutions. Rather they provide opportunities to explore interesting and important ideas associated with the learning goals. We usually create these examples, which demonstrate common misconceptions, poor style and similar elements, after grading final exam questions. We also strive to provide multiple examples, so we can illustrate different ways to do one task.

Students also need to assess the example solutions they read with criteria provided by the teacher, as in Figure 2. Each criterion corresponds to one of the learning objectives of the task. Consequently, each criterion asks the student to rate one key aspect of the example solution. For instance, in Figure 2, it asks about the coding style of the solution. The student can choose from a range of options for each marking criteria, for example, excellent, good, ok, poor or rotten. These example solutions have been pre-assessed by teachers using the same criteria. Students' assessments are then compared to the teacher's assessment, as illustrated in Figure 3. The comparison gives the student feedback on:

- How the teacher marked the example;
- The difference between the teacher's assessment and the student's assessment and;
- Why the teacher assessed it that way.

Since the marking schemes used to assess the solution are consistent with the learning objectives of the task, the

discrepancy between the student's and the teacher's assessments indicate how well the student comprehended these objectives. The probability that the student has understood each learning objective is then estimated and recorded in his/her learner model, updating the system's belief of how well he/she understands that objective. This information is used to illustrate the student's learning progress to promote learner reflection-on-action, as upon seeing this information, the student may explore why the teacher thinks differently from him/her and learn how to think as the teacher does, so to achieve a new level of understanding of the issues.

**Assessing SOFT2130 Counting Elements example 0**

You should now assess the example using the following marking scheme. This window can be moved so that you can view both the example solution and the marking scheme at the same time.

Rate the style of the code, especially with respect to the use of sensible identifier names.	no opinion
Rate the style of the code, especially with respect to the good use of indentation.	no opinion
Rate the style of the code, especially with respect to comments presented.	no opinion
Rate the design of the loop, with respect to: <ul style="list-style-type: none"> <li>Sensible exit condition in loop.</li> <li>Non-recursive solution.</li> <li>Iterating the loop correctly</li> </ul>	no opinion
Rate the overall correctness of the code.	no opinion

Done

**Figure 2. Solution Assessment**

**Comparing assessments for SOFT2130 example 0**

Criteria	Yours	Ours	Disc
Rate the style of the code, especially with respect to the use of sensible identifier names.	excellent	poor	3
Rate the style of the code, especially with respect to the good use of indentation.	good	excellent	1
Rate the style of the code, especially with respect to comments presented.	ok	ok	0
Rate the design of the loop, with respect to: <ul style="list-style-type: none"> <li>Sensible exit condition in loop.</li> <li>Non-recursive solution.</li> <li>Iterating the loop correctly</li> </ul>	poor	excellent	3
Rate the overall correctness of the code.	rotten	excellent	4
<b>Total discrepancy</b>			<b>11</b>

Aim to minimise the discrepancy between your assessment and ours.

**Explanation of our assessment**

The variable name foo is not very descriptive of it's purpose.  
The comments do not increase the level of understanding of the code.

done

**Figure 3. Comparison of Student's (Yours) and Teacher's (Ours) Assessments of an Example and a Measure of the Discrepancy between Them**

Once the student has viewed and assessed all the example solutions of the task, they can attempt their own answer to the problem; submitting it in the large answer pane in Figure 1. Then, the *self*-assessment phase begins, shown as Step 3 in the figure.

Students assess their own solutions just as they did for the example solutions, using the same marking criteria, as shown in Figure 2. Importantly, when students self-assess their solutions to a problem, it encourages reflection-in-action (Schön 1983; Schön 1987). They are able to read each marking criterion and reflect, as they rate a solution with respect to the criterion. Since they have read and assessed the example solutions in Step 1, they can

rate their own solutions more accurately. As the criteria are shown in a new pop-up window, they are more likely to consider each criterion as they review their own solution. For instance, the second element of the fourth marking criterion in Figure 2 explicitly requires students to supply non-recursive solutions, students may realize that they had used a recursive solution, which is not preferred or encouraged in our subject. Because these criteria are created by teachers to be consistent with the teaching goals, when students assess their solutions with the set of marking criteria, they will be helped to reflect and learn, focusing on the relevant learning objectives. After self-evaluation, the students' solutions and assessments are saved, so that they can later reassess the solution. This concludes the three steps of student self-assessment process in Reflect.

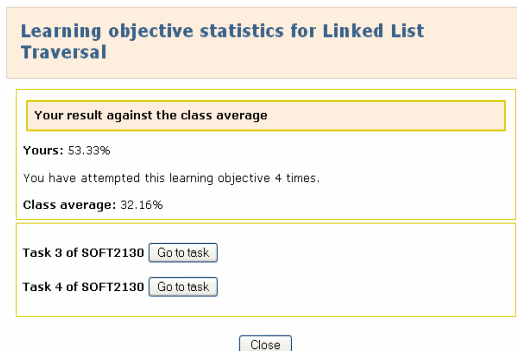
Reflect also provides each student with an individualized user profile, which displays the student's learning progress on a concept-by-concept basis in Scrutable Inference Viewer or SIV (Uther 2001; Kay and Lum 2005). SIV displays a user model and allows it to be viewed in different ways. On the profile page, next to SIV, there is also a list of hyperlinks for each learning objective. If the student clicks on one of them, more statistics about the concept are presented in a popup window, as shown in Figure 4. This information includes the student's mark for the concept, the number of times the student has attempted it, the class' average mark for it and the tasks that have it as a learning goal.

The user profile promotes reflection-on-action (Schön 1983; Schön 1987) as it shows students a visualisation of their learning progress by externalising their learner models with SIV (Zapata-Rivera and Greer 2001). It allows a user model to be explored in certain ways, such as selecting, deselecting and expanding a user model component and displays the model accordingly. In the SIV display, teaching/learning goals are displayed with different indentations and in different colours. In particular, it answers the four questions, which should be addressed by a scrutable learner model, mentioned in (Kay 1997), as it shows

- What they know (the learning goals that are displayed in green in SIV);
- How well they know it (the saturation of colours, if a student knows a concept better, the concept is displayed in SIV in darker green);
- What they do not yet know (the learning goals that are displayed in red or yellow in SIV) and;
- How to learn it (what tasks aim to teach the concept, e.g. in Figure 4, the student can do *Task 3* to learn about *Linked List Traversal*).

From this information, students are encouraged to think about what they have learnt and how they have learnt it, i.e. reflecting on their experience. Furthermore, students are able to compare the system's beliefs about their C programming knowledge with their own beliefs. In this way, reflection is further encouraged; especially if the system's beliefs and the student's own beliefs differ. SIV can also show students what and how the learning

concepts are related by displaying them in different sizes, with larger ones more closely related to the currently selected concept. It then becomes possible for students to improve their understanding of one concept by working with closely related concepts.



**Figure 4. More Useful Statistics**

In summary, both learner reflection-in-action and reflection-on-action, facilitated by scrutable learner modelling are supported by the process of assessing solutions, reviewing overall progress with SIV and reviewing detailed performance comparison against the class.

### 3 Evaluation

In this study, a general overview of students' system usage is presented and a selected group of students' submitted solutions were reviewed in detail qualitatively to reflect their underlying learning behaviour.

As described in Section 2, students interact with the system when they view and assess example solutions and when they write and self-assess their own solutions. Their assessments of example solutions were compared with the teacher's assessment. This was used to calculate a value indicating how well the student understands the learning objectives. Such values were saved and amalgamated over time to create individual learner models. The students' own solutions were also saved in the system.

Reflect activities have been integrated into the laboratory work in our C programming in UNIX subject. Students needed to complete Reflect tasks before or during weekly laboratories for course credit. Typically, each week, students were required to do one or two tasks. Students' motivation was increased when they became aware that the Reflect tasks were based on past exam questions. Our records show that the system was used by most of the class. Furthermore, it is to be noted that although completing Reflect tasks was part of the subject's assessment, there were still students who did not use the system as instructed, partially because credit associated with the system was about 5% of the total assessment. Nevertheless, many students used Reflect as an extra learning resource in addition to the lectures and tutorials they attended, as shown from our analysis of the learner models and system logs.

There were over 260 students enrolled in the course in 2005. Forty-six students' activities were analysed in detail. Of these 46 students, 20 earned final grades at the top of the regular class, and 26 were borderline students (they either just passed the subject or just failed). They were chosen because these are two interesting populations to study. Table 1 gives an overview of the overall activity of these students. It shows the average number of tasks each student attempted, the average number of solutions he/she provided (where multiple solutions could be submitted), the average number of self-assessments he/she made as well as average the number of evidence the system gathered from each student. Fifteen tasks available, with thirteen set as part of the subject's assessment. The top 20 students used the system much more than the borderline students. Even so, only about 60% of the top students used the system. Notably, one borderline student used the system very frequently, and the system gathered 266 pieces of evidence about this student. Excluding this student, the average number of evidence gathered from borderline students was 25.7.

	Tasks attempted (max =15)	Solutions provided	Solutions self-assessed	Evidence gathered
<b>Top 20 students</b>	8.3	10.8	8.6	60.5
<b>Borderline students</b>	3.7	5.04	3.04	34.9

**Table 1. Summary of Students' System Usage**

To gain a better understanding of students' learning behaviour when using Reflect, we conducted a detailed qualitative evaluation of the actual solutions that students submitted. Sixteen students, with whom we had personal contact during the semester, were chosen. We selected students we knew to be hard working during the semester and likely to make serious attempts at practical work. These students are representative of a broad range of performance levels with examination marks range from the top of the class (81/100, the pass barrier mark is 40) to the bottom (17/100). Eight of them passed the exam, and the other eight failed. In this study, these students' answers to four Reflect problems were reviewed. The four tasks are List Traversal and Counting Elements, which are two linked list traversal problems, and Test Script and Fibonacci, which are UNIX shell scripting questions. They were chosen because dynamic data structures and UNIX concepts were two of the most important components of the subject curriculum and, thus, most students made attempts to do these tasks.

The List Traversal task requires students to write a function that goes through a linked list and find the average value of all the integers in the list. Since students had the opportunity to read our example solutions first, it is not surprising to find that their own solutions are influenced by our code. Unfortunately, sometimes they were too influenced by the incorrect answers, their answers were exactly the same as the supplied examples. Nonetheless, we found that more able students were more likely to provide a solution that was different from the



examples and they could identify errors in our examples (recalling from Section 2 that our example solutions are not “perfect”) and rectified them in their solutions. We call these “authentic” answers. For example, for the List Traversal task, eleven students submitted answers and eight of them passed the exam. Four students who passed the exam provided authentic answers. The task has two examples answers illustrating different misconceptions. The remaining seven students had solutions similar to the more correct example, which is presented below:

```
float meanAge(Node *start) {

    int sum, count;

    while (start) {

        sum += start->age;

        count++;

        start = start->next;

    }

    return (float) (sum / count);

}
```

This answer is mainly correct, but with a critical problem. The integer variables, sum and count, are not initialized and using un-initialized variables can lead to unexpected program behaviour. Our commentary on the examples explained this. Four other students who passed the exam had solutions that were similar to this example, but corrected the error. For example, one of them had the following answer:

```
float meanAge(Node *start) {

    float sum = 0;

    float count = 0;

    while(start != NULL) {

        sum = sum + start -> age;

        count = count + 1;

        sum = start -> next;

    }

    return sum/count;

}
```

This solution initialized the integer variables but still failed to handle the case when the variable “count” is zero. On the other hand, although the three less able students had solutions similar to this more correct example, they also reproduced the un-initialized variable error. It is apparent

that they were not able to recognize this error from our examples. There was also one student who submitted two solutions: the later submitted solution showed corrections and improvements to the first submission.

The Counting Elements task is similar to the List Traversal task. Students need to write a function that traverses a given linked list with each node containing a single integer value and count the occurrences of a particular value. There are three example solutions. One of them is correct. All fourteen students who submitted a solution were able to avoid the mistakes from the two incorrect solutions. However, four students (two passed the exam, two failed) submitted solutions that are similar to the correct example but incorporated new errors that were not present in the two erroneous examples.

The Test Script task in Reflect asks students to write an UNIX shell script for testing that is flexible and versatile. We provided three examples. Two of them were clearly wrong and one was correct. By studying the students’ submissions, we had similar findings to the linked list questions. Eight of thirteen students who provided a solution had authentic or solutions that were similar to the correct example. One top student submitted an authentic solution. Three failing students and one student with 80/100 in the exam copied the wrong example. This again shows that weaker students were unable to recognize the errors in the examples.

The Fibonacci shell scripting task incorporates UNIX concepts such as pipes and redirections. We provided two example solutions, one correct and one with serious mistakes. We observed that most students’ solutions were a copy of or very similar to the correct example (12 of 14 students who made submissions). Of the two remaining students, one who failed the exam copied the example with serious mistakes and the other who achieved the top mark (81/100) provided an authentic solution but with errors.

We also reviewed how students self-assessed their solutions and found that the more able students rated their own solutions more carefully. The self-assessments of their own answers were often similar to how a teacher would assess them. The less able students, on the other hand, tended to over-rate their solutions. This was in line with earlier studies from (Boud 1989).

To sum up the qualitative evaluation, we found that students were strongly influenced by the supplied code. However, it also shows that the approach of reading examples before writing a solution does necessarily enable students incorporate common elements from our code. The examples are beneficial to learning so long as the tasks are designed to ensure that straight copying is not attractive.

#### 4 Discussion and Conclusions

So far, our experience with Reflect has been in a subject devoted to programming, and so all of the examples have been questions where the task is primarily synthesis. In other fields of computing, the nature of assessment tasks is often quite different, and this raises some issues for a self-assessment system such as Reflect. We next offer

some thoughts on how to use our system with theory material, from a data structures subject.

Traditional data structures classes use assessment with a large weighting on questions requiring tracing an operation. For example, students are asked to show how a binary search tree will be modified when a particular element is deleted. These questions are usually graded as correct or incorrect, without much consideration of partial marks, and so the self-assessment system is of limited use. If a correct answer is shown, the student will not have difficulty answering themselves. Thus the best use of Reflect is to show students purported solutions that result from common misconceptions (these are like appropriate distracters in multi-choice questions). Identifying that the answers are wrong can help students remove these misconceptions in their own thinking, and this process is assisted by the message justifying the lecturer's low score on "correctness", which is displayed after the student has rated the purported solution.

Another common type of assessment task in the data structures subject requires the student to explain or to prove something. Here it is easy to offer the student many interesting potential answers, each with a mixture of good and bad features. The difficulty for the instructor is to choose assessment criteria which apply sensibly to "answers" with a mistake that means the flow-on reasoning takes an incorrect path. Of course, this is also a difficulty when designing the marking scheme for conventional assessment questions. For example, in a question on collisions in hashing, many students are confused between open and closed hashing. With Reflect, it would be good to use a purported answer to build awareness of this error, but then how should one score a criterion like "explains how the algorithm re-establishes the data structure invariants after the operation"? We suggest having marking criteria which divide the ideas up more finely. In the example, one could define three criteria: "explains what invariants the data structure must satisfy" and "states that the data structure must be re-established after the operation" and "explains what invariants are violated after the initial changes" and "explains how the structure can be modified to re-establish the invariants". A side-benefit of such a detailed set of assessment criteria, is that it indicates to students a good format for presenting their own explanation.

One question type is very important for data structures, and can be treated in either of the previously mentioned ways. This is where students need to analyze the runtime cost of an operation. We feel that Reflect is best used if the question asks to give the expression for the cost and justify the answer. Here a nice use of Reflect is to define the marking criteria to include separate aspects for each of the main ideas that are needed in an answer. For example, one can have one criterion which is "does the answer show awareness that the operation acts on each level of the tree", and another which is "does the answer show awareness that the worst-case number of levels is linear in the number of nodes", and yet another which is "does the answer show awareness that an operation which does linear work in some cases, and constant work in others, has worst-case cost which is linear".

We are currently exploring improvements for Reflect. The first, and most pressing, is to provide automatic evaluation of a student's own solution. If students' solutions can be automatically evaluated and the testing results can be resolved and fed into the learner models, we can provide more concrete feedback, which should make this part of the system more valuable to students, possibly overcoming the overestimation problem in the current system. A second goal is to provide learning guidance. With the existing learner models, it is possible to adaptively guide students, based on their knowledge state and select the most suitable learning path for them. Since learner reflection mostly occurs when the learner feels challenged (Boud, Keogh et al. 1985), appropriate choice of tasks for the individual can may promote learner reflection, especially reflection-in-action. Furthermore, we want to investigate further the impact on students' learning behaviour if they read examples first, compared with writing the program first and then reading example solutions.

In summary, we have presented Reflect, a student self-assessment system with support for learner reflection. Two types of learner reflection, namely reflection-in-action and reflection-on-action are supported by allowing the student to self-assess solutions and by providing intelligent and informative learning progress feedback through open learner modelling. We presented a qualitative analysis of student use of the system. In light of our earlier studies, which have shown that students feel comfortable using the interface that displays the learner model (Uther 2001) and they value the reflection opportunity and experience in general, we believe that Reflect can provide motivated students with help that they value and the system measures give a good indication of student's ability to "see" code from the perspectives that the teacher intended that they consider for their current learning objectives.

## 5 References

- Boud, D. (1989). "The role of self-assessment in student grading." *Assessment and Evaluation in Higher Education* **14**(1): 20-30.
- Boud, D. (1991). "Implementing Student Self-Assessment." Campbelltown: Higher Education Research and Development Society of Australasia. HERDSA Green Guide **5**.
- Boud, D., R. Keogh, et al. (1985). Promoting reflection in learning: A model. *Reflection: Turning Experience into Learning*. D. Boud, R. Keogh and D. Walker. London, Kogan Page: 18-40.
- Bull, S. and H. Pain (1995). "Did I say what I think I said, and do you agree with me?" Inspecting and questioning the student model. *World Conference on Artificial Intelligence in Education*, Charlottesville, VA, AACE.
- Corbett, A. T. and J. R. Anderson (1995). "Knowledge tracing: Modeling the acquisition of procedural knowledge." *User Modeling and User-Adapted Interaction* **4**: 253-278.
- Fekete, A., J. Kay, et al. (2000). Supporting Reflection in Introductory Computer Science. *SIGCSE*.

- Fekete, A., B. Kummerfeld, et al. (2003). Identifying and improving the learning of borderline students in SOFT2004. Sydney, Australia, School of Information Technologies, The University of Sydney.
- Hartley, D. and A. Mitrovic (2002). Supporting Learning by Opening the Student Model. ITS 2002, Springer-Verlag Berlin Heidelberg.
- Kay, J. (1997). Invited keynote address: Learner know thyself: Student models to give learner control and responsibility. ICCE'97 International Conference on Computers in Education.
- Kay, J. and A. Lum (2005). Exploiting readily available web data for scrutable student models. 12th International Conference on Artificial Intelligence in Education, Amsterdam, Netherlands.
- Schön, D. A. (1983). The reflective practitioner. New York, Basic Books.
- Schön, D. A. (1987). Educating the reflective practitioner. San Francisco, Jossey-Bass Publishers.
- Uther, J. (2001). On the visualisation of large user models in web based systems. Basser Department of Computer Science. Sydney, The University of Sydney.
- Weber, G. and P. Brusilovsky (2001). "ELM-ART: An adaptive versatile system for web-based instruction." International Journal of Artificial Intelligence in Education(12): 351-384.
- Zapata-Rivera, J.-D. and J. E. Greer (2001). Externalising learner modelling representations. AI-Ed 2001 Workshop: External Representations in AIED: Multiple Forms and Multiple Roles. San Antonio, Texas.



# Differing Ways that Computing Academics Understand Teaching

## Raymond Lister

Faculty of IT, Uni of Technology,  
Sydney, Australia  
raymond@it.uts.edu.au

## Anders Berglund

Dept of Information Technology,  
Uppsala University, Sweden  
Anders.Berglund@it.uu.se

## Iiona Box

Faculty of IT, Uni of Technology,  
Sydney, Australia  
ibox@bigpond.net.au

## Chris Cope

Dept of Computer Science  
La Trobe University, Australia  
c.cope@latrobe.edu.au

## Arnold Pears

Dept of Information Technology,  
Uppsala University, Sweden  
Arnold.Pears@it.uu.se

## Chris Avram

Faculty of Information Technology  
Monash University, Australia  
chris.avram@infotech.  
monash.edu.au

## Mat Bower

Department of Computing  
Macquarie University, Australia  
mbower@ics.mq.edu.au

## Angela Carbone

Faculty of Information Technology  
Monash University, Australia  
Angela.Carbone@infotech.  
monash.edu.au

## Bill Davey

School of Business Inf Tech., Royal  
Melbourne Inst of Tech, Australia  
billd@rmit.edu.au

## Michael de Raadt

Faculty of Sciences, University of  
Southern Queensland, Australia  
deraadt@usq.edu.au

## Bernard Doyle

Faculty of IT, Uni of Technology,  
Sydney, Australia  
bjd@it.uts.edu.au

## Sue Fitzgerald

Info & Comp Sci, Metropolitan State  
University, St. Paul, MN USA  
sue.fitzgerald@metrostate.edu

## Linda Mannila

Turku Centre for Computer Science  
Turku, Finland  
Linda.Mannila@abo.fi

## Cat Kutay

School of Comp Sc & Eng, Uni of  
New South Wales, Australia  
ckutay@cse.unsw.edu.au

## Mia Peltomäki

Turku Centre for Computer Science  
Turku, Finland  
mia.peltomaki@utu.fi

## Judy Sheard

Faculty of Information Technology  
Monash University, Australia  
Judy.Sheard@infotech.  
monash.edu.au

## Simon

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

## Ken Sutton

Southern Institute of Technology  
Invercargill, Southland, New  
Zealand  
ken@clear.net.nz

## Des Traynor

Department of Computer Science,  
NUI Maynooth, Co Kildare, Ireland  
dtraynor@cs.nuim.ie

## Jodi Tutty

School of Information Technology  
Charles Darwin University, Australia  
jodi.tutty@cdu.edu.au

## Anne Venables

Victoria University  
Melbourne City, Australia  
Anne.Venables@vu.edu.au

## Abstract

This paper presents first results from a wide-ranging phenomenographic study of computing academics' understanding of teaching. These first results focus upon four areas: the role of lab practical sessions, the experience of teaching success, conceptions of motivating and engaging students, and the granularity of the teacher's focus. The findings are comparable with prior work on the

understandings of academics in other disciplines. This study was started as part of a workshop on phenomenography. Most participants at the workshop received their first training in phenomenography. This paper summarises the structure of the workshop.

**Keywords:** phenomenography, conceptions of teaching, computing education.

## 1 Phenomenography

While most readers of this paper would be familiar with the dual concepts of deep and surface learning, fewer might know that the origins of these concepts lie in phenomenographic research. Phenomenography is a research approach that focuses on the qualitatively different ways that people experience, understand,

perceive, or conceptualise a phenomenon (Marton 1986; Marton & Booth 1997; Berglund 2005).

A phenomenographic researcher typically gathers data by interviewing a number of subjects about a particular phenomenon. Analysis of the interviews then seeks to identify *variations* in the interviewees' perceptions of the phenomenon. Even when many people are interviewed, the analysis generally elicits only a small number of qualitatively different ways of experiencing the phenomenon.

These ways of experiencing the phenomenon are then delineated as distinct 'categories of description'. Each category represents a different conception, a different understanding, of the phenomenon being studied. It is often the case that the categories are hierarchical, with each new category supplementing, rather than supplanting, the lower levels of understanding.

Importantly, what is being categorised is the understandings, not the people who evince them. It is common for a single person in a single interview to express understandings from several different categories in the same hierarchy. The aim of a phenomenographical study is not to pigeonhole people but to categorise the full range of understandings of a phenomenon.

### 1.1 Phenomenographical studies of computing students

The seminal phenomenographic work in computing education was a study of student's conceptions of learning to program. Booth (1992) identified four qualitatively distinct ways in which students experience learning to programming: they experience it as learning a programming language; as learning to write programs in a language; as learning to solve problems; and/or as becoming part of the programming community.

Other phenomenographic work on student conceptions of programming includes that by Bruce et al (2004) and that by Stoodley et al (2004). Some very recent phenomenographic work explores student understandings of object-oriented concepts (Eckerdal & Thuné 2005; Eckerdal & Berglund 2005).

Further phenomenographic studies of computing students have explored educationally critical aspects of learning about information systems (Cope 2000) and students' understandings of network protocols (Berglund 2005).

### 1.2 Phenomenographical studies of academics

Academics' understandings of their teaching have been the subject of several past phenomenographic studies, most often of science teachers (Samuelowicz & Bain 1992; Prosser et al 1994; Trigwell et al 1994), but also in other areas such as mathematics (Runesson 2005), accounting (Levesson 2004), and economics (Pang & Marton 2003).

There have also been cross-disciplinary studies. The four personal theories of teaching by Fox (1983) are based upon his anecdotal encounters with newly appointed polytechnic teachers for "a number of years" (p151),

probably across a variety of disciplines. The 55 academics studied by Dunkin (1990) were spread across many disciplines. Samuelowicz and Bain (1992) interviewed 13 teachers, five who taught "science" (presumably one or more of the physical sciences) and eight who taught in the social sciences.

All of these studies have identified several qualitatively different understandings of teaching that teachers bring to the classroom. At one extreme, teachers focus on the content of their course, seeing teaching as the act of transmitting knowledge and concepts to the student. At the other extreme, teachers focus on the student, seeing teaching as the act of helping students to develop or change their own understandings.

Why are there so many phenomenographic studies of academics, ranging across so many disciplines? As Prosser et al (1994) explain, a person's understanding of teaching or learning

*"... needs to be identified and described within particular contexts, in terms of particular tasks and from the perspective of the learner or teacher within that context engaged in a particular task"* (p219)

Some characteristics of computer science as an academic discipline make it hard to learn and to teach: the subject area changes fast; it is known to be hard to learn (Ben-Ari 2001); it is simultaneously abstract or theory-based and concrete or skill-based; it requires an understanding of both the static and the dynamic properties of a computer. Therefore it would be no surprise if the ways computing teachers understand teaching were to differ from the ways teachers in various other disciplines understand it. Given Prosser's thoughts on the importance of context, it would be unwise to assume that the results from any of these studies transfer directly to computing.

While all of the studies referred to above identified different conceptions of teaching, and indeed different numbers of categories, all of their categories fell into two broad and well known groups, content- or teacher-focused and student-focused.

An early study that in some ways set the scene for this grouping is that by Fox (1983). Fox established four categories, or personal theories of teaching, two in each group.

#### 1.2.1 Content- or teacher-focused conceptions

Within the broad content- or teacher-focused category, Fox identifies two further categories of understanding, 'transfer' and 'shaping'.

In the transfer understanding of teaching, the focus is upon the knowledge of the discipline, with the student being a container into which the knowledge is to be poured.

In the shaping understanding of teaching, the student is viewed as a raw material to be moulded, or turned by some other 'manufacturing' process into a finished product. The domain knowledge is still the primary focus, and that knowledge is a specification of the product.

### 1.2.2 Student-focused conceptions

Within the broad student-focused category, Fox again identifies two categories of understanding, ‘travelling’ and ‘growing’.

In the travelling understanding, education is seen as a journey. Students have a ‘guide’ (a teacher) who leads them through the countryside, pointing out the major landmarks. While the discipline knowledge (the countryside) is still seen as separate from the student, the focus in this conception is on the student who is making the journey.

In the growing understanding, a pure constructivist perspective, the domain knowledge has no existence independent of the human mind. To explain this concept, Fox quotes Northedge (1976):

*“In this case we conceive of the teacher as a gardener with the student’s mind, as before, an area of ground. But this time I suggest we view the ground as already covered with vegetation (concept systems), some of which is clearly worth retaining and cultivating.”*

## 2 A phenomenography workshop for computing academics

Phenomenography is less widely used and less well known in computer education research than in education research in general. Even when recognised, the method is likely to be seen as not particularly useful or relevant. In a thorough survey of methods used to evaluate computer science teaching, Carbone and Kaasbøll (1998) wrote that

*“[Phenomenographic] studies like Booth’s are powerful ways for understanding how students think and how our teaching succeeds or misses out. Training as a social scientist is necessary to carry out such studies, and the studies are time consuming, so they are beyond the capabilities of most computer science teachers.”*

To help address this situation, the first five authors of this paper devised a two-day workshop that would introduce more computer education researchers to phenomenography by involving them in a major project.

It was clear that an empirical study focusing on computing academics’ perceptions of teaching would not be redundant. While one or two such studies have appeared recently (Lister et al 2004, Kutay & Lister 2006), much of the ground remains uncovered. A project such as this would provide a vehicle for teaching the use of phenomenography, while at the same time investigating whether the results of the earlier multidisciplinary and single-disciplinary studies have any bearing on understandings of teaching within the discipline of computing.

This paper briefly describes the workshop and presents some preliminary results from the phenomenographic research project around which it was built.

### 2.1 Prior work

Prior to the workshop, participants were required to read several papers (Åkerlind 2005; Booth & Ingeman 2002; Eckerdal & Thuné 2005; McKenzie 2002) and selected parts of a book (Ramsden 2003).

Also prior to the workshop, participants were required to interview one or more colleagues about their teaching and to transcribe the interviews. The interview script and instructions given in the appendix of Kutay and Lister (2006) were adapted to the purposes of this workshop.

In addition to the interview transcripts produced by the workshop participants, the transcripts from the Kutay and Lister study were also used as data for the workshop project, giving a total of 25 interview transcripts.

### 2.2 The workshop meeting and group work

The workshop itself occupied the two days immediately prior to ACE2006 in Hobart, Australia. It began with about half a day of formal instruction on phenomenography. Participants began working with the collected transcripts in the afternoon of the first day. By the end of that day, participants had broken into four groups and each group had chosen a topic area for their analysis of the transcripts:

- Group 1: The role of lab practicals
- Group 2: Conceptions of success
- Group 3: Motivation
- Group 4: Granularity of focus

The groups’ findings within their respective topic areas are described in the next four sections of this paper.

The second day began with another hour or so of lecture, after which the participants returned to their groups to continue their analysis of the transcripts. The groups then came together to present their preliminary findings, and the workshop leaders made suggestions as to how to proceed from that point.

### 2.3 Post-workshop analysis

The groups continued analysing the transcripts for several weeks after the workshop. These collaborations were sustained by email, voice-over-IP, and other means of communication. Periodically, when each group felt ready, they presented a written report of their findings to the workshop leaders, who responded with feedback. Most groups benefited from at least two iterations of feedback.

Sections 3 to 6 of this paper are reports on the findings of each of those groups. We have not attempted to synthesise the work of the groups into a single coherent whole; rather, we present them as four separate but related studies of the same interview transcripts. As a unifying feature, however, we compare the findings of each group with Fox’s categories of transfer, shaping, travelling, and growing.

The interview transcripts as used at the workshop were anonymous. The names of people and institutions were

removed, along with such things as explicit degree or subject names that might have identified the participants. Each transcript was given a code, and in the sections that follow, quotations from the transcripts are marked with these codes.

### 3 Group 1: The role of lab practicals

One group at the workshop investigated the understandings of lab practicals, which they defined as classes in a computer lab in which students work to learn the use of a software tool, device, or similar. For example, if students were seated at computers using word-processing software, their class would be called a lab practical if they were learning word-processing, but not if they were using the word processor to write about, say, database design and implementation. In the transcripts these classes are called by a variety of names, such as labs, workshops, or tutorials.

This group identified four categories of understanding of the role of lab practical classes:

- Acquiring and practising skills
- Reinforcing lectures and textbooks
- Refining and troubleshooting
- Applying skills

These findings are presented in detail in Simon et al (2006), and are summarised in the following four subsections.

#### 3.1 Acquiring and practising skills

When understanding the practical class as a means of acquiring and practising skills, academics perceive the class as somewhat independent of lectures. While the lectures will deal with the theory component of the course, the practicals are where the students learn about, acquire, and practise specific IT skills.

*“There’s nothing particular in the labs that reflects back on general lecture material. Because the labs are primarily focused on the Haskell language, it’s obviously related to any Haskell lectures I give, which is early in the semester, so there’s a kind of one-to-one correspondence there. But there’s not a great deal of correspondence to the general material or conceptual material that’s spread widely in [the course] because the labs are really focused on mainly learning a brand new programming paradigm, which is only one part of the whole course. So there’s not a great deal of cross-linking.” (L1)*

At this level, the class is highly structured and the teacher tends to assume the primary responsibility for the learning experience.

#### 3.2 Reinforcing lectures and textbooks

When understanding the practical class as a means of reinforcing lectures and tutorials, academics see it as the opportunity for students to put into practice the skills that

they have been taught in the lecture or the textbook. The learning is teacher initiated with focus on content. What is learnt at the practical is determined primarily by the teacher. The lectures, for example, will be used to teach and demonstrate a particular skill; then in the practical class, students will be given exercises in the application of that skill.

*“They link with the lectures in that we’ll cover something in the lecture, or I’ll say ‘you can do this’, and in the labs we’ll see how to actually do it.” (E4)*

While the class is still fairly structured, the students assume some responsibility for the learning experience, and would not be expected to benefit greatly from it unless they have studied the lecture or textbook material.

#### 3.3 Refining and troubleshooting

When understanding the practical class as a means of refining and troubleshooting, academics expect the students to do the bulk of the skill acquisition in their own time, and perceive the practical as a facility whereby students are provided with help on aspects of the work that they have found problematic.

In this category the student is expected to spend significant time prior to the lab acquiring the skills in question, so that the practical can be a productive troubleshooting session.

*“Well, I really like it if they do some themselves. Two things I expect beforehand. First of all... I encourage them... to work through the whole of the textbook so that when they come to the tutorials, they’re just doing the exercises that I’ve set them. And, if possible, they can do the exercises before the tutorial; then they only need to come to the tutorial and ask about anything they had trouble with, and they can perhaps go home early.” (E4)*

Responsibility for learning is now predominantly the student’s.

*“Some students will have done all the questions, and come in ready with their questions, the ones they had trouble with. Other students won’t have done anything, and they’ll start working... Everyone’s working at their own speed, covering the material. Some students will do all the questions, some won’t. It depends how much they’re willing to do beforehand at home.” (E4)*

#### 3.4 Applying skills

When understanding the practical class as a means of applying skills, academics no longer expect skills to be acquired in the class. The troubleshooting assistance is still provided, but in the context of applying the skills to a particular task such as a project or a major assignment.

As with the previous category, the students are expected to acquire the skills in their own time (or perhaps in earlier practical sessions), so that this practical can be devoted to work on the project. The practical is now of



less importance than the prior work, and can indeed become optional.

*"An hour a week of tutorial / computer laboratory. Not many turn up to that very often; although they're available, they normally do it in their own time." (E1)*

Responsibility for learning is thus almost entirely the student's, with the academic providing few or no instructions.

*"The following four [classes] are, as I say, basically one-liners, saying implement the philosophies and material from the [lectures]; for example, it might have been on help, it might have been on how to implement pop-up help in a web [page], so the tutorial might just say 'implement pop-up help in your assignment'. And that's it; that's what the tutorial says... I'm trying to wean them off, as much as possible, specific instructions on how to do a particular job, and get them to think about how it should be done." (E3)*

### 3.5 Relationship to prior work

This group identified four categories of description of the lab practical class, which bear a passing correspondence to the four categories of Fox.

Experienced as a means of acquiring and practising skills, the practical appears to conform to Fox's transfer category. As a way of reinforcing skills taught in lectures or the textbook, it could perhaps be seen as shaping. As a class for refining and troubleshooting, it is not unlike travelling. And as a means of applying skills acquired elsewhere, it fits clearly into the growing category.

## 4 Group 2: Conceptions of success

A second group focused on how IT academics experience success in their teaching. Of the 25 interview transcripts, 23 contained data pertinent to this question. In this analysis no distinction was made between different types of teaching approach such as lectures, tutorials, assignments, etc.

Analysis of the transcripts revealed three hierarchically inclusive categories of understanding of success. They are, in order of increasing sophistication:

- Success experienced as a perception
- Success experienced as delivery
- Success as developing student thinking

These three categories are discussed in greater detail in the following three subsections.

### 4.1 Success experienced as a perception

When experiencing success as a perception, the IT teacher has an intuitive feeling about successful teaching. Success is experienced as strongly connected to affective perceptions: either what teachers feel, or what they think or imagine that the students feel. At this level of understanding, teachers focus on their own perceptions

about their teaching. Often the teachers' comments about these perceptions are expressed in the context of explaining difficult IT concepts.

*"I think I have to say that I've come off most of my lectures fairly happy with the way it went." (E3)*

*"I suppose that a lecture that I've particularly enjoyed giving would probably qualify as one that I felt was most effective, wouldn't I?" (E2)*

*"It can be the way you've said it, or the material somehow resonated with their knowledge or their point of knowledge or their point of awareness at that particular point in time and somehow it allowed them to make that next step during the class and if that happens that's wonderful, you really get a positive feeling out of that ..." (T2)*

### 4.2 Success experienced as delivery

When experiencing successful IT teaching as delivery, the teacher distinguishes good delivery as an essential component of success. Not only does the teacher feel good about the teaching experience, he or she explicitly mentions the importance of a well-organised or effectively delivered lecture. Sequencing, or telling a good story, is seen as an illustration of successful teaching, supporting the belief that the discipline is sequential in nature. A well-organised presentation is linked to student satisfaction or perceived feelings of happiness.

*"I try to concentrate on one theme. And then I talk through the theme, if it doesn't take the 90 minutes, that's totally fine. If it takes more than 90 minutes, that's a situation I try to avoid so... they don't get too tired and too hungry. But I'd like to see one theme, one lecture." (J1)*

*"Well, an effective lecture was just ... demonstrating how to go from a static web page to a dynamic web page. Just being able to demonstrate the elements that have extra tags that, you know, this tag will invoke this, another program which gets all this data. So you can do that fairly simply." (D1)*

Delivery encompasses more than sequencing and organising the presentation of topics. Some transcripts indicate a need for abstract thinking and synthesis.

*"Okay, I guess it might be, after we discuss the concepts I like to show them some small programs of how we can implement those concepts. I once had this experience when I tried to be too ambitious that I showed them a program that had too many lines of code. I felt at the end of the day that I hadn't actually achieved what I set out to achieve, because I think I lose probably more than half of the class, in the sense that they can't follow those codes. I guess it would be more effective if I'd left the coding, the understanding of the coding to the students but to discuss more on the concepts rather than pay attention to the coding of the program." (F1)*

In this conception, good, inspiring demonstrations or examples are perceived as successful IT teaching.

*“Basically all fairly similar, but I think it’s good when I can provide a key example that encapsulates all of the concepts being considered, and I use that example well. ... They have a concrete idea of how it works that they can abstract from and build upon.” (R1)*

*“In general, the ones I think work better are ones where there is a fairly practical content, ones where students can actually perceive, if you like, an actual real-world application ... There’s one part of a lecture where I talk about a famous, if you like, disaster in computer programming back in the 60’s where a probe that was sent to Mars missed, it went off-course, it never made it; and the reason that it didn’t was because of a ... basically a fault in programming language design. It wasn’t so much a fault in the program that was written but it was in the design of the language ... That little bit is fun to explain and students really cotton onto it. And it’s not a unique example ... the bad design features of languages and so on are really fun to talk about and they’re ones that the students can really relate to. ... So concrete examples that are really clear and they can see the relation between the concrete example and the principle I am trying to talk about, they seem to work really well, and you get attention, they’re really focused on it, and they get something out of it.” (L1)*

Teachers who ‘own’ the subject matter express confidence that their delivery is successful:

*“Any material I produce rather than from the textbook, actually I feel very confident. And when I deliver I always I feel it is actually much better than the contents provided by the textbook. Because I would I say that I know the aim, why I put in this document? And I also know what I want to achieve through this tutorial.” (F2)*

In this category successful teaching is still experienced as a perception, but in addition teachers see specific techniques as essential to successful IT teaching. Examples must be well chosen and well explained; success looks a particular way; the story must be told well and the successful teacher knows how to do this.

### 4.3 Success as developing student thinking

When experiencing success as developing student thinking, IT academics understand successful teaching as inspiring students to engage in their own learning and express their own intellectual curiosity. Success is perceived in active engagement of the student, a goal that is actively pursued in the classroom. Successful teaching is conceptualised as enabling students to understand and synthesise materials so as to form their own opinions, make new connections, and apply their learning in new situations.

*“That they’ve thought, and formed some opinions about a specific topic. That they have been able to synthesise the material that they’ve found related to*

*that topic into some kind of conceptual framework. That they can then explain.” (E1)*

*“Yeah, they got it straight away. A few of them went and wrote code, and counted the number of operations, to verify for themselves, but they really seemed to get it. ... At the end of workshop they were telling me ‘We saw the different complexities of sorting algorithms’. That’s how you know it works.” (M1)*

*“[The students] can pull it apart, and then say ‘yes, I can do that’, and they can take it to a parallel but similar situation and implement it. It’s not something that they could just copy mine and paste in; it’s something they would have to design themselves, because it’s only a similar environment, and I think, you know, to get that sort of thing up and running would be highly effective; they really would have a skill that would be worthwhile, and it had been done at a fairly modest cost... A very valuable skill has been developed, and they can then apply it to other, similar, environments in their – when they get out working.” (E3)*

In this view, student engagement, synthesis and extrapolation of knowledge do not stand alone; they start with effective delivery of materials:

*“So they would have been given three things by the time they get out of the lecture: they would have been given a basic philosophy, they would have been given details of how you implement that philosophy in a particular set of tools, and they would have been given a demonstration on how to do that, and they would also have the backup notes with the screen shots. I would then expect them to be able to transfer that knowledge, to be able to apply it to a parallel but similar situation, which is their main assignment.” (E3)*

This third conception encompasses the two previous ones while adding one further aspect. Here teachers also put value on the extent to which student thinking is perceived to have developed, and whether they can see the students applying and synthesising their newly acquired knowledge and skills.

### 4.4 Relationship to Prior Work

There is a strong correspondence between Fox’s teacher-oriented transfer and shaping theories and our category of successful teaching perceived as delivery. Well-organised lectures clearly fall into the transfer category, but teacher observations on well chosen real-world examples and demonstrations reveal a desire to go further and shape student understanding.

Fox’s travelling and growing conceptions are both included in our category of success as developing student thinking. Here the focus is clearly on the student: teachers experience success when students learn independently or synthesise and apply learning to new situations.

Although there is no correspondence between Fox’s theories of teaching and our category of success

experienced as a perception, our other categories robustly support similar analyses reported in the literature.

## 5 Group 3: Motivation

A third group analysed the transcripts for academics' understandings of motivating the students.

The psychological literature on motivation is massive, so why study motivation again using phenomenography? A phenomenographic analysis of teacher's conceptions of motivation complements the psychological literature. Psychology is concerned with an objective reality of engagement and motivation, whereas a phenomenographic study is concerned with teachers' subjective experience of it.

Unlike the results from other groups, the four understandings identified were constructed with Fox's categories in mind. The four understandings are:

- Transfer: motivation as coming from the lecturer
- Shaping: motivation as something to be developed within the student
- Travelling: motivation as something determined by the journey's path
- Growing: motivation as something to be cultivated within and by the student

These four categories are discussed in greater detail in the following four subsections.

### 5.1 Transfer: motivation as coming from the lecturer

Some interviewees discussed how the lecturer's level of engagement affects student engagement:

*"... obviously having a personal relationship with that topic helps [in teaching]. So if you're digging into your own experiences and coming up with good examples, then I think that helps." (T2)*

*"The [lectures] that are more effective are the ones that I'm interested in. ..." (E1)*

### 5.2 Shaping: motivation as something to be developed within the student

In this understanding, motivation forms in the student, but under the clear guidance of the teacher.

*"I don't stick to a rigid structure. I go in and I play by the ear even though I might want to achieve. I give them a weekly schedule at the start of the semester and it is quite a fluid one in the sense that only gives them a guide as to how we are going to progress throughout the semester. But if I feel that students need to spend more time in understanding certain concepts in the topics, I will then do it." (F1)*

*"I find that those assignments that use games and animations and so on, with the proper specs built into the assignment, the students are actually more motivated to finish off the assignment ... that also gives room for creativity and imagination, because*

*they may have animation, they may have, you know, colours, they may have multimedia or included in part of submission." (F1)*

### 5.3 Travelling: motivation as something determined by the journey's path

Understanding Fox's travelling as a journey through the countryside of discipline knowledge, motivation can be found in the use of local distractions that highlight aspects of the landscape.

*"It seems the most effective lecture I had this last semester is when I walked in and almost off the top of my head designed a database that would store information about what was then the ashes series going on in England in the cricket – England versus Australia. ... I talked about constructing a database to store the results. It was remarkable how some of them really lit up, and enjoyed that example..." (N1)*

### 5.4 Growing: motivation as something to be cultivated within the student

In this excerpt, motivation begins in the shaping category, then shifts to growing as the students begin to motivate themselves.

*"They would come back and 20% of them would have done it and some of them didn't do it because they didn't understand it and the rest didn't do it because they just weren't motivated. I found that where we worked together on the board they actually got into it – they saw that ah this isn't so bad after all – and once they started seeing if we took them through some ... you work on a simple problem then you add just a little bit to the next one and a little bit to the next one they go to the point where they liked to achieve and they could [see] that they could do it, and they were almost getting competitive." (I2)*

### 5.5 Relationship to previous work

As discussed above, these four understandings were formed with Fox's personal theories in mind. Some phenomenographers suggest that the search for categories should be carried out with absolutely no preconceptions, but this would appear to be all but impossible for a researcher who is familiar with the area being studied. We take this result to suggest that our transcripts can be seen as supporting the long-established categories.

## 6 Group 4: Granularity of focus

This group did not look at a specific aspect of IT teaching but instead at the broad perspective of the teachers' understandings of their teaching. The categories of description that emerged define the focus that educators have for their teaching work:

- Focus on subject
- Focus on course
- Focus on employment at graduation

- Focus on career
- Focus on life and society

These five categories are discussed in greater detail in the following five subsections.

Each category describes a framework that scopes and directs how the teachers plan, design, and implement their teaching program and activities. A fine-grained focus, on individual subjects, tends to correspond with short-term goals for students, while a coarse-grained view, on the rest of the students' lives, looks at the far longer-term goal of developing lifelong learning capability.

### 6.1 Focus on subject

When understanding IT education as focusing on the subject, the educator is concerned primarily with teaching content. The educator's focus is directed towards helping the students gain the skills and knowledge to complete the subject successfully.

*"The aim of the subject is to get the student to understand, as the title says, Web technology ..."* (D1)

### 6.2 Focus on course

When understanding IT education as focusing on the course (the full program of study leading to a qualification), the educator is aware of the content of other subjects within the course. The educator's focus is on ensuring that students gain the necessary knowledge and skills to progress through the course. The educator is also sensitive to what students may have studied or are concurrently studying in other subjects within the course.

*"... there are a number of outcomes ... the immediate outcome is that they can succeed and progress with the following [subject] because this is*

*a prerequisite for a number of [subjects] ..."* (O1)

### 6.3 Focus on employment at graduation

When understanding IT education as focusing on employment, the educator is concerned with preparing students for the workforce. The focus is on ensuring that students are able to gain employment in computing and to work effectively immediately after graduation. The focus here is on the presentation of content that is current and relevant. This also involves designing activities to develop skills that will be useful to students in the workplace.

*"Knowing the features of different languages – the faults and good bits – means that when they are out in the workplace and they have to choose a language for a project then they are better equipped to choose the language, the best language for the job."* (L1)

### 6.4 Focus on career

When understanding IT education as focusing on the career, the educator endeavours to prepare students for lifelong work in a range of careers and workplaces. The focus is on helping students develop attitudes and approaches to learning that will help them evolve and adapt as needed throughout their working life. The educator is concerned with linking subject content to the issues the students will encounter in their future employment.

*"... you have to learn to adapt to continuous change and so we do not know at the moment what will happen after a few years on ..."* (K1)

### 6.5 Focus on life and society

When understanding IT education as focusing on life and society, the educator is concerned that students understand the role of the IT profession in society and

	Fox (1983)	Practicals (sec 3)	Success (sec 4)	Motivation (sec 5)	Focus (sec 6)
<b>Content- or teacher-centred</b>  Positivist "Sage on the stage"	<b>Transfer:</b> teacher-driven learning with focus on content	Acquire and practice skills	Delivery	Coming from lecturer	Fine grain, short-term goals
	<b>Shaping:</b> teacher-driven learning with focus on student change	Reinforce learning from lectures and/or textbook		Developed within student under lecturer's guidance	
<b>Student-centred</b>  Constructivist "Guide on the side"	<b>Travelling:</b> student-driven learning with focus on content	Refine and troubleshoot acquired skills	Develop student thinking	Determined by the journey	Coarse grain, long-term goals
	<b>Growing:</b> student-driven learning with focus on student change	Apply skills acquired elsewhere		Cultivated within student by student	

**Table 1: The results of four groups compared with Fox's personal theories of teaching**

consider how the technology influences various aspects of life.

*“Telecommunications are nowadays more and more everyday life for students ... I think it is quite essential that they understand those problems and chances that are included in telecommunications.”*  
(K1)

## 6.6 Relationship to Prior Work

We are not aware of any prior research that has considered this question of granularity of the academic's focus. We do find a possible link with Fox's broader categories: a fine-grained focus and a short-term teaching goal appear to correspond with teacher- or content-centred categories, while the coarser-grained focus and longer-term goals appear to correspond with the student-centred categories.

## 7 Conclusion

There has been some earlier phenomenographic research on the experience of teachers within various disciplines, but the experience of computing teachers has until now remained relatively unexplored.

We have conducted broad-ranging interviews with 25 computing academics to elicit the different ways that they understand their teaching. Our preliminary analysis of the transcripts took place at a workshop to introduce computing academics to phenomenography, and in the following weeks the four groups from the workshop continued their analysis, guided by the workshop's leaders.

For the most part unaware of Fox's personal theories of teaching, most groups produced results that are reasonably consonant with those theories (see Table 1). At the same time, the results include some observations that do not appear to have appeared in prior work, such as the question of granularity of an academic's focus and the corresponding time span of the academic's goals for the students.

This project contributes to the broad educational community by illustrating how educational research can be fertilised by the questions and results of non-educationalists. Results from computing education become accessible to the wider community, including educationalists. In addition, the study further confirms Prosser's suggestion that analysis of this sort is highly context-dependent, and adds one more discipline to the tally of those studied.

The contributions to the computing education community are perhaps more concrete. First, the study offers computing academics an insight into the understandings that underlie their teaching. Second, an awareness of these understandings can help by providing a framework for the analysis of proposed teaching methods and materials.

When debating the construction or revision of a subject or degree, it is our contention that a group of IT academics will function better as a team if they are explicit about the

beliefs and values that lie behind their individual positions. More specifically, they should place their statements into the context of the categories that have emerged from our study. They should ask themselves questions such as the following:

1. Do we want any lab practical classes to be classes in which students
  - acquire and practice skills that are more or less independent of the lectures?
  - reinforce material presented in lectures and/or textbooks?
  - refine and troubleshoot skills that they have acquired elsewhere?
  - apply skills that they have acquired elsewhere?
2. Will we determine the success of this course
  - by the way it feels, either to us or to the students?
  - by the quality of delivery of the teaching material?
  - by evidence of development of student thinking?
3. Do we expect the student motivation for this course
  - to come from the lecturer?
  - to be developed within the student under the lecturer's guidance?
  - to be determined by the student's journey through the subject matter?
  - to be cultivated within and by the student?
4. Are our goals for the students in this course
  - fine grained, focused on the subject or the course?
  - coarse grained, focused on the student's career prospects or role in society?

We do not argue, as one might when studying student understanding of a topic in computing, that the higher-level categories are necessarily the best or the most appropriate for all circumstances. We suggest only that academics who are aware of the range of understandings will be better able to decide how to design a revision or a new offering.

We believe that our work demonstrates that research of this type can bring valuable insights. We hope that others will follow us in applying this research approach to problems in the teaching and learning of computing.

## 8 Acknowledgments

This study was supported by a Special Projects Grant from the ACM Special Interest Group in Computer Science Education (SIGCSE). The authors thank the Department of Computer Science, University of Tasmania, and Nicole Herbert in particular, for hosting the PhICER workshop.

## 9 References

- Åkerlind, G (2005): Phenomenographic methods: A case illustration. In J. Bowden & P. Green (Eds.), *Doing Developmental Phenomenography*, 103-127. Melbourne, Australia, RMIT University Press.

- Ben-Ari, M (2001): Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching* **20**(1):45-73.
- Berglund, A (2005): Learning computer systems in a distributed project course: the what, why, how and where. *Acta Universitatis Upsaliensis*, Uppsala, Sweden.
- Booth, S (1992): *Learning to program: a phenomenographic perspective*. PhD thesis, University of Gothenberg, Sweden.
- Booth, S, & Ingerman, A (2002): Making sense of physics in the first year of study. *Learning and Instruction* **12**(5):493-507.
- Bruce, C, Buckingham, L, Hynd, J, McMahon, C, Roggenkamp, M, Stoodley, I (2004): Ways of Experiencing the Act of Learning to Program: A Phenomenographic Study of Introductory Programming Students at University. *Journal of Information Technology Education* **3**:143-159. <http://jite.org/documents/Vol3/v3p143-160-121.pdf> [accessed October 2006]
- Carbone, A, & Kaasbøll, J (1998): A survey of methods used to evaluate computer science teaching. *Proc 3rd Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Dublin, Ireland, 41-45, ACM Press, New York.
- Cope, C (2000): Educationally critical aspects of a deep understanding of the concept of an information system. *Proc Fourth Australasian Computing Education Conference (ACE2000)*, Melbourne, Australia, 48-55.
- Dunkin, M (1990): The induction of academic staff to a university: processes and products. *Higher Education* **20**:47-66.
- Eckerdal, A & Berglund, A (2005): What Does it Take to Learn "Programming Thinking"? *Proc 1st International Computing Education Research Workshop (ICER'05)*, Seattle, WA, USA, 135-142. ACM Press, New York.
- Eckerdal, A, & Thuné, M (2005): Novice Java programmers' conceptions of "object" and "class", and variation theory. *Proc 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Caparica, Portugal, 89-93, ACM Press, New York.
- Fox, D (1983): Personal Theories of Teaching. *Studies in Higher Education*, **8**(2):151-163.
- Kutay, C, & Lister, R (2006): Up close and pedagogical: computing academics talk about teaching. *Australian Computer Science Communications* **52**:125-134.
- Leveson, L (2004): Encouraging better learning through better teaching: a study of approaches to teaching in accounting. *Accounting Education* **13**(4):529-548.
- Lister, R, Box, I, Morrison, B, Tenenberg, J, Westbrook, S (2004): The Dimensions of Variation in the Teaching of Data Structures. *Proc 9th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Leeds, UK, 92-96, ACM Press, New York.
- Marton, F (1986): Phenomenography – a research approach to investigating different understandings of reality. *Journal of Thought* **21**:28-49.
- Marton, F, & Booth, S (1997): *Learning and Awareness*. Mahwah, NJ, USA: Lawrence Erlbaum Associates.
- McKenzie, J (2002): Variation and relevance structures for university teachers' learning: Bringing about change in ways of experiencing teaching. *Research and Development in Higher Education* **25**:434-441.
- Northedge, A (1976): Examining our implicit analogies for learning processes. *Programmed Learning and Educational Technology* **13**(4):67-78.
- Pang, M-F & Marton, F (2003): Beyond "lesson study": comparing two ways of facilitating the grasp of some economic concepts. *Instructional Science* **31**(3):175-194.
- Prosser, M, Trigwell, K, Taylor, P (1994): A Phenomenographic Study of Academics' Conceptions of Science Learning and Teaching. *Learning and Instruction* **4**:217-231.
- Ramsden, P (2003): *Learning to teach in higher education* (2nd ed.). London; New York: Routledge.
- Runesson, U (2005): Beyond discourse and interaction. Variation: a critical aspect for teaching and learning mathematics. *Cambridge Journal of Education* **35**(1):69-87.
- Samuelowicz, K, & Bain, J (1992): Conceptions of teaching held by academic teachers. *Higher Education*, **24**:93-111.
- Simon, de Raadt, M, Sutton, K, Venables, A (2006): The unique role of lab practical classes in computing education. *Proc 6th Baltic Sea Conference on Computer Education Research (Koli Calling 2006)*, Koli, Finland.
- Stoodley, I, Christie, R, Bruce, C (2004): Masters Students' Experiences of Learning to Program: An Empirical Model. *Proceedings of QualIT2004: International Conference on Qualitative Research*. <http://sky.fit.qut.edu.au/~bruce/pub/papers/QualIT2004-Bruce.pdf> [accessed October 2006]
- Trigwell, K, Prosser, M, Taylor, P (1994): Qualitative differences in approaches to teaching first year university science. *Higher Education* **27**(1):75-84.

# A Maturity Model for Computing Education

Christof Lutteroth

Andrew Luxton-Reilly

Gillian Dobbie

John Hamer

Department of Computer Science  
The University of Auckland  
38 Princes Street, Auckland 1142, New Zealand  
Email: {lutteroth, andrew, gill, j.hamer}@cs.auckland.ac.nz

## Abstract

We propose a maturity model for computing education which is inspired by the Capability Maturity Model (CMM) used in software engineering. Similar to CMM, the Computing Education Maturity Model (CEMM) can be used to rate educational organisations according to their capability to deliver high-quality education on a five level scale. Furthermore, CEMM can be used in order to improve an institution's capability by implementing the best practises and organisational changes it describes.

*Keywords:* Education, CMM, quality, maturity

## 1 Introduction

In this paper we draw an analogy between process improvement in software development and process improvement in computing education. Teaching and software development have a lot in common. Both are complex activities, both undergo a development life cycle, and we would like both to be of high quality, despite finding this difficult to measure.

In both domains, a main ingredient of success is good structure and the use of best practises, i.e. a process that helps us to structure and do things right. From a teaching perspective, the process of education is even more relevant than a development process for software engineers: a skilled software engineer may neglect good software development practises but still produce a good product at the end. The process by which software is developed is not directly visible in the quality of the end product. Teachers, however, can influence the end product of their work only indirectly. The actual learning outcomes are, ultimately, not up to them but up to the students. All a teacher can do is perform the process of teaching the best he can, and that is why the process is of such an importance: it is the only tool we have.

What exactly do we mean when we speak of a *process* for computing education? In our analogy, software development projects correspond to courses as they are taught, in contrast to a course as an abstract concept. Somehow a course has to be defined: what is taught, when is it taught, how is it taught and what is the measure of success (e.g. exam marks)? All these questions are very familiar to lecturers, and it helps to have most of them answered before the course is taught. Now, the same what, when and how—but of course with a different context—have to be answered

for a software development project, and constitutes what is commonly understood as software development process. Software development is not the only analogy one can draw; a course is similar to other processes as well. However, in the context of computing education it seems particularly appropriate to refer to processes of software engineering.

If processes are so important for the quality of the product—and many disciplines agree in this matter—then we should spend time and effort on improving them. In manufacturing, for example, we find a long tradition of process management and process improvement. This is why meta-processes have been created that aim at improving the quality of the processes themselves. The point that we want to make is that for education, and in particular computing education, quality improvement concepts exist as well and can make a big difference for a teacher's success. In this paper we show how such a meta-process can be created for computing education along the lines of the Capability Maturity Model (CMM), a meta-process for software development.

We acknowledge that many factors affecting educational success are of a human nature: good teachers make a difference, and teaching and learning are greatly influenced by the personal interactions between teachers and students. Nevertheless, the benefits and support provided by a high-quality educational process should not be underestimated. Process can benefit both students and teachers. Furthermore, the maturity model for education which we propose in this paper should not be misunderstood as an attempt to dictate any particular approach to teaching. Our approach is descriptive rather than prescriptive, and the intention is to provide best practises and improvement strategies to support teachers in their work. The aim is essentially the same as for CMM: to narrow the variance of quality by applying common sense process management and quality improvement concepts.

We use CMM as an inspiration but not as a strict guideline. Thus, we are interested in the general insights and concepts, and not in the many details (such as the differences between the different versions of CMM and its successor CMMI). Education is covered in neither CMM nor CMMI, and we believe that the context of academia in contrast to the context of industry requires us to deal with CMM and related material in a constructive yet selective and critical manner. It is not possible to create a maturity model for education by strict analogy.

Section 2 motivates our maturity model. We present an overview of CMM in Section 3 before describing our approach in Section 4. Section 5 revisits a model of major factors in software engineering and transfers it to the domain of education. Section 6 provides an analysis and discussion of our work. In Section 7 we discuss related work, and Section 8 describes further work we intend to do on this topic.

Our conclusions are offered in Section 9.

## 2 Motivation

Reflective practise plays an essential role in improving an individual's professional activities (Schön 1987). As educators, we use reflective practise to examine our own teaching, and refine the approaches we use in the classroom. Harris (1998) reports that "effective teaching is linked to reflection, enquiry and continuous professional development and growth."

As professional educators, we should reflect on our processes and practise at a variety of levels. A teaching portfolio can be an excellent aid to reflective practise at an individual level. Course portfolios play a similar role for courses, assisting us to engage in reflective practise about courses. However, there is no comparable tool used to reflect upon process and practise at a departmental level.

### 2.1 Individual reflection

Teaching computing is difficult. A large (and growing) body of research shows that individual staff are trying to improve student outcomes, but with limited success (Robins et al. 2003). Although those interested in CS Education may have developed more formal processes to evaluate and improve their own teaching performance, for most staff the cycle of teaching occurs on a more *ad hoc* basis.

Teaching staff generally have few processes in place to analyse and improve teaching performance. Student evaluations are reasonably common, but other formal processes are rare. One such process is the maintenance and evaluation of a teaching portfolio.

A teaching portfolio is an important tool used by reflective educators. It provides a means of exhibiting professional proficiency (i.e. it can be used for summative feedback), but is also frequently used to afford insight into an individual's professional practise (i.e. it can also be used for formative feedback) (Seldin 2004). The reflective practise exemplified by teaching portfolios is clearly focused at an individual level.

### 2.2 Course reflection

Although reflection on individual teaching practise plays an important role in the scholarship of teaching, reflection on the course itself is also valuable. A course portfolio can be used as a tool to reflect on a course and guide future improvement. Cerbin (1994) describes a course portfolio as consisting of four core elements:

1. a teaching statement;
2. an analysis of student learning;
3. an analysis of student feedback; and
4. a course summary.

Course portfolios are intended to form part of an ongoing process that critically analyses the course in detail. The reflection required for a course portfolio overlaps with that of a teaching portfolio, and the development of one portfolio can provide leverage for effective reflection on the other.

*Ideally instructors should reflect upon every assignment, chapter, or lab experiment in their course. In reality, instructors may write down their personal reflections only once a week or for every major section of the course.* (Reeves et al. 1998)

Our experience indicates that course portfolios are not as widely used as teaching portfolios, and critical reflection at the level of courses is not as common as individual reflection.

### 2.3 Departmental reflection

The problems faced by computing educators are compounded when department processes fail to provide adequate support. Although reflective practise is often demonstrated at an individual level, and occasionally directed at courses, it is rare to see any critical reflection on departmental processes, although such processes have significant impact on both courses and individuals.

There is a need for a framework within which we can position current teaching practise at both an individual and departmental level. Such a framework would encourage educators to examine their own processes more critically and might suggest avenues for improvement at a strategic level.

Using a model of process maturity would act to support critical reflection at both an individual and departmental level. The reflection engendered by process analysis would complement the development of teaching portfolios and course portfolios. We intend to show here how the Capability Maturity Model used to describe software development processes can be adapted for use in the education domain, providing a framework for positioning current academic processes and reflecting upon current practise.

## 3 The Capability Maturity Model (CMM)

The Capability Maturity Model (CMM) (Institute 1995) was originally developed in the 1980s by the U.S. Department of Defence Software Engineering Institute (SEI) at Carnegie Mellon University as a method for objective evaluation of contractors for military software projects. It has been continuously revised since then.

There are numerous instances of large software systems suffering unexpected cost increases, schedule delays, and even complete failure. As a consequence, the U.S. military and other organisations were looking for a way to rate the reliability of the software development work a contractor could offer. The original CMM and its successors were, and are still, used for many government projects.

The idea behind CMM is that a high-quality process yields a high-quality product at the end. As a consequence, CMM aims at providing objective measures for the quality of software development processes and strategies for their improvement. CMM tries to define the key elements of an effective process and outlines how to improve suboptimal processes, i.e. the evolution from an "immature" process to a "mature, disciplined" one. It describes key practises for meeting goals for cost, schedule, functionality, and product quality. The CMM standard is relatively heavy-weight, being several hundred pages strong.

CMM ranks software developing organisations according to a hierarchy of five maturity levels, with the first being the least mature and the fifth being the most mature. The five levels are: *initial*, *repeatable*, *defined*, *managed*, and *optimising*. Each maturity level defines a certain capability of producing quality software, key process areas (KPAs) that state what is done in order to achieve the respective level of quality, and key practises which specify how it is done. A software developing organisation ranked at a certain maturity level can improve over time and reach the next level of maturity. However, a new level has to be well established before the next level can



be achieved, so that it is not possible to skip levels. This is because each level builds on the preceding ones and adds features to the process rather than replacing them.

In 1997 development of CMM was halted in favour of its successor, Capability Maturity Model Integration (CMMI) (Chrissis et al. 2003). CMMI provides a structured view of process improvement across an organisation, i.e. not just the organisational parts concerned with software development. It provides models for four different disciplines—Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing—and is intended to provide a framework for the integration of other models that might emerge. It further aims at creating appraisal and training products for process quality. For our education maturity model we solely refer to the much better known CMM, which provides all the necessary core concepts and ideas. In the following we will describe all the five maturity levels and summarise some of the criticism of CMM.

### 3.1 Initial development process

This level is the lowest possible and (tragically) the level most software developing companies fall into. It is also known as “ad hoc,” “chaotic” or “heroic” level. It refers to a process which is informal and poorly controlled, and thus “chaotic.” An organisation of this level does not provide a stable environment for developing and maintaining software, so constant changes of the process make it unpredictable, i.e. “ad hoc.” The organisation’s performance relies on the capabilities of individuals (“heroes”), who may do or not do their work well. Thus, the performance varies greatly with their innate skills, knowledge, and motivation. This all leads to unpredictable cost, schedule, functionality, and product quality.

### 3.2 Repeatable development process

Level two refers to an organisation in which a good performance is repeatable. A project management system is in place, and planning and management of new projects is based on experience with similar earlier ones. Thus, successful practises from those earlier projects can be repeated. Such an organisation has established policies for managing a software project and procedures to implement those policies, i.e. effective management processes for software projects are institutionalised. Key process areas of this level are management activities like requirements management, project planning, project tracking and oversight, quality assurance, and configuration management.

### 3.3 Defined development process

On level three, the process used in an organisation is standardised and documented. The organisation uses effective management as well as effective software engineering practises, and software engineering and management processes are integrated. The process is characterised and fairly well understood. Organisations on this level have formed a dedicated Software Engineering Process Group (SEPG) that takes care of all the process-related activities, i.e. process definition, adaption and development. Furthermore, such an organisation provides a training program about the process so that everybody can acquire the knowledge and skills required to fulfil the roles the process assigns to them. The standard process of an organisation can be tailored to the unique characteristics of a project, and the result of this adaption is called the project’s defined software process.

In summary, this level adds engineering processes and organisational support for process management. Key process areas include: process focus, process definition, training program, integrated software management, software product engineering, intergroup coordination, and peer reviews.

### 3.4 Managed development process

On level four the products as well as the process are quantitatively controlled. This means that the process is instrumented with well-defined and consistent measurements, and there exist quantitative quality goals for both the software products and the process. An organisational measurement program exists that measures productivity and quality for all the important activities, and the surveyed data is collected in an organisation-wide software process database. Processes on this level are predictable because the process is measured and controlled so that it operates within measurable, well-defined limits. Thus, we ideally achieve a predictably high quality. The focus of this level is on product and process quality, and key process areas are quantitative process management and software quality management.

### 3.5 Optimising development process

On level five process improvement is institutionalised. The whole organisation is focused on continuous process improvement. The collected data on the effectiveness of a process is used to analyse the cost benefit of new technologies and proposed process changes. Furthermore, data is analysed for causes of defects, so that known types of defects can be prevented from recurring. Weaknesses are identified and respective improvements are undertaken proactively. Communication within the company ensures that innovation, once identified, is disseminated, and experience is exchanged also between projects. Key process areas are defect prevention, technology change management, and process change management.

### 3.6 Criticism

CMM has received a degree of criticism over the years. First of all, one has to observe that CMM has failed to take over the world, and commercially there are more successful methodologies, e.g. IBM’s Rational Unified Process (Kruchten 2001), which try to provide a framework and guidelines for high-quality software development. CMM is neither a recipe nor guarantee for success: an organisation operating on level one may be more successful than one operating on a higher level, although this is considered less likely, especially for larger organisations. There is little validation of the cost savings provided by CMM below the fourth level since this is where quantitative measurement starts, and unfortunately there are only very few organisations on this level. The vast majority of software developing organisations is considered to be still on level one.

Many critics accuse CMM of having too much bureaucratic overhead, and it is therefore often thought to be only suited for organisations that exhibit a high degree of bureaucracy anyway, such as government agencies or large corporations. CMM may influence an organisation to focus on perfectly completed paperwork rather than on productive tasks like application development or sensitivity to client needs and the market. A highly-regulated process may stand in the way when entering a market with some kind of product is more important than functionality and high quality. The main criticism objects that CMM

promotes process over substance, i.e. predictability over the actual service provided to end users.

#### 4 The Computing Education Maturity Model (CEMM)

CMM is used to rate the maturity of software development organisations. Our proposed Computing Education Maturity Model (CEMM) aims to do the same for educational organisations. The basic building block of CEMM, however, is the course. This is possible because courses, as an abstract entity in an educational organisation's curriculum, are (ideally) well-specified and (usually) adhere to fixed time and cost constraints. Whenever the course is held, these invariants will typically stay the same. Consequently, the course unit is a stable enough concept to be rated. In contrast, software development projects are more often unique events without such invariants and thus the only stable concept to be rated in CMM remains the organisation itself.

Like CMM, CEMM rates an education process according to a hierarchy of five levels. We retain the names of the levels, and the generic concepts and ideas of each level are essentially the same, transferring them into the new domain. On the course level, the absence or presence of a process for a course and its characteristics determine the maturity level of that course. On a higher organisational level, maturity is determined by the maturity of individual courses and the way in which all those processes affecting the quality of the organisation are standardised and integrated. In the following sections we will discuss the different maturity levels.

##### 4.1 Initial education process

The initial level of maturity is like the one in CMM: informal and poorly controlled. This means that the process is neither understood nor reflected upon. A typical cause for this are changes of requirements, as would be the case for a new course that has never been held before, or for a lecturer who decides to change large parts or all of a course. Such drastic changes invariably lead to instability and frequently result in a rather chaotic and stressful environment. Course preparation and material is made on demand, i.e. *ad-hoc*, and because of the instability there is hardly any reviewing at all. Time pressure also plays a role here, with the last lecture slides often being created the hour before the lecture starts. Unsurprisingly, this results in a lot of mistakes in material like handouts and slides. For reflective activities like evaluation of feedback or grades there is usually no time left. The reuse of the created material is low and constrained to the personal teaching activity of the lecturer. Like in CMM, the success of courses depends on the skills and initiative of "hero lecturers," who are able to perform despite the lack of structural support.

##### 4.2 Repeatable education process

On level two, courses can be taught again as they were taught before, and previous successes can be repeated. Planning and implementation of courses are based on previous experience, and the plan for a course is used to track its progress. The educational organisation has established policies for important aspects concerning the management of courses, and procedures to implement those policies. For example, there should be policies about helping students to catch up with missed lectures, on how students can seek assistance, and how cheating is dealt with. Courses have well-defined prerequisites, requirements

and intended outcomes that fit well with the other courses offered. Course materials like slides, assignments, lab sheets and exam questions exist and can be reused not only by the lecturer who created them but also by others. A transition is made from personal resources to resources that can be shared with others. Changes to course material are controlled; i.e. checks are made to ensure they are in harmony with the course plan and other constraints, and managed with a version control system.

##### 4.3 Defined education process

On level three, the course concepts and materials have been improved in several iterations, so that they have reached a certain level of stability. Courses are well-documented and come with a rich collection of material, for the students as well as for the lecturers. There exist scripts or textbooks for all the topics covered in a course, which potentially give students the possibility of achieving the aims of a course solely by guided self-study. There exist sets of slides and other materials that may be appropriate for a course, such as source code examples. Students can find learning material and important course information on a well-defined location on the Internet. If material cannot be made available online (e.g. textbooks), references and instructions are provided on how it can be acquired by students. For the lecturers there exist collections of proven lab exercises, assignments and exam questions. Material is of high quality and can be reused by different lecturers. It is accessible in a shared repository which is an official organisational resource, and maintained collectively by all lecturers. The quality of the material is ensured by the systematic application of reviewing techniques, e.g. peer reviews. Lecturers teaching (or having taught) the same or related courses communicate and work together and organise themselves in working groups.

##### 4.4 Managed education process

On level four, the educational organisation has established a measurement program. Data about each course—assignment, exam and teaching evaluation results—are collected and stored in a central database. The data is fine-grained and not unnecessarily aggregated, e.g. the data set will contain scores for the individual items of a test. Statistical methods are applied on a regular basis in order to verify course practises and manage quality, e.g. controls check whether the variance of grades is within a certain range. Intervention policies are defined and triggered when controlled quality parameters do not fall within well-defined acceptable limits. For example, a lecturer may contact a student who fails an assignment, to propose a meeting and offer advice.

##### 4.5 Optimising education process

On level five, changes to the process are carefully managed and reflect best practice as informed by the community of scholars engaged in education research. Qualitative and quantitative research methods are used to gain insight into the process and lead to improvements in the process itself.

Statistical methods are used in order to control and change the process itself. Changes of the process are carefully managed, i.e. identification and implementation of process improvements is institutionalised. This may be, for example, in the form of process improvement meetings at the end of each course iteration. Statistical methods can be of great help in, for example, the selection of exam questions and

analysis of course results. Consequently, improvement meetings need to discuss descriptive statistics that illustrate the characteristics of the collected data effectively. Defects have to be identified and analysed, so that people get to know what went wrong and why. Knowledge about defects is used to prevent recurrences, and knowledge about weaknesses used for proactive process improvement.

## 5 Sneed's Square revisited

Every industrial software project has some key factors that have to be balanced against each other by means of careful project management. Sneed's "devil's square," shown in Figure 1, illustrates this balancing act with a simple model of how these factors influence each other.

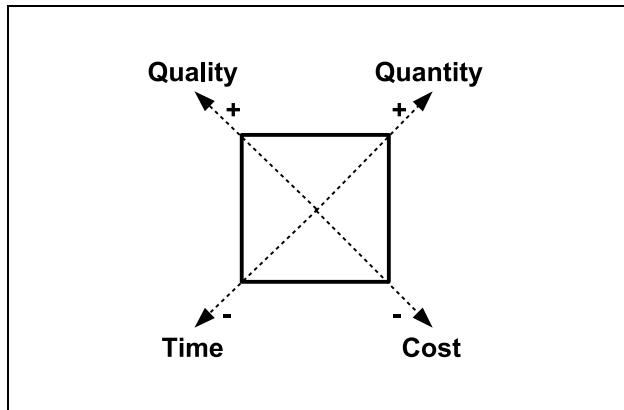


Figure 1: Sneed's "devil's square"

Four factors are considered:

1. the *quality* of the software that is developed;
2. the *quantity* of the software, i.e. the amount of functionality covered by it;
3. the *time* required to develop the software; and
4. the development *cost*.

In the figure these factors are represented as axes pointing outward. On the top two axes the outward orientation represents positive change, and on the bottom two the outward orientation represents negative change. The values for each factor are given by a square, with the area of the square (or, for better illustration, the circumference) being the productivity of the developing organisation. The productivity is considered invariant over the project duration. The square can be thought of as a thread that is looped around movable pins on the axes. If we want to increase the quality or quantity of the developed software, then the pins on the quality or quantity axes are pulled outward. Since the pins are constrained by the thread, which has a constant length, this will inevitably pull the pins for time and cost into the square: if we want to increase quantity or quality, then development will take longer and/or cost more. Hence the devil's square models the antagonistic effects of the four factors.

When teaching a course we face a balancing act similar to that of the devil's square, and, as in software development, changes to the different factors have to be managed carefully in order to preserve the structural integrity of the course. We illustrate this by means of a similar model, the education square, which is depicted in Figure 2. The education square models four factors:

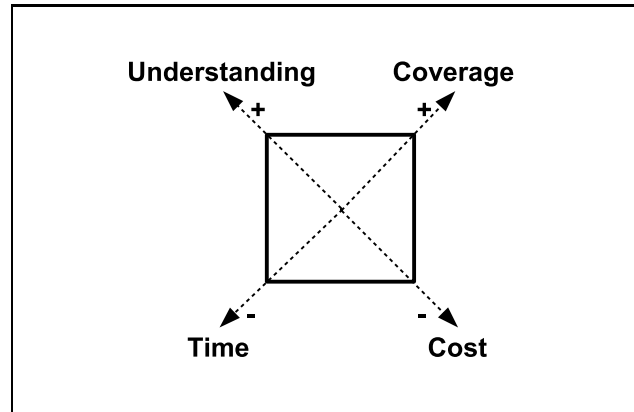


Figure 2: Education square (similar to Sneed's "devil's square")

1. the breadth of *coverage* of important topics in a course (quantity);
2. the students' depth of *understanding* of the taught subject in the course (quality);
3. the *time* required to develop the material for the course, teach the course and for students to learn the material; and
4. the *cost* of the course for both the lecturers and students.

The mechanism of the model is the same as the devil's square. The area within the square is the productivity of the department. In order to increase breadth of coverage of material and depth of understanding, with the same productivity, there must be more time and/or more cost. If it is possible to increase the productivity of a department, perhaps when moving up maturity levels, it is possible to increase the breadth of coverage and depth of understanding without increasing the time and cost input.

What the education square does show us is that in order to improve the maturity of a department we have to use the experiences of each course iteration to incrementally change this balance for the better. Such changes, however, should not be made thoughtlessly but have to be controlled and managed well.

## 6 Analysis and discussion

In this section we address these questions:

- Will CEMM defined for computing education suffer from the same shortcomings as the CMM used in the software industry?
- How will the CEMM be used in practise?
- What is the cost of introducing and maintaining CEMM at Department level?

### 6.1 Possible shortcomings in CEMM

The main criticism levelled at the CMM is the overhead in following the process. One overhead is documentation preparation, and another the measurements that must be taken.

In computing education, it is recognised as good practise to handout a course outline at the beginning of a course and submit a course audit at the end of the course. The course outline enables students to gain an understanding of what the course is about and what is expected from them. Completing the course audit allows the lecturer to reflect on what

they have done in the course, and document the success of teaching techniques they used in the course. The latter information is useful to anyone who takes on lecturing the course the following year or anyone who intends to use the same techniques in their own courses. Course material such as slides, course handbooks, exams and tests, and text books, which are already designed for and used for courses would be included in the documentation. In a well organised department, all this material is kept in a content management system. The documentation required in the CEMM fits naturally with the education process as it is already undertaken in many computing departments.

Measurement in the software industry is fraught with problems. For example, how do we measure productivity, how do we measure the size of a system, and how do we compare the measurements. In addition to this, measurement might not be accepted by employees as they may have fears that the surveyed data might be used against them. In computing education, there are measurements that we deal with on a regular basis. We grade students, we have incoming and outgoing GPAs, we have pass rates. Collecting these measurements are a basic requirement in computing education. These measurements can be used in CEMM.

## 6.2 CEMM in practise

What are the benefits to individuals and organisations from using CEMM?

Some lecturers and departments operate at level one. Individual lecturers prepare new material each year a course is taught and do not reflect on or learn lessons from previous years. At the department level, teaching is distributed on an ad-hoc basis with no reflection on previous years.

In our experience, individual lecturers and departments are more likely to be operating at level two. Lecturers place course outlines, lectures, assignments and lab specifications on the web. This resource is available for use the following year. Departments ask lecturers what courses they would like to lecture and typically the same person will lecture the same material each year.

What is required to lift individuals and departments to level three? A process by which the material is collected in a repository in a format such that it could be used by anyone in the department. The resource would need to be seen to be “owned” by the department rather than by the individual lecturer.

In order for individuals and departments to move to level four, it is necessary to measure and collect the measurements. Some of this may be going on now, but to be at this level it is necessary to have a process that collects this information. Many Universities currently have the tools in place to collect this information, but they often do not utilise these tools to their full potential.

To get to level five there must be a process that uses this information for improving the teaching within the department, and the promotion of departmental reflection.

There are small gains to be made for the individual lecturer. They gain from:

- reusing material that they prepare;
- being able to measure and document the improvement in their teaching practise;
- maintaining statistics that can be used when applying for tenure, grants and promotion;

- sharing of practise with other lecturers in the department.

There are larger gains to be made for the departments. They gain from:

- reusing material and best practise between courses;
- being able to measure and document the improvement in their teaching practise;
- maintaining statistics that can be used at faculty level;
- sharing of best practise across the department.

## 6.3 Cost of CEMM

What is the cost of moving from level to level for a department?

The cost is mainly one of cultural change. Computing education has advanced a long way over the past twenty years, from chalk-and-talk through overhead transparencies to the web. As we pointed out above, much of the documentation that is needed is already prepared and many of the measurements are already taken, so the implementation of the process is simply one of using the material that is available in the management and maintenance of the process.

However, in many departments there is still a feeling of ownership with respect to not only the material prepared for courses but also for courses themselves. Convincing staff to give up this ownership to the department could require quite a large shift.

## 7 Related work

The idea of a CMM for education is not new. For example, Jalote (2003) points out a general strategy for its realisation as a means to overcome the lack of quality standards in the education sector. He addresses the need but says hardly anything about how such a model should look like.

An e-learning Maturity Model (eMM) (Marshall & Mitchell 2004, 2003, 2002) was proposed which is based on the Software Process Improvement and Capability dEtermination (SPICE) model (ISO/IEC 1998). SPICE can be seen as the ISO answer to SEI's CMM, and uses basically the same five levels of maturity plus an additional level zero for processes that are performed incompletely or not at all. It groups process activities into five areas, and eMM defines five corresponding areas for the domain of e-learning. Benchmarks and best practises for e-learning were examined and used in order to compile a set of practises that would fit into the five areas. Unlike CMM, the maturity levels are used for the evaluation of each practise and not just the whole organisation. The result is a framework for e-learning maturity evaluation, which has been applied to the e-learning systems of several New Zealand universities (Marshall & Mitchell 2005). While we believe that the aims of eMM with respect to an educational organisation are similar to the ones of CEMM, its domain is a different one and thus not entirely suitable for our purpose. However, some practises, e.g. from the process areas evaluation and organisation, could be reasonably applied in CEMM as well.

Neuhauser (2004, 2005) presents the Online Course Design Maturity Model (OCDMM), which is also a maturity model for e-learning, based on CMM. This model describes different states of adoption of e-learning technology, which form five maturity levels. The difference between the levels is mainly the degree to which e-learning technology is successfully

used. Like eMM, OCDMM targets specifically the domain of e-learning and is very different from our approach. Only some aspects of OCDMM are of direct relevance to CEMM, e.g. the proposed measures for the success of a course.

Kajko-Mattsson et al. (2001) propose the Corrective Maintenance Maturity Model (CM<sup>3</sup>) for Maintainers Education and Training, a maturity model for workforce development of software maintenance engineers. Their maturity model is based on two educational processes from industry and several generic process models, among them CMM. It defines three levels of maturity: initial, defined and optimal. This maturity model looks at continuing education in an industry environment, which is radically different from that of an educational organisation. In contrast to our domain, education is not the main focus of the context of CM<sup>3</sup>. Therefore, even though both maturity models deal with education, they are hardly comparable.

Frailey & Mason (2002) describe how the Software Engineering Body of Knowledge (SWEBOK) (Abran et al. 2004) was used in order to direct workforce development at a company and curriculum planning at a university. SWEBOK is an attempt of the IEEE to categorise and summarise the knowledge a software engineer should have after four years of practise. The common use of SWEBOK enabled the company to find appropriate university courses for their employees, and the collaboration resulted in the creation of certificate programs that were inspired in their structure by CMM. While this study is not directly related to our work, it shows that standards like SWEBOK and CMM can be useful for educational organisations as well as industry.

People CMM (Curtis et al. 2001) is a maturity model for workforce management and development that was created by the SEI. It addresses the importance of continuing education for the improvement of a software organisation's maturity according to CMM. It describes "best practises" from human resources, knowledge management and organisational development, fitting them into the hierarchy of the five CMM maturity levels. Although the model is tailored to the industry context, its practises for motivation, communication and the realisation of potential synergies within an organisation may inspire similar practises in the educational context. Motivation of students is a major factor for education, as can be communication among students, between students and lecturers, and among lecturers. The practises described in our model rely on collaboration between the staff and, on a higher level, between the organisational units of an organisation. This is a basis on which further synergies may evolve. Fostering a "culture of excellence" is one of the aims of People CMM, and also something that we as educators hope to instill into the students we teach.

To the best of our knowledge, CMM has not been applied to the general domain of teaching computing, with the exception of the work presented here. As discussed above, other maturity models for education either focus on professional development in an industrial context, or on e-learning as a very particular sub-domain. Neither of these approaches are applicable to CS education in general, but may inspire analogous best practises.

## 8 Future work

In this paper we have proposed CEMM. The next step is to validate the proposed model. We will do this by introducing the process into a computing department and collecting data at both the individual

lecturer and department level. Computing departments are technically savvy, and we believe it may be more challenging to introduce an education maturity model to other departments. The next step is to work with another department and analyse whether the CEMM would be a good fit or whether it needs to be changed. Ultimately, we want to undertake a university-wide study, analyse the model at this level and study its effect.

## 9 Conclusion

We described a maturity model for computing education that is based on the well-known Capability Maturity Model created by SEI. Although our approach is new and has not yet been validated, the model incorporates best practises which are either based on common sense or have been successfully applied to other domains with similar motivations. Other best practises are supported by CS education literature, or are founded on our personal experience as lecturers. We find that certain key practises of CMM like rigorous documentation and use of a measurement program seem even better suited for our purpose, as they occur naturally in the context of education. This mitigates the negative impact of bureaucratic overhead on our model, the main criticism about CMM. We plan to continue the research on CEMM and further elaborate and validate this maturity model.

## References

- Abran, A., Bourque, P., Dupuis, R. & Moore, J. (2004), *Guide to the Software Engineering Body of Knowledge-SWEBOK*, IEEE Press.
- Cerbin, W. (1994), 'The course portfolio as a tool for continuous improvement of teaching and learning', *Journal on Excellence in College Teaching* **5**(1), 95–105.
- Chrissis, M., Broekman, B., Shrum, S. & Konrad, M. (2003), *CMMI: Guidelines for Process Integration and Product Improvement*, Addison-Wesley Professional.
- Curtis, B., Hefley, W. E. & Miller, S. A. (2001), *People Capability Maturity Model (P-CMM) Version 2.0*, Addison-Wesley Professional.
- Frailey, D. & Mason, J. (2002), Using SWEBOK for education programs in industry and academia, in 'CSEE&T'02: Proceedings of the 15rd Conference on Software Engineering Education and Training', IEEE Press, pp. 6–10.
- Harris, A. (1998), 'Effective teaching: a review of the literature', *School Leadership & Management* **18**(2), 169–183.
- Institute, S. E. (1995), *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley Professional.
- ISO/IEC (1998), Software process assessment, Technical Report TR 15504:1998, ISOSPICE.
- Jalote, P. (2003), 'Needed: a capability maturity model for engineering education', *The Economic Times India*.
- Kajko-Mattsson, M., Forssander, S. & Olsson, U. (2001), Corrective maintenance maturity model (CM<sup>3</sup>): maintainer's education and training, in 'ICSE'01: Proceedings of the 23rd International Conference on Software Engineering', IEEE Press, pp. 610–619.

- Kruchten, P. (2001), What is the rational unified process?, Technical report, Rational Software Canada.
- Marshall, S. & Mitchell, G. (2002), An e-learning maturity model, *in* 'Proceedings of EDUCAUSE'02: 19th Annual Conference of the Australian Society for Computers in Learning in Tertiary Education'.
- Marshall, S. & Mitchell, G. (2003), Potential indicators of e-learning process capability, *in* 'Proceedings of EDUCAUSE'03: 20th Annual Conference of the Australian Society for Computers in Learning in Tertiary Education'.
- Marshall, S. & Mitchell, G. (2004), Applying SPICE to e-learning: an e-learning maturity model?, *in* 'ACE'04: Proceedings of the Sixth Conference on Australasian Computing Education', Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 185–191.
- Marshall, S. & Mitchell, G. (2005), E-learning process maturity in the New Zealand tertiary sector, *in* 'Proceedings of EDUCAUSE'05: 22th Annual Conference of the Australian Society for Computers in Learning in Tertiary Education'.
- Neuhauser, C. (2004), 'A maturity model: Does it provide a path for online course design?', *The Journal of Interactive Online Learning* **3**(1).
- Neuhauser, C. (2005), A five-step maturity model for on-line course design, *in* 'Proceedings of the 19th Annual Conference on Distance Teaching and Learning'.
- Reeves, J., Hugo, K., Heussner, R., Hala, A., Sarlioghu, B., Bialek, S. & Courter, S. (1998), Course portfolios: A systematic mechanism to document teaching and learning, *in* 'Proceedings of the 28th ASEE/IEEE Frontiers in Education Conference', IEEE, Tempe, AZ.
- Robins, A., Rountree, J. & Rountree, N. (2003), 'Learning and teaching programming: A review and discussion', *Journal of Computer Science Education* **13**(2), 137–172.
- Schön, D. A. (1987), *Educating the reflective practitioner: toward a new design for teaching and learning in the professions*, Jossey-Bass, San Francisco.
- Seldin, P. (2004), *The teaching portfolio: a practical guide to improved performance and promotion/tenure decisions*, 3rd edn, Anker Pub. Co., Bolton, Mass.

# Software engineering class eating its own tail

**Samuel Mann**

Information Technology  
Otago Polytechnic, Dunedin NZ  
smann@tekotago.ac.nz

**Lesley Smith**

Information Technology  
Otago Polytechnic, Dunedin NZ  
lsmith@tekotago.ac.nz

## Abstract

This paper describes an experiment where software engineering students were given the task of developing a project management system for use in capstone projects. The Agile Development Framework, which is a combination of agile and structured methodologies, is introduced. Using examples of student work, the paper describes the effect on learning of this recursive approach to learning software engineering.

**Keywords:** Software engineering projects, computer education

## 1 Introduction

In this paper we describe the results of an experiment in teaching software engineering. By means of a project based teaching approach, students were given the task of developing software for teaching software engineering.

Teaching software engineering at undergraduate level poses the challenge of presenting a robust discipline to students while reflecting industry currency, as software engineering methodologies have been continuously evolving since inception.

In previous papers we describe the ongoing development of an approach to teaching software engineering (Mann and Smith, 2001, 2004, 2006).

A disadvantage of the project based approach is that students focus excessively on learning the subject matter of the project, rather than the software engineering process that is the objective of the course. The research question for this paper is – can we harness this incidental learning by selecting the development of a project management tool as the class project?

### 1.1 Context

Software Engineering is a one semester course in the second year in three year degree in Information Technology. It is a compulsory pre-requisite for the capstone project and as such it gives the students the

tools for undertaking their projects (Mann and Smith, 2004, 2006).

The course is taught using a project based approach, following a strategy of making it real (“real projects for real clients”) and following an “empowerment” approach (Robinson 1994, Smith, Mann and Buissink-Smith 2001). The approach should incorporate a real-life client to mirror an industry experience. It should be flexible (to demonstrate adaptability to change, both in the project and in teaching). It should have a user focus – rather than a plan focussed approach, and it should be applicable over a wide variety of projects, including hardware and network based projects.

Iterations of the course have included a ship safety system, an online motivation project, an animal ethics management system, systems for a maritime museum, a job management system for engineering firms, and a student management system.

A key question in this project based approach to software engineering is the choice of project. In Smith and Mann (2004) we examined our history of such projects and developed a set of guidelines for the selection of the project. In addition to being real, exciting and interesting, we stated that the project should

*“facilitate teaching the structure of the chosen methodology. For early stage developments the client should have an idea of a business problem, but not a solution...facilitate teaching each of the methodologies’ range of tools and techniques. The more creative projects are better for logical design work but are difficult to apply to data modelling”. (Smith and Mann, 2004)*

In Mann and Smith (2005) we investigated the relationship between project methodology, the scale of the project, and student learning. The value of a formal methodology in providing a pathway was demonstrated by a large and complex project:

*“At the onset I had no clue of being able to do what was required, so I didn’t have a preconception about what it would look like. I did not think we could do it and definitely not me. Now I see it can be done...”*  
(Student review)

A potentially huge project, though, can have a disempowering effect, as it was clearly not feasible and students lost interest. Poor groups responded by scoping this project very small, to the extent of developing little more than login systems.

*“When we first looked at the brief for Captain Black we thought that the scope for the project had the potential to be much larger than anything we could confidently develop”* (student review)

At the other end of the scale, a very small project can further shrink as students lose interest as a result of feeling that a formal approach is overkill. In the mid range are projects that turn out to be much bigger than students’ initial understanding.

These examples demonstrate a relationship between the project and the learning of the actual subject: software engineering.

*“We felt disadvantaged, the other groups had members who had worked in business and knew what the system should do”* (student review)

A disadvantage of the project based approach is that students focus excessively on learning the subject matter of the project, rather than the software engineering process that is the objective of the course.

*“We had no idea, none of us knew anything about ships, we spent the first few weeks becoming experts on shipping”* (student review)

In the guidelines for the selection of the projects we argued for real, exciting and interesting, and stated that the project should “facilitate teaching the structure of the chosen methodology. Can this facilitation be achieved by the selection of a project management tool as the class project? In focusing student effort on project development, can we generate a deeper understanding of the methodologies and tools involved?

## 2 Method

Students were given the project of developing a Capstone Project Management System. In a break from our usual practice of using “real” external clients, one of us was the client.

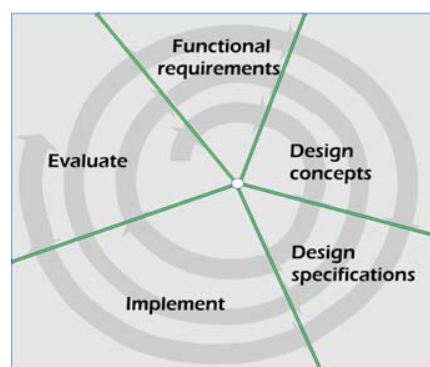
Using a qualitative approach, quotes from student work, reflective journals, etc are given here to identify emergent themes. Two groups (here coded H and C) are used to illustrate contrasting levels of understanding and approaches.

The Capstone projects (and hence Software Engineering) follow an integrated methodology that combines elements of both agile and structured software development. This Agile Development Framework (ADF) approach is described more fully in Mann and Smith (2006).

The focus of the methodology is on the production of robust working systems (software, hardware and maintenance documentation). Planning, comprehensive development documentation and processes are important but are ‘means to an end’ with a focus on content rather than format/representation.

Noble, Marshall, Marshall, & Biddle (2004) note that the shift to an agile approach in industry “has created a need for a similar shift in software engineering education”, explaining that “document centric project methodologies do not align well with students’ reasonable expectations of more agile working methods”.

Each iteration of the development cycle is divided into “sectors” defined by a deliverable output and communication with the client. The sectors here can be seen to form a structured development process (Figure 1).



**Figure 1 Sectors**

The Agile Development Framework comprises three iterations. The first iteration is aimed at building understanding within the development group and stakeholders. The second iteration is aimed at designing and releasing a functional system that meets many of the functional requirements. The third iteration, “robust delivery” is intended to review the success of the second iteration in meeting business requirements, to review functional requirements, and to deliver a robust and stylish “bullet proof” implementation. Each sector is defined by what it produces (Figure 2); the focus is on achieving that outcome. Processes within each sector are determined by agile principles using integrated templates.

The 16 week teaching schedule of software engineering follows the three iterations. In the first four weeks students are introduced to the agile approach and use agile tools to produce a project proposal. The second iteration takes weeks 5-10 and can be considered analogous to a single iteration of a structured approach (with constant reminders of the agile context). Students produce a functional delivery. With little directed structure students take control in the third iteration in working independently as groups to produce a “robust delivery”. This third iteration takes weeks 11-16.



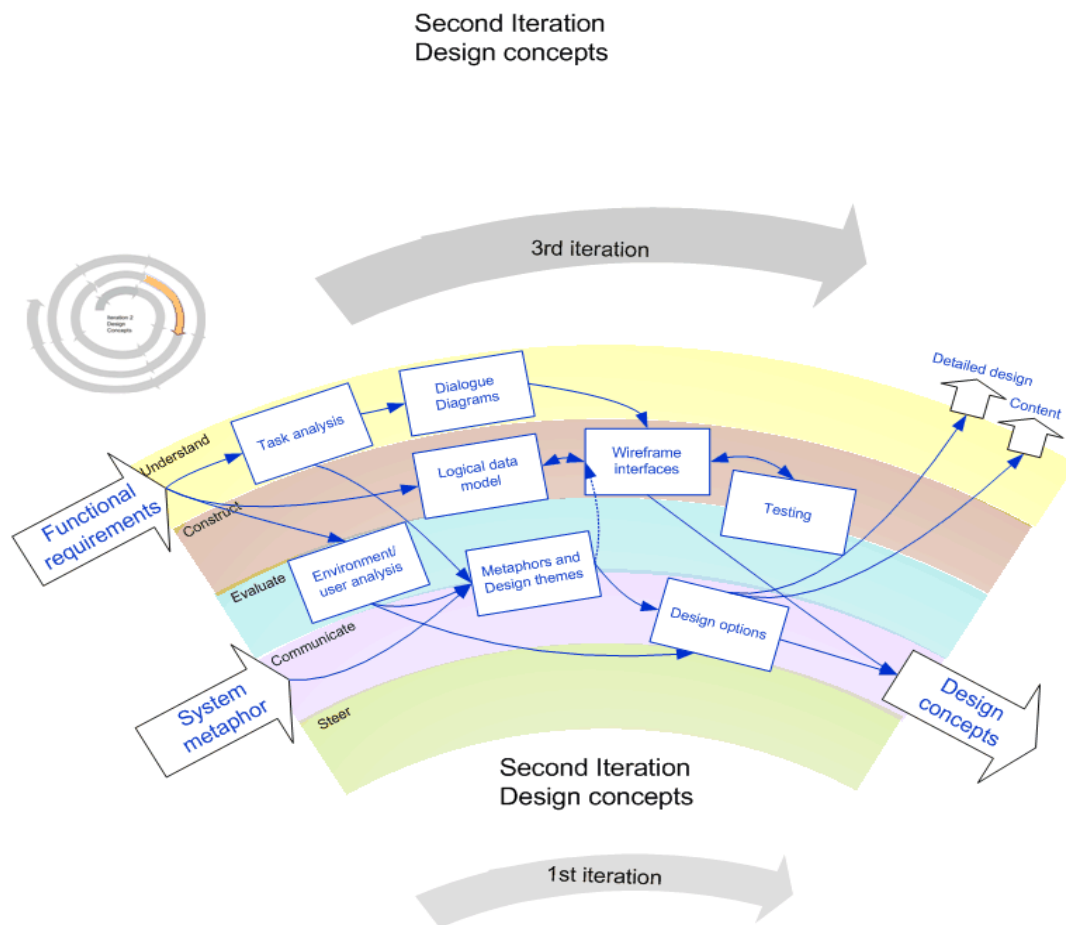


Figure 2: Example sector Design Concepts from 2nd iteration

### 3 Results

#### 3.1 First iteration

In the project the first iteration is scheduled for two weeks. The intention is that after group forming processes, a few days are spent investigating potential technology and building a very rapid prototype. This serves three purposes, first it is a chance for the newly formed group to work together, second, it helps to identify the business problem (is it something that could be solved with something like this), and third, it gives some insight into the complexity of the technology that is likely to be involved.

The project is introduced to students through a client interview. The student notes (Figure 3) illustrate a tight integration of the project and software engineering – it is not really possible to tell to which these notes refer.

The integration had clear benefits for this group:

*“material that was collected during our research met two goals. One was to inform ourselves on the components of the Agile methodology and iterative processes; and two*

*was to find examples of project management software that already existed.”* (Final review, Group H)

However, the limited understanding of the approach was exposed in the client letter from another group:

*“we have decided to develop this system with a combination of the Spiral and Scrum Methodologies”* (Client letter, Group C)

A significant part of the first iteration is an attitudinal change by students. Unlike most courses, where assignment requirements are clearly specified, in software engineering, the groups themselves manage the development of the requirements of the projects. Students often have an assumption that we know the functional requirements and are keeping them hidden. A big part of the learning in the first iteration is of responsibility for development. The agile practices of system metaphor and the planning game are useful in this process.

- *Achieve a system that can facilitate people doing their projects*
- *Shouldn't be a disconnection between project and documentation/management*
- *Make backups regularly, high chance of losing everything*
- *Understand, construct, evaluate, steer and communicate – must remember all five*
- *Group collaboration: need way of allocating work to group members, don't give specific tasks to people, hard to compile at end*
- *System should be supporting agile manifesto*
- *Like to have templates for some things. Ownership of ideas could be a problem*
- *Need a public front end to system, be able to cope with differing amounts of client involvement.*

**Figure 3: Student client interview notes (Group A)**

For this project the system metaphor of a fridge door was extensively explored in class:

*For me the project is a fridge door: always there (you don't have to open it to find information), messages to family (from each other, phone messages), calendar (today, arrangements, upcoming events, shopping list, progress charts, current work, document repository (music tickets, bills to pay), reconfigurable (information held together by fridge magnets), public place, central place: go past it as part of normal life (work)flow (but without it jumping out and interrupting) achievements, photos, newspaper clippings, limited space, information has to be managed to avoid loss in clutter (but this is done without any rules, design or manager) (note, I did write "it does this" here, the door itself of course doesn't do anything except act as a static repository, it is the family who actively manage the information), phone numbers, menu for pizza place, ...and does all these things without getting in way of real job (keeping food cold - our system mustn't get in way of project). (Notes from classroom session)*

Other metaphors explored were: diary, library, mind mapping, job management system (a naïve metaphor).

Some groups used the system metaphor well:

*dashboard metaphor, forward movement, dials for information, everything within reach, doesn't distract from main function (driving car). (First iteration, Group H)*

This group recognised early on that the measure of successful development was “*better project outcomes and streamlined process*” (First iteration, Group H) while most other groups were seeing non-functional metrics such as error counts as the best way of assessing their success.

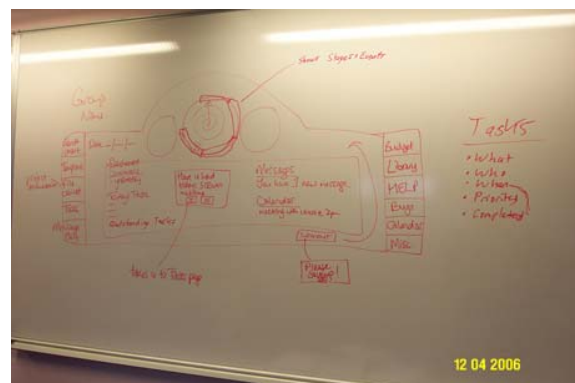
One mature student objected strenuously to the time spent on system metaphor and the planning game “let me just talk to users” and was quite frustrated when we pointed out that she was one herself!

Although useful at the start, the confusion over the project and the methodology soon began to hinder some groups:

*“System Metaphor Description: For this project we will be using the Spiral methodology, incorporating elements of the Iterative Methodologies. The spiral methodology extends the waterfall methodology by introducing prototyping”.*

(First iteration, Group C)

Some weaker students really struggled to get beyond these early stage tasks. They saw them as major pieces of work rather than small exercises aimed at increasing understanding and communication. Although this is common, it was exacerbated by the close links between the project and the course – while previously we could say “forget ships for a while, let’s carry on with software engineering” it was difficult to say “forget software engineering for a while, let’s carry on with software engineering”.



**Figure 4 Group H: Iterations combined with dashboard metaphor**

For the first iteration in software engineering the deliverable can take almost any form. Group H presented a whiteboard diagram of the direction they saw the project taking (Figure 4). In this they extended their dashboard metaphor and incorporated the iterations of the

ADF, but the content area is sparse with “messages, calendar, click here to go to tasks”

Group C finally came to understand the separation of the project and the course and presented a spreadsheet-based approach that aimed to “assist students with a better information management system” (First iteration, Group C). They did not quite understand what the system would do: “reduce extra costs, user friendly, improve data security” (First iteration, Group C) although the prototype spreadsheets to calculate cost of work did show some creativity.

### 3.2 Second iteration

The second iteration ends with the development of a functional system. It starts with an analysis stage where groups develop entity relationship and dataflow diagrams (etc) on their way to describing functional requirements.

It is usual practice for student groups to first write meaningless functional requirements. We had hoped that being closer to the project than usual would lessen this problem. It didn't work; the first functional requirements consisted of variations on: “The system shall store, retrieve and move the project data” (Group C).

We then had the groups analyse the data form and content. This didn't work either, the weaker groups slipped back into their confusion about project and course.

The breakthrough came by getting the groups to write a job description for a person to be employed to facilitate the capstone projects. Suddenly the level of understanding about both the project and course was raised a great deal. This led to questions based around structures of the ADF (Figure 5).

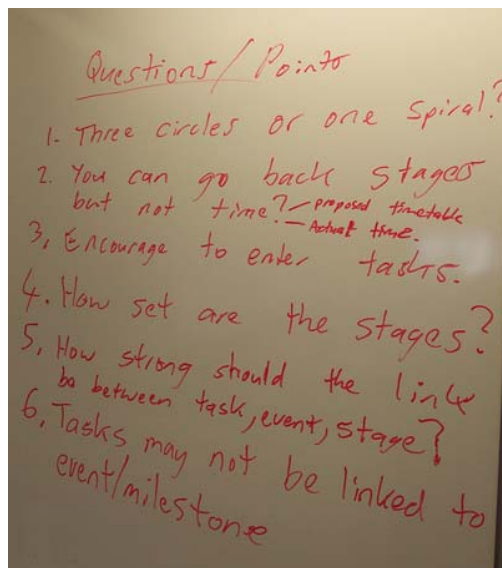


Figure 5: Development questions (Group C)

The functional requirements (admittedly still not perfect) were very much improved:

- Provide process templates for the student's project throughout the three iterations: To guide student throughout the process and allow them to check on the progress of the project.
- Provide a task management function to set up, choose task and check overall progress of tasks and monitor tasks' progress. Students can set up tasks by enter task name and description. Then enter new taskID into Iteration Sector task table
- Enable user to select Iteration and Sector and choose Task To guide student in the process of task management through the use of spiral methodology (Second iteration, Group C)

In the second and third iterations, students are expected to work out their own pathway through each sector. Blank “rainbows” are given out for groups to populate with their own workflow, the only requirement being evidence of a rational flow of information between the inputs and outputs of the sector (Figure 6).

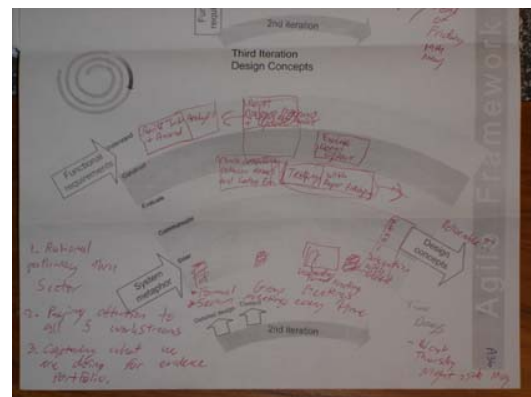


Figure 6: Group working on their own management

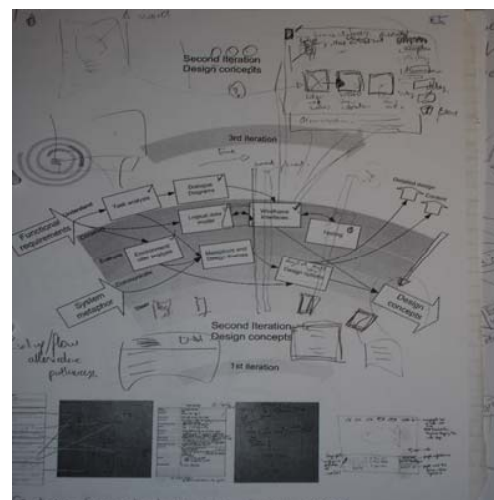
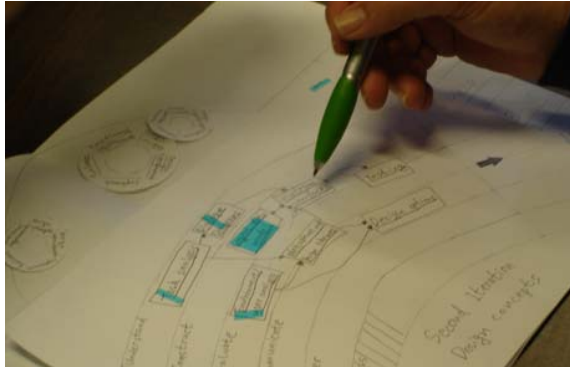
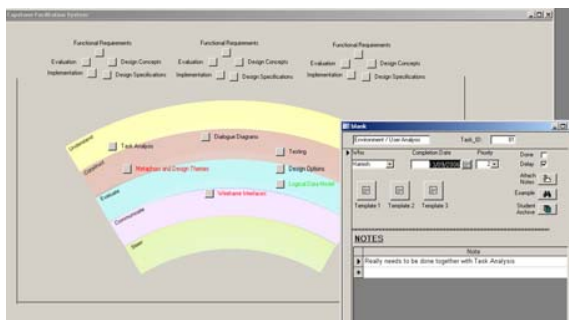


Figure 7: Group management of tasks

Group H saw this flow of information and managing the flexibility of the tasks as the key to the system. They used their own management notes (Figure 6 and Figure 7) to develop a system focused on this support for agility whilst keeping track of the management of tasks. This they developed and tested through paper based prototypes (Figure 8 and Figure 9).



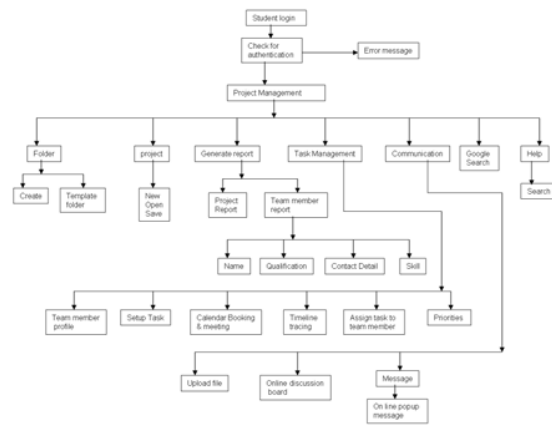
**Figure 8: Group H: The content area has changed to represent the structure of the ADF.**



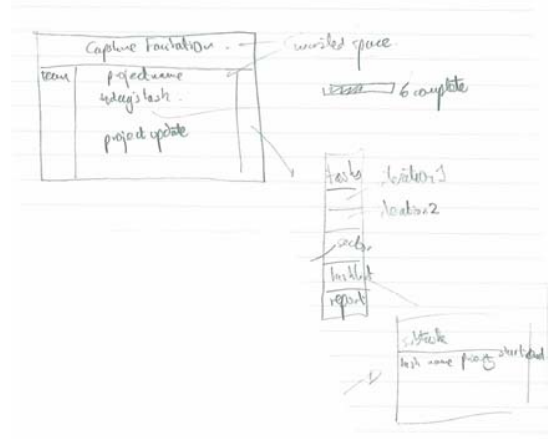
**Figure 9 Group H: Second iteration deliverable**

*“Once a stable database structure was constructed with well-established relationships, we decided to produce a front-end to test the robustness of the back end. Our main goal was to develop the core functionality of the capstone system (the reasoning being that if we could develop the core functions then the other peripheral functions would be relatively easy to incorporate into the system at a later date. What happened...It worked! We were satisfied with the product and felt confident that it did what the system was designed to do eg assign and prioritise tasks, provide access to templates, allowed the addition of notes, and that it proved the stability of our backend. We decided the best way to test this release was to use it to complete the project. We found that it required a minimal amount of*

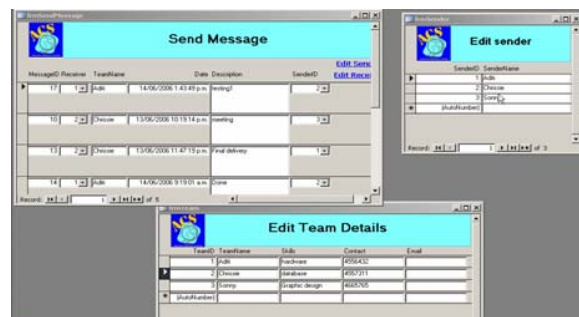
*effort to use and allowed us to track what was happening with each task” (Second iteration, Group H)*



**Figure 10: Group C (supposed to be) dialogue diagram**



**Figure 11: Group C, based at task level and % complete indicator.**



**Figure 12: List based functional delivery**

While Group H developed what could be considered a holistic or integrated approach (perhaps based on their deeper understanding of the ADF), Group C, on the other



hand, continued to focus on detail. This can be attributed to a weaker understanding of the ADF, and in particular, little understanding of interactivity.

*“We did a certain amount of prototyping of design in Iteration Two. With the use of Excel to create interface and the cut-out papers and symbols, we were able to test many users for opinions. It gave us a clearer view as how to navigate around the software environment and understand more of the details that needed to be implemented in assigned tasks in the subtask stages”. (Second iteration, Group C)*

Figure 10 shows what should have been a dialogue diagram representing interactivity, but instead looks more like a menu structure. This flows through to interface design (Figure 11) and components of the Functional Delivery (Figure 12).

### 3.3 Third iteration

In the third iteration, groups go around the cycle again, working towards a Robust Delivery. In terms of teaching, we return the focus to agility, instead of structured classes, we spend the time facilitating scrum meetings, steering paired programming and so on.

In the third iteration Group H moved their system to the web (Cold Fusion, Figure 13). The system is essentially the same as the previous iterations. There is a system of templates with edit and later upload. Multiple tasks can be opened.

The group sensibly saw that some features could not be completed in the confines of Software Engineering and would have to wait: this became known as “Iteration 4”. This is an appropriate interpretation of our intention in the ADF – timeboxing – delivering what you can within a set amount of time and resources is critical to development (Tate 2006). On reflection, the group recognised that some decisions were inappropriate. The choice of what to develop and what to leave until later was somewhat flawed – they had spent time developing the “cracked dials” rather than fixing the actual function of annotation.

After completion the group also recognised that while promoting flexibility, the system had little to explicitly support agility. Here, the integration of course and project has aided learning.



Figure 13: Group H final system

The final product from Group C looks similar to that from Group H. Figure 14 shows a similar spiral and rainbow as the index but then the lists of tasks are poorly integrated (no advance on Figure 12). The group did demonstrate substantial progress in this iteration but, unlike Group H, the change appears to be forced, rather than based on a deeper understanding.

Changes were made somewhat grudgingly in response to “client” instruction but this wasn’t mirrored in general understanding. On reflection the group stated:

*“We did have lots of changes requested from the client; we followed the agile process of using feedback to further improve our design decisions.” (Third iteration, Group C)*

But, while accepting the agile approach of embracing change, discussion about the fundamental purpose of the system (and by association the ADF) was displaced by lots of mechanical details. Client interaction took the form of lots of detailed questions:

1. Can Joe display all tasks on one screen?
2. Is the system automatically saved on a regular basis?
3. Does the system remind Joe that he needs to backup his project?
4. Can Mary view a list of previous reports printed to avoid reprint?
5. Can Mary also access the system to find out who has printed what reports?
6. Can the system handle all iteration on one screen for a print out?

There are fundamental errors in design, for example, the percent complete bars for each iteration (cf the required timebox approach). This reflects how far behind this group was (although this tardy development would have happened even if a different project had been used).

Again, there was little support for agile concepts, the system supporting less flexibility than Group H and, with tasks needing to be assigned to only one person, the system actively discourages teamwork and paired programming. In the final presentation, this group described their system as “providing support for the spiral methodology” rather than Agile Framework.

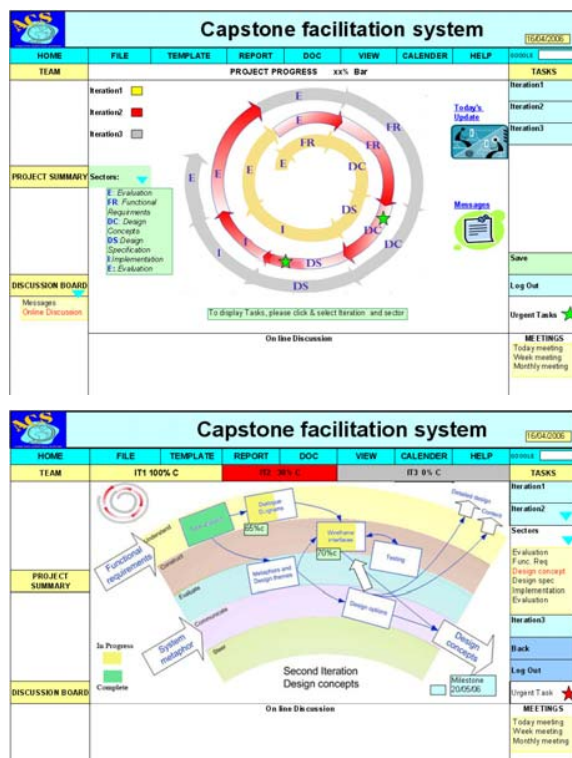


Figure 14 Group C final system

All students in Group C and H passed the project component of software engineering, the exam and the course.

#### 4 Reflection

In this paper we have described an attempt to leverage the incidental learning associated with the project to assist with learning of software engineering material. We did this by using the development of a support system for Capstone projects as the project for software engineering. The questions now are: did it work and would we do it again?

On balance, we conclude that it did work. The trade-off is between a potential confusion of the course and the project, and the benefits of “double dipping” on learning.

*“Overall this has been a worthwhile experience for me technically and personally, in learning the process of software engineering, and in developing skills to*

*become more adaptable to the ever-changing work environment.”* (Student review, Group H), and

*“In my final conclusion of this project I would like to reiterate that this was an enjoyable learning experience for me. As I gained more understanding of software engineering I also develop a greater excitement and passion for the IT sector. So many times in class I was distracted by new software ideas going through my head and having to write them down I sometimes missed what was being taught in class.”* (Student review, Group C)

Two major areas need addressing: formalising client interaction and requirements of implementation.

We had hoped that having one of the lecturers act as the client would mirror the extensive client involvement promoted by agile approaches (indeed this has always been a difficulty of “real clients”). This didn’t work effectively, instead it added to student confusion:

*“We feel that the client interaction was too informal during this process; however this was probably due to ill-defined boundaries where the client was also the lecturer and also the mentor.”* (Student review, Group C)

In future work, we hope to formalise this interaction.

A second area of concern is the requirement to implement systems as part of software engineering. Before we adopted the ADF, the course was document centric and, as the project was not implemented, students were moving into the capstone project with little awareness of implementation issues and often failed to produce the required outcomes.

Getting students to do implementation can have the effect of restricting their design thinking – design decisions are made on the basis of limited technical ability eg: user look up rather than linking. Crucially, however, in this double dipping approach, limited design can be closely related to limited learning.

Strong groups can see past this difficulty:

*“We also feel that since this is our first time directing a software engineering project with the focus on learning rather than on getting every step correct, that our group cannot be too critical of our effort and overall, we consider we have achieved a successful project for the client and the end users”* (Student review, Group H)

*“By following an iterative approach, teams are not focused on producing code, but instead are able to focus on project innovation. This focus increases innovation and cuts delivery*

*time and also encourages parallel-development activities.” (Student review, Group H)*

But for weaker groups this can become a barrier to learning (although it provided many opportunities for learning about conflict resolution and group dynamics).

*“There is lots of paper work to be done; however the detailed analysis steps helped us to understand the process of the project clearly. I think that the relationship between what we originally conceived as our project, and what was our final outcome, was very close. Next time around I would like to prepare myself better in programming so that I can manage to design the software more efficiently.” (Student review, Group C).*

*“As we are all new to this process, we had neither the ideas nor the experience to anticipate the difficulties of working together under a high-stress load. Secondly, with open communication, understanding of the individual commitments and capabilities will certainly help in establishing a functional group able to work together.” (Student review, Group C)*

## 5 Conclusion

The use of a project management system as the project for the software engineering class led to some confusion for some students. However, the benefits of the approach were clear for those students who were able to effectively move between the dual roles of developer and user. The question “what would the user need now?” was easily addressed, and generated useful group discussions.

Apart from the difficulty of using a lecturer as client, the experiment was worthwhile, especially in promoting useful class discussion around the critical area of user requirements. The students’ awareness of project management was extended beyond what they would have gained through a traditional project.

## 6 References

- Mann, S., & Smith, L.G. (2004) Role of the development methodology and prototyping within capstone projects. *Proceedings 17th Annual NACCQ*, Mann, S. & Clear, T. (eds). Christchurch. July 6-9th 2004. p119-128.
- Mann S., & Smith, L.G. (2005) Technical complexity of projects in software engineering *Proceedings 18th Annual NACCQ*, Mann, S. & Clear, T. (eds). Tauranga. July 10-13<sup>th</sup> July 2005. p249-254
- Mann, S. and Smith, L.G. (2006). Arriving at an agile framework for teaching software engineering. 19th *Annual Conference of the National Advisory Committee on Computing Qualifications*, Wellington, New Zealand, NACCQ in cooperation with ACM SIGCSE. 183-190
- Robinson, H. (1994) *The Ethnography of Empowerment: The Transformative Power of Classroom Interaction* The Falmer Press.; Bristol:
- Smith, L.,G., & Mann, S. (2004) Projecting Projects: Choosing Software Engineering Projects, *Proceedings 17th Annual NACCQ*, Mann, S. & Clear, T. (eds). Christchurch. July 6-9th 2004. p183-190.
- Smith, L.,G., Mann, S., & Buissink-Smith, N. (2001). "Crashing a bus full of empowered software engineering students." *New Zealand Journal of Applied Computing and Information Technology* 5(2): 69-74.
- Tate, K. (2005). Sustainable Software Development: An Agile Perspective, Addison Wesley Professional, NY. 264p





# Review of Work Experience in a Bachelor of Information Technology

**Joel W. Pauling, Peter Komisarczuk**

School of Mathematics, Statistics and Computer Science  
Victoria University, P. O. Box 600 Wellington, 6140, New Zealand

peter.komisarczuk@vuw.ac.nz / joel.pauling@vuw.ac.nz

## Abstract

This paper describes the development and evaluation of a significant work experience component within a Bachelor of Information Technology (BIT) honours degree. Work experience provides students with an appreciation of the IT work place, enhances their skills with a range of industry experience and tools and provides a step towards professional accreditation. The initial experiences with the programme are outlined in this paper which discusses the combined support from the Careers Service, Student Job Search and employers to make work experience a success. Feedback from students and employers is provided in this paper indicating industry satisfaction with students, the ICT career aspirations of students, student work patterns and the change in student wages, which are gathered through questionnaires. The indications are that industry is satisfied with the quality of students in work experience, the wages for work experience have increased beyond the rate of inflation and that through the scheme flexibility work experience is gained both part and full time in almost equal measure.

**Keywords:** IT Work Experience, Bachelor of Information Technology, Careers Service, employers.

## 1 Introduction

The designers of the current Bachelor of Information Technology (BIT) honours degree at Victoria University of Wellington, responding to feedback from the IT industry, decided to follow the pattern of many engineering degrees by including a substantial work experience component as a core requirement for students. Indeed the ICT industry is demanding that students gain more work oriented skills. To some extent the work experience resembles the COOP schemes provided in many US, Canadian and European universities. These COOP schemes provide a close cooperation between industry and the educational establishment and have a worldwide organisation – WACE (World Association of Cooperative Education), see WACE (2004).

In the BIT degree the work experience programme requirement is for just two trimesters of work to be undertaken and can include hours credited for part-time work during other trimesters of study. The requirement

on students is that their work experience is taken with just a few employers, none lasting less than 150 hours and that a substantive work experience of more than 300 hours is taken with one employer. In other respects work experience has similar requirements, assessment and administrative requirements as other tertiary work experience initiatives such as COOP schemes.

The initial BIT design was carried out in 2000/1, in a relatively healthy IT marketplace and determined that students should undergo approximately 800 hours of relevant work experience starting at the end of the second year onwards, subject to passing 2 full years worth of papers and four trimesters of study. The work experience can be gained part-time throughout the year or full-time, preferably over the summer trimester as otherwise the student may end up taking longer to complete the degree. Effectively two summers of full time work should complete the requirement for work experience; the final trimester is defined as a “major” work experience, which is broadly defined as non trivial. In such a position the student must demonstrate the analysis, design, development, testing or administration of a non-trivial system.

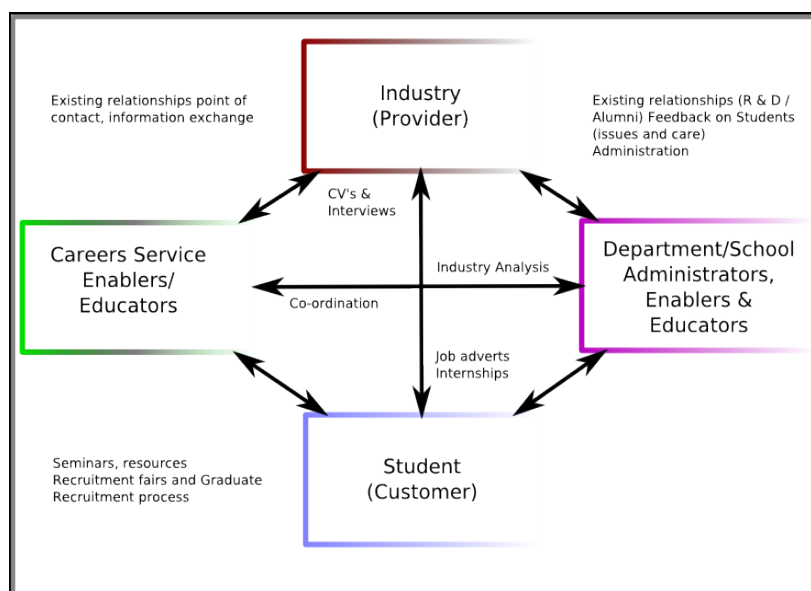
The BIT degree is split into four majors across the schools of Mathematics, Statistics and Computer Science, Information Management and Chemical and Physical Sciences. The four majors are:

- Computer Systems Engineering
- Software Engineering
- Internet Computing
- Information Systems

Students progress to the BIT degree on gaining passes across their part 1 papers and achieve at least three papers with a B+ or better. At the end of their second year students have gained a broad practical perspective in computing science, information management and digital technology. The course foundations are programming (which covers Java, C/C++, VB), systems analysis, information systems, database programming and management fundamentals, an introduction to management and discrete mathematics. The student is also likely to have taken several other papers.

These foundations are used extensively in practical work, including the usual aspects of computing science degrees: data structures and algorithms, networking and multimedia. Furthermore the students will have begun to specialize into one of the four BIT majors in their fourth trimester. Those students taking the Information Systems

Copyright © 2007, Australian Computer Society, Inc. This paper appeared at the *Ninth Australasian Computing Education Conference (ACE2007)*, Ballarat, Victoria, Australia, January 2007. Conferences in Research in Practice in Information Technology, Vol. 66. Samuel Mann and Simon Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.



**Figure 1: Relationships, Information Exchange and Process.**

major will have taken a practical course in systems implementation based on the Microsoft software development process. Others will have taken analogue and digital electronics, computer architecture and an introduction to formal methods.

The knowledge, experience and confidence of students varies greatly, because both mature students and school leavers have been attracted to the BIT. Because of the spread of majors in the BIT the work experience required covers a wide breadth of areas and must provide for wide variance in students levels of maturity.

### 1.1 Work Experience Support Infrastructure

The work experience is effectively governed by a partnership between the schools providing the BIT degree courses and the Careers Service, who act as enablers for students and employers. This partnership is depicted at the top level in Figure 1.

The Careers Service are key to developing student skills in finding employment. They have a substantial set of resources to aid students based around workshops, lectures, self-help materials and one-to-one clinics. These are aimed at graduating students rather than for work experience; however the skills required to find work experience are similar. In fact finding relevant IT work experience is probably a tougher proposition when the work experience is taken over the summer, which in the southern hemisphere includes the Christmas holiday period thus reducing the effective period of employment and the return on investment seen by the employer.

An analysis of the Career Services workshops outlined the following workshops which, based on their existing graduate programmes, could be used to lead the student through the process of finding work experience:

- CV writing workshops and one to one feedback sessions including mock interviews
- Interview and assessment skills workshops – several employers indicated that their selection

process for work experience would include skills assessment through standard psychometric tests

- Graduate destinations information (student job offers, employer), employer profiles and financial results etc.

Events are also organised by the Careers Service, which are based on graduate recruitment initiatives. Recruitment fairs and graduate recruitment events, such as visits by employers aid substantially in interesting the students in certain companies. As these events were established the employers were canvassed for their views on work experience and their readiness to engage undergraduate students. From the perspective of finding future graduate entrants, work experience was seen as a worthwhile activity by many companies.

Employer support in terms of funding can be available for some student work experience. Within New Zealand the Foundation for Research Science and Technology (FRST) provide Technology for Industry Fellowships (see: [http://www.frst.govt.nz/business/fund\\_TIF.cfm](http://www.frst.govt.nz/business/fund_TIF.cfm)) to enable employers to take on senior students for project related activities. The TIF provides NZ\$14 + GST per hour for up to 400 hours to an employer toward the salary for the student. The FRST requires an industry based project, a university mentor and is a common system employed by the largest internship providers, such as Fonterra.

Support services for overseas students (for VISA variance and also in terms of skill enhancement in key areas, such as improving English language skills) are critical as the number of overseas students increase. In the case of the work experience programme the overseas students are able to take some of their work experience within their home country, but they must take the major work experience component in New Zealand.

An externally facing web page <http://bit.vuw.ac.nz> and an internally facing work experience resource based on blackboard have been created for information dissemination and surveying of the students. Mailing lists have also been developed in order to enhance

dissemination of information to students and employers. The external site links to company information and ancillary information for current students, prospective students and information about the process and work experience for employers. The internal site caters for secure job advertisement and as a source for student information with careers advice. The blackboard system has been augmented by the deployment of CareerHub which has been enhanced with a resume builder to aid the management and structure of student work experience.

## 1.2 Employer/University Benefits

The benefits of work experience or internships to the employer are many. Firstly graduates from the BIT are more likely to have relevant business experiences. Secondly the employers have a new means of graduate recruitment – a try before you buy mechanism, in which future employees are identified and steps put into place to retain them.

In some circumstances the employer benefits by the provision of extra hands on projects, for example extra help in product testing, development, or in back office roles to aid sales and marketing. However it is realised that internships over the summer are not optimal as the summer is not usually a peak time in employer productivity from a southern hemisphere perspective, with staff holidays there are fewer supervisors available for student internships etc. As such to optimise student work experience the degree course may need to be restructured to allow students to undertake work experience over other trimesters.

A model employed by IBM in Australia is for students to work for 9 months of the year and spend 3 months of intensive learning at a university over the summer trimester. As such the students are employees of IBM and have their fees paid. Another IBM initiative in Australia offers students employment at their call centre, providing 20 hours per week employment to IT students.

The work experience programme should enable students to better compete in the market place when faced with competition from those graduates with one or two years of experience. The work experience programme also should reflect on the Universities standing within those industry sectors where the graduates are employed. Evidence indicates that graduates have had no trouble in gaining employment once they have completed the BIT.

Obviously through student internships and work experience the relationship between the University and industry have been strengthened. This is in two distinct areas. Firstly through increasing relationships between the careers service and employers through the provision of summer work experience and part-time employment opportunities. Secondly better relationships between the university and industry. Previously very few capstone projects were industry based, now approximately 20% of honour level projects are industrially based or sponsored.

## 1.3 Student Issues and Learning Outcomes

From the perspective of the student the work experience

should provide a set of learning outcomes. The work experience co-ordination needs to focus students on these learning objectives as the majority of students will not actively seek learning outcomes from a work experience. It is generally acknowledged from survey of thick and thin sandwich degree schemes in Europe that the student should keep a log book and take stock of their position in order to view the dynamics of the work place, etc.

Within our experience a pay survey was found to be a key requirement for both students and employers. Employers were keen to identify what level of remuneration was expected by students, likewise students can determine the potential reward. This enabled students to better assess their competitiveness with other job prospects. Students were unhappy taking an ICT work experience if they were expected to take a pay decrease over other types of work. The initial results of the survey indicated two levels of pay for students; firstly for older/mature students and second younger students.

Responsibility for finding appropriate work experience is up to the student, subject to approval by the university administrators. This is a daunting task to many students and so as much help as possible is provided. The careers service provides jobs and seminars for job preparation ([www.vuw.ac.nz/home/studying/careers.html](http://www.vuw.ac.nz/home/studying/careers.html)), Student Job Search ([www.sjs.co.nz](http://www.sjs.co.nz)) provides short term job opportunities and the university has industry contacts.

## 1.4 Student Assessment

The students gain valuable communications skills through their work experience. Through the process of defining themselves in their CV's, through the interview process and the subsequent employment, the student is developing. The assessment of an individual through the work experience was initially very informal. It is recognised a more formal mechanism is required where students present their CV and are assessed through workshops as they progress through preparation and plan employment. Having a formalised approach to Communication Skills Development (CSD) needs to be integrated with the work experience programme, e.g. such as discussed in Gruba (2004). The intention is to further develop these as we move toward a Bachelor of Engineering.

Work experience is either attained or not by a student. The work experience terminates with a written report – an assessment of that aspect has yet to be conducted. The employer can also optionally provide feedback on the student through a questionnaire. The assessment of the student is not dependent on the results of the questionnaire or any comments received through any back checks that are made.

## 1.5 The Work Experience Support Systems

Through 2003 - 2005 a number of support systems were introduced to aid students and staff in the management of the Work Experience processes. These consist of:

- An Electronic Work Experience records database and Web front end developed and

supported within the faculty of science.

- An employment opportunity and events site in in VicCareers which bought the CareerHub web application to provided a university wide web system for career information dissemination. Job advertising is enabled through the CareerHub® system and distributed through email lists.
- A web based portal for students which provides a central point for links to resources and information about the work experience process. This was accomplished through the use of Blackboard® e-learning system as it is widely used throughout the University.

The reason for this number of systems is purely pragmatic. CareerHub is maintained by a third party and required a costly development to include functionality for work experience. Blackboard is not flexible enough to provide all the functionality required.

### **Electronic Work Experience Records**

This consisted of a transition from a paper/spreadsheet oriented record to a web based front-end enabling display and recording of events such as initial employer placement, changes in job description / status and logging of hours worked. The system enables students to fill out work experience application forms on-line, and generates a printable copy to be signed by the employer at each phase. Phase A consists of an initial notification of a placement and phase B the completion of a placement. Additional information can be uploaded at any time and notes added. The retention of the paper/printed step was required to enable verification of placements and as a disaster recovery mechanism. Having all steps initially entered and recorded through a web delivered medium enables the triggering of a number of automated systems, such as notifying work experience administrators.

### **Goals and Outcomes**

One of the primary goals of the Electronic Work Experience records was to provide students with the ability to query current status of their work experience positions, view historical positions and develop their CV. This was originally planned as a module for CareerHub, but was abandoned in favour of purpose built application on cost and management issues. A prototype system was developed as part of a student project in 2004. The prototype was then used as the conceptual basis for the final system which was implemented internally.

The secondary goal was to increase the capture of valuable information such as the contact information of employers, provide the ability to add comments and attachments such as job descriptions and changes in status; for example moving from part time to full time work. An administrative interface was designed to enable staff to generate simple reports showing students current status, pending approvals and total number of hours.

### **Blackboard Work Experience Portal**

The blackboard portal is widely utilised throughout the University, available to all students enrolled and

integrated with student administration systems. Blackboard offers a number of useful features, group e-mail options and the ability to post announcements. Both these features are used extensively to provide updates and information about upcoming events and guides to completing the work experience process for currently enrolled students. The work experience portal was the first Information Support System introduced in 2003.

### **Goals and Outcomes**

Employing the blackboard system allowed the separation of the outward facing website, to be utilised primarily as a information site for employers and prospective students, from internal administrative and assessment mechanisms. It also creates a community and central portal for currently enrolled students to share work experience information and their own experiences.

Blackboard in retrospect has not been an ideal delivery method. Some students have expressed frustration with design features of the package. For example a lack of functions such as Really Simple Syndication (RSS), poor compatibility with major browsers and limited forum functionality. Because of these issues Blackboard has not been fully successful in creating a central community portal for work experience students.

From an administrative perspective this has been further exacerbated by various technical restrictions such as Blackboard's poor survey capability, and an inability to easily add students to the user group by degree (rather than course) resulting in tedious manual addition of new enrolments every year, as well as a number of limitations in blackboard's implementation of e-mail functionality.

These criticisms aside blackboard does fulfil its main purpose as an information repository, link mechanism, and announcement repository. Using a Content Management System which offers more flexibility and technical appeal may supply a more compelling a reason for students to become involved and actively use a portal, creating a "meta" community of innovators within the BIT degree who are highly exposed to various technologies and feel an investment in the development of tools to support themselves and others. These types of schemes have been successfully employed elsewhere (Clarke, 2006).

### **Career Hub Job Placement Application**

In 2004 the careers service purchased a web based application to manage its job advertisement, career events and employer relationship management. This service is implemented as a multiple view web based application. It supports a number of advanced features such as special interest group management and multiple on-line CV management for students.

The BIT administration were invited to participate in the implementation of the service, and a special group was created in the application to tag job positions and advertisements especially relevant to BIT work experience students. BIT students are encouraged to utilise the service by creating profiles and ensuring they were subscribed the BIT work experience group to

receive specific job offers. Limited information is exposed to students outside CareerHub in order to hook them into using the system.

### Outcome and Goals

The goal of the CareerHub application was to provide a targeted mechanism for providing information about career events and jobs relevant to BIT work experience students. Additionally it provides access to existing careers services within the University, such as CV preparation and psychometric testing practice information.

CareerHub has been successful in enabling students to gather information about careers services on campus. The authors cannot comment on the use of the advanced features by students, such as multiple on-line curriculum vitae as no usage information that can be extracted.

The utilisation of CareerHub's CRM functions within the BIT programme has been limited, mainly due to the diverse and infrequent nature of updates, thus its success has been limited in this capacity. This can partially be accounted for because of its externally managed nature and primary utilisation by the careers services which restricts access to some functionality.

## 2 Evaluation of Work Experiences

Since the introduction of the work experience component employers have been asked to provide comments about students, as part of the work experience completion form. We have also designed and administered a feedback survey which has been administered to students from 2003 to 2006. This has allowed us to track information not normally collected such as pay rates, hours of work and desired career specialisation, as well as satisfaction with the work experience process and experience from both employers and students. This feedback is desirable not only for students but also for administration to aid in improving of processes, and allowing students to benchmark themselves against others.

### 2.1 Employer Feedback

Employers are asked to complete a 6 item questionnaire when approving the completion of work experience form. The six target items are rated on a 5 point Likert scale, with 1 being the best possible indicator and 5 being the worst for each of the items. A total sample pool of 54 employers, completed the questionnaire over the 3 year period from 2003-2006. Rudimentary reliability analysis showed high inter-item correlations, and a very high overall scale reliability, as measured by a Cronbach's alpha, of  $\alpha = 0.970$ .

This scale can be thought of as an overall employer satisfaction with work experience students, however it was never designed as a complex instrument and should not be compared to a survey such as "Employer Satisfaction with Graduate Skills" (Hagen, 2004). Thus our feedback mechanism needs to be updated to better record employer satisfaction.

Overall there was an overwhelming positive response ( $M = 1.432$ ,  $SD = .893$ ) from employers. Additional

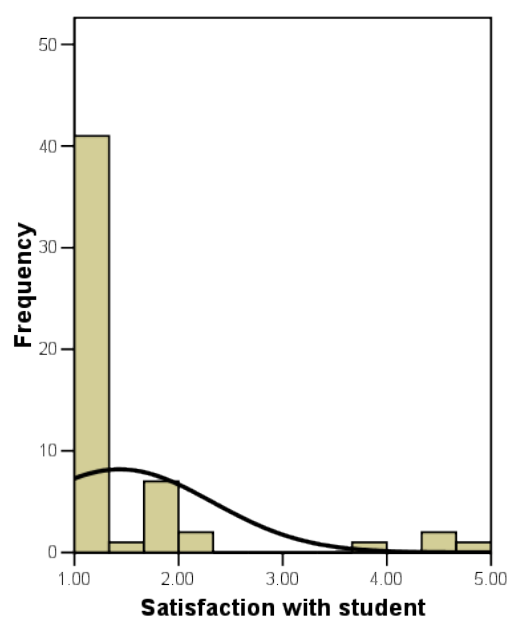


Figure 2: Employer satisfaction 2003-2006.

comments from employers were not analysed for the purposes of this review. Employers appear to be satisfied with work experience students for the most part, with the rate of incidents resulting in poor employer satisfaction limited to only 4 cases from the sample of 54 over a period of 3 years. Figure 2 shows the distribution of ratings among employers with the mean curve overlaid.

### 2.2 Student Feedback

The student feedback survey has been administered once each year since the first BIT students entered work experience arrangements. For the purposes of analysis the first years survey responses (from 2003-2004) are omitted due to the small sample size. The measures used have been standardised to allow longitudinal comparisons between years. Data from 2004-2005 and 2005-2006 surveys are included here.

The survey is administered through a web delivery mechanism. Students are notified of the survey via an e-mail announcement to all BIT students, and supplied with a URL. This announcement was also made available through the BIT work experience portal. Participation in the surveys is completely anonymous and voluntary, only students who have or are currently undertaking work experience are invited to participate. The GPL licensed suite PHPEsp - PHP Easy Survey Package (Team Butterfat!, 2006) used most recently as a delivery method because of technical problems experienced in the past with the survey module within Blackboard. Analysis was conducted with SPSS, R and Openoffice.org software.

Demographic information was not recorded as part of the surveys, however information from enrolment for the proportion of males to females has consistently for the last 2 years has been a 9:1 ratio. Table 1, shows percentage enrolments by ethnicity. Students are eligible to undertake work experience, having completed and passed the BIT part 1 and completed the majority of

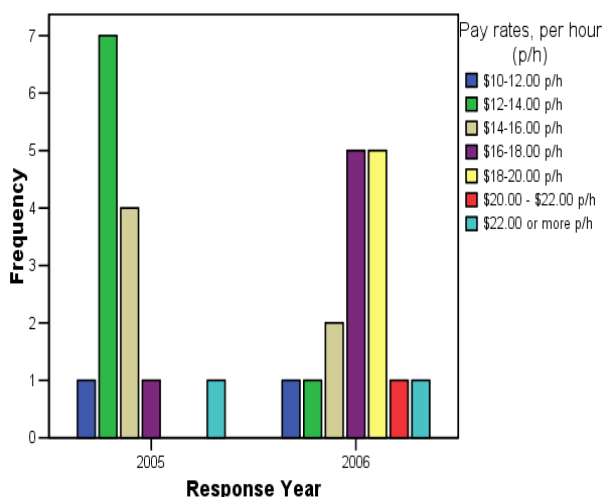
second year papers.

### 2.3 Work Experience Pay Rates

Pay rates for students over the years surveyed have increased significantly ( $X^2(6)=13.76$ ,  $p = 0.32$ ). Some increase can be attributed to inflation, however the median rate has increased from \$12-14.00 dollars per hour in 2004-2005, to \$16-18 for the 2005-2006 period, tending towards the \$18-20.00 range. Figure 3 shows the distribution of pay rates for both periods. This is in-line with the students expected pay rate, which has continuously been at a median of \$14-16.00 p/h combined (inclusive of 2004 period). This increase can also possibly be explained by the increased awareness of current market pay rates and better awareness of other students pay rates in similar positions as provided through feedback channels such as positing survey results periodically to BIT students. This allows students to be in a better position when negotiating contracts with employers. The median rate however is lower than the average hourly wage in New Zealand which was recorded at \$19.30 in 2005 (Statistics New Zealand, 2005)

<i>Ethnicity</i>	<i>Percentage %</i>
NZ/European/Pakeha	53.5
Chinese	14
Other Asian	9
Indian	8
Samoan	5
NZ Maori	4
Cook Island Maori	0.5
Tokelauan	0.5
Fijian	0.5
Other	5

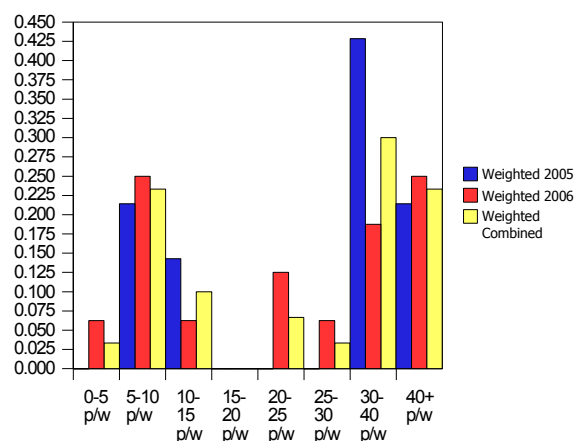
**Table 1: Breakdown of ethnicities enrolled in the BIT degree as of July 2006**



**Figure 3: Pay rate distribution, 2005/2006**

### 2.4 Hours of work per week

Most students work full time during university breaks, however many are also working part time during teaching trimesters as satisfied employers keep them on. This trend has not changed significantly from year to year, with the percentage of students reporting that they are employed during term time at 67.3% over the two years presented here. The data reported for the 2006 year indicates a trend toward longer hours spent in part time work during term time. Figure 4 shows comparisons between the two years sampled. This trend should be examined more closely in the future especially in regard to the students academic performance which has not been measured in the surveys.



**Figure 4: Hours worked per week by year of survey**

### 2.5 Student Satisfaction

The student satisfaction sub scale items were comprised of two seven point Likert styled items, ranging from Strongly Agree(1) to Strongly Disagree(7), designed to elicit opinions about student satisfaction in regard to quality of work experience placements and access to potential work experience jobs through the Work Experience support systems. The scale had a high reliability (Cronbachs  $\alpha = 0.894$ ). Overall the mean for the satisfaction scale was,  $M = 3.483$  ( $SD = 1.483$ ), no significant difference was found between years ( $F=1.181$ ,  $p = .398$ ).

These results can be roughly interpreted as a middling satisfaction with the available work experience positions. Open ended comments and feedback from students on this topic evidenced some dissatisfaction. A number of students were dissatisfied with the predominantly software engineering nature of advertisements, while others expressing the lack of the same! While efforts to find and relay targeted positions for the various specialisations directly to BIT students, have been made, this example demonstrates the doubled edged nature of advertising positions directly, rather than encouraging students to seek positions through other means such as family, CareerHub and Student Job Search, which allows them to feel a sense of ownership and accomplishment in regards to the positions they are seeking.



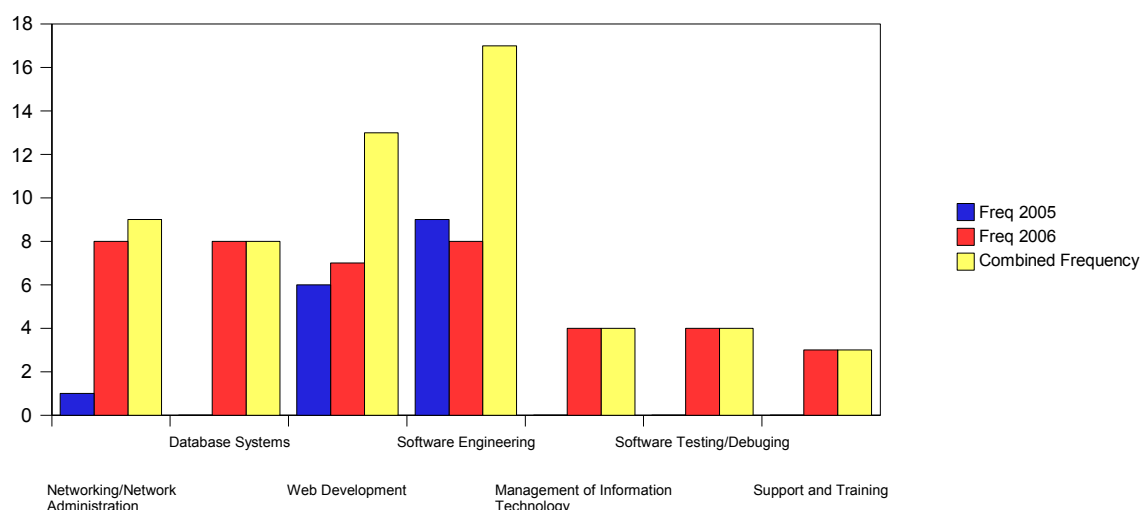


Figure 5: Frequency of students choosing ICT area as a career

## 2.6 The Perceived Worth of Work Experience

The analysis of student perception of work experience used a scale comprised of four seven point counterbalanced, Likert styled items, designed to elicit student's experiences with work experience positions they had held. Students were primed with preceding items to evaluate only the most recent work experience placement. The scale includes a perceived acquisition of learning/knowledge component, workplace environment factors, training and instruction and recommendation of employer to other students.

The scale had a satisfactory reliability (Cronbach's  $\alpha = 0.781$ ) and an exploratory factor analysis revealed all items loaded onto one factor with Eigen values  $> 0.78$  for each item, however the authors admit that with such a small sample size the statistical validity of any factor analysis is dubious. An FA does confirm the validity of the scale and rationale for grouping these items as one factor rather than individually evaluating them.

Students experiences were strongly positive  $M = 2.267$  ( $SD = 1.0723$ ), with no significant difference between surveyed years ( $F = 0.918$ ,  $p = 0.544$ ). Positive experiences were not significantly correlated with the satisfaction with advertised positions. This finding suggests that the administrations role in providing directly targeted work experience positions for students had little impact on the perceived worth of the work experience itself.

A follow-up regression was performed against the student satisfaction scale to determine if there were any corresponding explanations of variance. While the overall regression for the student experience sub scale was non-significant, one individual item's beta weight was very high ( $\beta = 0.598$ ) and significantly different to the other items in the scale in explaining variance in the student satisfaction sub scale. This item was the item relating the amount of training and instruction provided in the work placement.

Comparing these results with the open ended feedback, indicates these results are typical. Many students felt that their work experience provided the most important part of

their degree, but at the same time felt that there was some un-preparedness on the part of the employing organisation for taking in students, especially in terms of training and work structures. There was more negative feedback with regards to administration and work experience availability in the 2004-2005 sample as compared to the open ended feedback from the 2005-2006 sample. This is suggested, by the survey evidence, a result of providing more targeted positions directly to students in 2004-5, as opposed to the 2005-6 period where students were provided with and encouraged to use external resources to seek jobs. This is perhaps a function of saliency, because the jobs advertised in 2004-5 tended towards more software/web development and those not seeking those positions felt "left out". This is an insightful yet counter-intuitive finding. It is suggested that this is a wider criticism of the wide ranging specialisation areas present within the BIT degree, of which targeting work experience placements for in sufficiently equal number is problematic. The current situation, while not ideal, of only supplying wide ranging scholarship and internship positions directly to students through direct communication channels has attenuated some of this criticism.

In the 2006-7 period the number of internships on offer has increased significantly. To the end of September there was an increase of 40% in employers seeking students for summer internships. These job opportunities span from government departments and large businesses such as IBM, through to technology incubator start up companies that created a "summer of code" around web application development.

## 2.7 Career/ICT Area of Choice

Students were asked to select the ICT areas which they were most interested in working within. They were allowed multiple choices from 7 broad categories as well as an optional "other" category. This data shows how students preferences for certain positions have developed over the two year period of the surveys presented here.

Figure 5, shows the breakdown of preferred area of work

selected by students. The most striking feature is that those sampled in 2006 tended to choose multiple categories. Whereas the 2005 year tended towards either software engineering or web development exclusively. Areas such as support and testing as well as management have become more desirable, with web development dropping in preference.

## 2.8 Other Observations from Surveys

Students were also asked about their ideal scenario for work experience in terms of how many positions and what type of organisations they would like to work for to make up their 800 hours. Roughly half of students indicated a preference to work within a single large organisation, but working in different areas or different projects. The other half preferred to gain work experience with a number of small companies instead.

Other recurring themes in open ended feedback is the desire to gain industry accreditations such as Red Hat Certified Engineer (RHCE) and Cisco Certified Network Architect (CCNA) alongside traditional university content. Requests were made to provide learning opportunities around specific technologies such as Perl, Visual Studio and Linux. In 2004 we provided "Extended Educations" sessions where volunteers ran evening workshops on these topics. These were judged highly successful by the students, but left many wanting more in-depth sessions. This was not impossible given the volunteer nature of the sessions, but given funding would be ideal natural extensions of degrees including a WE component.

Students identified a desire to acquire skills in a number of technologies, languages and programs. There was a wide fluctuation in those mentioned most frequently from year to year, with a number of notable exceptions. Those exceptions are Java, (for obvious reasons as it is the core language taught in principle computer science papers), HTML/XHTML and Linux, which all seem to remain staples. Others such as .NET and Ruby see wide variations, for example .NET in 2004 (that year heavy marketing was introduced for the visual studio .NET product line) was mentioned as the fourth most requested topic, while in the 2005-6 sample it was mentioned only once. Ruby has undergone a similar rise in popularity in the 2005-6 sample. Other technologies that students would like more on are MySQL/MS-SQL server.

## 3 Final Thoughts

Acceptance for work experience within the IT industry was relatively difficult in the first two years of the programme. However from 2005 onwards the programme has become much more self sustaining with a number of internship offers regularly coming in. Some of the bigger players are now aiming to provide summer and one year internships such as IBM. Now that the BIT has reached maturity around 100 students per year are being given work experience in a wide variety of industry sectors. Students have typically undertaken work with 2 or 3 employers, although a few have had greater than 5. Students have often undertaken significantly more than

800 hours of valid work experience. Some students having gained over 2000 hours of work over a three year period, and one has clocked up 3400.

Findings from graduate destination surveys indicates that most graduates find employment in larger companies, typically with > 100 employees. For the students in work experience this trend seems to also be the case, with the majority of students employed by large local employers. Employment opportunities have been found from the lowly ICT helpdesk through to testing of JD Edwards One World solutions, .NET and Java development, web development and administration.

A small student company was formed to provide support services to high tech incubator companies. The student company developed a number of good contacts and worked on a number of different projects. This enables students to take on smaller jobs within the umbrella of a technology incubator and exposes them to some of the creative thinking within the local ICT landscape. These startup employers typically cannot sustain the employment of a student, however together they form a significant employment opportunity. This model has moved on to develop the "summer of software" concept where the technology incubator companies club together to share a pool of student resources across a number of projects, which is partially funded by government grants.

Feedback from students and employers has been provided in this paper indicating industry and student satisfaction. The results indicate the changes in ICT career aspirations of students, student work patterns through this work experience programme in both part time and full time work opportunities. The indications are that industry is satisfied with the quality of students in work experience, the wages for work experience have increased beyond the rate of inflation and that satisfactory work experience is gained both part time and full time and given the opportunity students far exceed the minimum number of hours required within the degree statutes.

## 4 References

- CareerHub (2003): CRM solution for Career Services. <http://careerhub.co.nz>. Accessed August 2006.
- Clarke, R.(2006): There and Back Again – IT Provisioning for IT Students. *19<sup>th</sup> Annual Conference of the National Advisory committee on Computing Qualifications*. 39-43. Mann, S. & Bridgeman, N. (eds)
- Gruba, P. & Al-Hamood, R.(2004): Strategies for Communication Skills Development, *Sixth Australasian Computing Education Conference (ACE 2004)*. 101-107.
- Hagan, D.(2004): Employer Satisfaction with ICT Graduates, *Sixth Australasian Computing Education Conference (ACE 2004)*. 119-123.
- Team Butterfat! (2006):PHP Easy Survey Package. <http://phpesp.sf.net/>. Accessed 12<sup>th</sup> August 2006.
- WACE (2004): World Association for Cooperative Education. <http://www.waceinc.org/>. Accessed 25<sup>th</sup> August 2004.



# Mandatory fields – a case study in the divergence between education and practice

Simon

School of Design, Communication, and Information Technology  
University of Newcastle, Australia

simon@newcastle.edu.au

## Abstract

In an ideal world the practitioners, educators, and students in a given discipline would share much the same perspectives. In the real world these groups often display a marked divergence. We illustrate this divergence with a study of the usage of mandatory fields in databases, showing that practitioners often end up compromising data integrity by forcing the collection of data that is inaccurate and contrived. We follow the case study with an appeal to database educators to teach their students to give more serious consideration to the choice of mandatory fields.

*Keywords:* mandatory fields, data integrity, names, usability.

## 1 Introduction

Educators tend to believe that their perspective on their subject matter is a good one, perhaps even the best one. They hope that their perspective will eventually be shared by their students, and that those students will later apply that perspective as practitioners in the field.

The reality is that while educators might indeed have a good perspective on their subject matter, many of their students do not acquire the same perspective in its entirety. On becoming practitioners, the students will develop a perspective that is influenced by their teachers, their peers, any existing systems on which they might be required to work, and their own sometimes misguided preconceptions. This can lead to seriously flawed systems.

We illustrate the problem with a case study in which we show that the general practice is widely divergent from the ideal perspective that we hope most educators espouse.

## 2 Mandatory fields – a case study

Most internet users outside the United States have come across the parochialism evident in much of the software based in that country. A user purchases a piece of software, tries to register it, and fails. One likely reason is that a State (or perhaps a State/Province) is mandatory –

and it must be selected from a drop-down list that includes only the states and provinces of the US and Canada. Another likely reason is that the user must provide a zip code, and it must be in the precise format of US Zip Codes.

The user then has two options: giving up the attempt to register the product, to purchase the service, or whatever else is being transacted; or continuing, providing false data because the system will not accept the true data. In the second case, a constraint intended to ensure that only accurate data is accepted ends up forcing the entry of false data.

Internet developers from other parts of the world tend to understand this parochialism, to understand that many citizens of the United States seem only vaguely aware of the rest of the world and even less aware that it is anything but a pale copy of the United States. And these developers smugly tell themselves that of course they would never be guilty of making the same sort of assumption in their own work.

This case study is a survey of some of the ways in which software developers make just that same sort of assumption in their own work, with very similar consequences on the integrity of the data collected.

A number of the examples given in the paper are anecdotal and are provided without substantiation. It is hoped that readers will accept these examples, or will at least accept that they are plausible, and will continue to read the paper in the spirit in which was written.

## 3 Driver's licence as a mandatory field

For reasons best known to itself, the video rental industry in Australia tends to have settled on the driver's licence as the ideal form of identification for customers. This need not be a bad thing, except that in some instances at least, 'the ideal form of identification' translates to 'the only acceptable form of identification'.

This is then implemented as a business rule in the databases of various video rental outlets. Driver's licence is made a mandatory field, and registration is not possible if that field is left empty. The consequence is that somebody without a driver's licence is unable to register with such a video rental outlet.

There are many adults in Australia who, for one reason or another, do not have driver's licences. That this should exclude them from registration with a video rental business could be seen as ludicrous. When business rules

turn away perfectly good customers, something is wrong with the business rules.

#### 4 Landline phone number as a mandatory field

A colleague related to us recently that he had tried to purchase an expensive item by telephone, and had failed because he could not provide a home phone number. The system on which his order was being recorded had home phone number, in landline format, as a mandatory field. He ended up purchasing the same item from an internet vendor who did not have the same requirement.

The same frustration is felt by a number of people who like to order home-delivered, or even pick-up, pizza. The business rules of certain fast-food delivery businesses depend on a landline home phone number, and if that mandatory field cannot be filled, the order cannot be recorded.

While most, though by no means all, Australians have some form of telephone contact, an increasing number have decided to dispense with a landline number in favour of a mobile one. A business whose own rules preclude commerce with these people is doing itself a grave disservice.

#### 5 Suburb and town as mandatory fields

The Leukaemia Foundation of Australia has for a number of years conducted the ‘World’s Greatest Shave for a Cure’, in which many people volunteer to have their heads shaved in order to raise money for the foundation’s research program.

People volunteering to take part in the shave are invited to register by way of a Web form. Like many Web forms, it marks the mandatory fields with asterisks. Unlike many Web forms, it includes fields for Suburb and Town, and both are mandatory.

Many years ago, before the introduction of postcodes in Australia, Suburb and Town were parts of many, though not all, Australian addresses. People lived in Maribyrnong, Melbourne, Victoria, or in Woolloongabba, Brisbane, Queensland, or in Fannie Bay, Darwin, Northern Territory, or in Frenchville, Rockhampton, Queensland.

Nowadays, and this is by no means a recent development, those same addresses are rendered Maribyrnong, Vic 3032; Woolloongabba, Qld 4102; Fannie Bay, NT 0820; and Frenchville, Qld 4701. What was the suburb has become the town, and what was the town is no longer part of the address.

We contacted the Leukaemia Foundation to suggest that making both suburb and town mandatory meant that almost every volunteer was forced to provide an incorrect address. They agreed, and indeed confirmed our suspicion that most volunteers simply doubled the place name: *Suburb* Woolloongabba, *Town* Woolloongabba, for example. This would of course result in the printed address including, probably on separate lines, Woolloongabba Woolloongabba.

We speculated that the only people able to provide something approximating a correct address were those who lived in place with double-barrelled names. They could enter *Suburb* Woy, *Town* Woy, or *Suburb* Clarendon, *Town* Vale, or *Suburb* West, *Town* Binnu. These pairs would then print more or less correctly as Woy Woy, Clarendon Vale, and West Binnu, although the names might be split over two lines.

Intriguingly, while the Leukaemia Foundation wholeheartedly agreed that the requirement for both fields was unreasonable, they have done nothing to change it.

The Royal Society for the Prevention of Cruelty to Animals (RSPCA) in Australia runs Paw Prints, a club for children. To register for membership of this club, children must provide title, first name, last name, date of birth, street address, suburb, city, state, and postcode. There are also some optional fields: address continued, contact number, email, favourite animal, how did you hear about us.

In both of these cases, and others that we imagine are out there, providing both Suburb and Town/City fields and making them mandatory actually forces the users to enter incorrect data. One presumes that this is not the intention. Why, then, does it happen?

We speculate that it is because somebody involved in the development of the database or the registration form is old enough to remember when some, though by no means all, Australian addresses did have this form, and has not caught up with the change that took place some 30 years ago. While it seems unlikely, we find this more plausible than the alternative explanation that a younger developer who has grown up with the newer form of address has somehow re-created the old form of address and decided that it should replace the real one.

#### 6 Address as a mandatory field

Most database developers would agree that in some form or other, address is reasonable as a mandatory field for many systems. We remind readers of the classic counterexample.

On 27 July 1996, during the Atlanta Olympics, a bomb exploded at Centennial Park, one of the main Olympic venues. A little more than 20 minutes before the explosion, a man called the emergency line and told the operator that a bomb was set to explode at Centennial Park in 30 minutes. The emergency operator tried to log the call so that it could be dispatched to the police, but failed because the computer logging system would not accept ‘Centennial Park’ as an address – it insisted on a street address.

A transcript of the relevant telephone and radio calls was released by Atlanta Police Department, and was published on the Risk-List, a forum of the ACM Committee on Computers and Public Policy. The transcript is still accessible on the web, for example as a case study provided by an academic at the University of Brighton (Neumann 1996).

The transcript clearly shows that, while there were other problems such as calls not being answered promptly, the search for the street address caused a delay of nearly 10 minutes – half the time between the warning call and the actual explosion.

It is not clear whether a further 10 minutes notice would have helped prevent the two deaths and many injuries that resulted from the bomb blast. What is clear is that a delay of this nature, caused by the system's insistence on a street address for a landmark location, can and should have been avoided.

## 7 Title as a mandatory field

It can be enlightening and entertaining to consider the purpose of titles: what are they for and do they achieve their goal? We believe that most rational thinkers would quickly agree that universal titles are quite pointless.

Be that as it may, many people like to use a title, either because it is one they have earned or simply out of habit. Does this mean that everyone must be saddled with one?

A vast number of databases have title as a mandatory field. Worse, most of these use a drop-down field to severely limit the choice of titles to Mr, Mrs, Miss, and Ms, totally disenfranchising people whose title is Doctor, Commander, Rabbi, Her Royal Highness, Professor, or any of hundreds of others.

Craig Cockburn has written an entertaining and persuasive article on Web usability (Cockburn 2003) that clearly explains the folly of insisting on a title (when a third of UK users prefer not to use one), of limiting the choice of that title, and of using people's titles to deduce their sex (which is illegal in many countries).

## 8 First name and last name as mandatory fields

In this section of the paper, for reasons that will be obvious, we intend to dispense with the customary academic first person plural (we) and instead use the first person singular (I). This will help to avoid such odd assertions as 'we have only one name'.

### 8.1 Not everybody has two names

Surnames or family names are not universal; for example, many Tibetans do not have surnames, and many male Javanese do not have surnames. Many Indonesian public figures have one-word names, but people outside those countries tend not to register the fact. Who in Australia knows that former Indonesian President Sukarno and former Indonesian President Suharto had no other names? Among those who carried out the 2002 Bali bombing, Mukhlas and Idris have no other names.

Closer to cultural home for many, albeit further away geographically, many royal families do not use a family name, though for some of those a family name can be found if it's really deemed necessary.

If any of these people is to be entered in a database that insists on two names, a false name must be invented to

satisfy the system. Alternatively, the attempt must be dropped and the person's record not entered.

Software developers in Australasia might argue that this is none of their concern, that they are not writing packages for use in Indonesia or Tibet. While this might be true, the current move toward globalisation and cultural sensitivity would suggest that it would be wise to recognise the issue even at this distance.

A stronger reason for recognising the issue is that people from those countries do come, temporarily or permanently, to Australia, New Zealand, and other countries whose culture is based upon one from Europe. I still recall the embarrassment I felt 30 years ago on learning that a colleague had told a student from Indonesia that he wouldn't be able to get by in Australia with just one name, and he would have to think up a second one.

In addition to these visitors there are people who have at some time changed their name so that it consists of just one word. Some 35 years ago I changed my name to Simon. My driver's licence and passport are issued in that single name, but many databases are programmed not to accept it.

### 8.2 A hospital in New South Wales

I was once taken to the emergency department of a hospital in New South Wales. The hospital admissions staff could see that I was in severe pain and needed to be treated, but they would not proceed to that stage until I had been entered in the computer system, and this could not happen until I provided a second name. Needless to say, I eventually did so, and that hospital's records have ever since recorded me by an invented name that I chose on the spur of the moment and promptly forgot.

On a more recent visit to the emergency department of a different hospital in a different Area Health Service, I realised the error of my previous invention, and invented a second name that (a) I would not forget, and (b) nobody would mistakenly take to be my real name. When I was recently admitted to that same hospital for two weeks, the registered name of Simple Simon caused a great many raised eyebrows, a certain deal of mirth, and even some embarrassment.

### 8.3 An Australian university

The University of Newcastle, my employer, recognises me by my correct name. But I have recently taken issue with the proliferation of Web-based forms that have both Surname and First Name as mandatory fields. I have taken this complaint as far as the University's Database Administrator, who told me bluntly that the forms would not be changed to permit the entry of a single name, because it would compromise the data integrity.

The DBA's suggestion was that I invent a second name and use it consistently for all University purposes. He seemed unable or unwilling to accept that this would be allowing his computer system to mould reality rather than reflecting it, and that it would ensure the collection of false and inaccurate data. At the same time, he was

appalled when I announced my intention of entering a different First Name each time I had to use one of these forms, and in so doing to work my way through the alphabet, alternating between male and female first names. Even the DBA could see that this would compromise the integrity of the data.

#### 8.4 Respected international computing conferences

Most computing conferences and organisations offer and indeed require online registration. Surely they, being organised by leading computing academics, will get it right?

Figure 1 illustrates that this would not be a good assumption.

#### 8.5 The Australian Taxation Office

Several years ago I decided to try the Australian Taxation Office's new e-tax system. The ATO itself had never had any trouble with my single name, so I optimistically expected e-tax to be equally amenable.

When I indicated that I was ready to lodge my return, I was directed to a page that listed three warnings: the Given Names field was blank; no interest earnings had been declared; and work-related expenses exceeded \$300. These were all true, but did not reflect erroneous data entry, so I proceeded to lodge the return.

On each of several attempts I was sent the same error message: "No digital signature files that match the current password were found in the current directory."

I did as directed and telephoned the e-tax helpline for assistance. I boldly suggested that my single name might

be the problem, but this suggestion was spurned by the helpline staff, who assured me that if the ATO accepts my name, so would e-tax. Instead they had me check my browser, download and install a newer version, try a different browser altogether, and so on. Eventually I persuaded the staff member to connect me to a supervisor, who concluded after some time that the problem was indeed my single name. I was advised to print the e-tax submission and post it in, and was told that the problem would be reported and fixed by the following year.

When the same problem arose the following year, I phoned the helpline again and managed to be put through to the same supervisor rather earlier in the proceedings. The supervisor apologised, advised me to print the e-tax submission and post it in, and suggested that I write to appropriate Deputy Commissioner of Taxation.

I took up this suggestion, not only asking for the problem to be fixed but also suggesting that the software be altered so that fatal errors would show up as such, rather than being listed alongside non-fatal warnings; and further that the generic error message be replaced with one that was rather more specific, to assist both the user and the helpline to diagnose the problem.

(That year's message said

*Lodgement Error – S8*

*Lodgement of your 2001 income tax return has been unsuccessful.*

*Please close your browser and lodge your return again.*

*If you have further problems lodging your return please ring the e-tax help desk on 1300 1300 17.)*

I was pleasantly surprised to receive a reply from the Director of Electronic Initiatives apologising for the

The screenshot displays the ACSW 2007 Registration Form. On the left, there is a navigation menu with links: Home, News, About ACSW 2007, Conferences, Important Dates, Invited Speakers, Registration, Program, Social Activities, Tours, and Committee. The main content area includes the ACSW 2007 logo, the event dates (Jan 30 - Feb 02, 2007), and the location (Caro Convention Centre, University of Ballarat, Mt Helen Campus, Ballarat, Victoria, Australia). A banner for 'Early bird registration' is visible, stating it is on or before 28th November 2006. The registration form fields are partially filled: First name (empty), Last name (Simon), Middle name (empty), and Address (Newcastle, NSW, Australia). A JavaScript error dialog box is overlaid on the form, titled '[JavaScript Application]', with a warning icon and the message: 'Registration Error. The following required fields do not have values: - First Name'. An 'OK' button is at the bottom of the dialog.

Figure 1: respected international computing conferences needlessly insisting on two names

problem and assuring me that it would be fixed for the following year. The Director also confirmed my suspicion that I was by no means the only Australian taxpayer with a single name.

Submission of e-tax involves two separate phases: preparation and lodgement of the tax return, and verification of identity. The latter phase produces a username and password that are used when lodging the return.

On my third attempt, I found that the problem with the e-tax software had indeed been fixed, and further that the distinction was now being made between non-fatal warnings and errors that would prevent the return from being lodged (Figure 2). Unfortunately, the identity verification software had also been rewritten, and now objected “*Error – Given name is a mandatory field*”. I tried to work my way round this problem, but the verification software ties in with the ATO’s own records, and the ATO’s own records had only one name on my record, so there was no solution. I printed the e-tax return and posted it, and penned yet another letter to the Director of Electronic Initiatives. She was even more apologetic this time, and assured me that the problem would be fixed. And indeed it was – in my fourth year of trying, I was able to submit an e-tax return for the first time.

I believe that the ATO was willing to change its software for three reasons: first, because the problem was real and was clearly demonstrated; second, because the ATO values data integrity and accuracy; and third, because the software was being developed in-house. The programmers and developers who were writing it were working directly for the ATO, and so could be asked to remedy this problem while working on other updates.

By contrast, for example, the hospital system of section 8.2 was almost certainly purchased off the shelf, so the cost of having it customised would be prohibitive; and accuracy of data is perhaps not quite so important to hospital admissions staff as it is to executives of the Australian Taxation Office.

Unfortunately, even good designs can be undone by later generations of developers. In 2006, while the e-tax system itself is still willing to accept taxpayers with only one name, the identity verification software has been

rewritten and once more insists on a first name – despite having already asked for *full* name in another field!

## 9 Just what is accurate data?

Many people have four or more names. Many databases have two mandatory name fields. If a person with four names enters only two of them, does that compromise the integrity of the data? I suspect that most DBAs would feel that two names out of four are enough for accuracy.

Extending that argument, or perhaps contracting it, if a system has only one mandatory name field, and a person with two names chooses to enter only one, does that compromise the integrity of the data? I suspect that most DBAs would feel that one name out of two is not enough for accuracy.

Why this distinction? What is so special about the number two when it comes to names? The simple answer is comfort. Most developers and administrators are accustomed to thinking of people as having two names, even those people who in fact have more, so they are quite comfortable collecting only two names from somebody who has more, but are most uncomfortable at the thought that some people might have fewer than two – and are therefore more than happy to disallow such a possibility, regardless of the truth of the situation.

But is it those two names that identify a person in a database? Of course not. That combination of first and last names is not even guaranteed to be unique. What identifies the person is the primary key, which is almost certainly artificial. So if a person with two or more names decided to enter only one in a system, there would be no compromise to data integrity, which would still be assured by the primary key. And it would surely not be as bad as forcing people to invent names so as to qualify for entry in a database.

Of course safeguards are still possible. It would not take a great deal of effort for a developer to program a warning; “You have only entered a last name; if you have other names, please go back and enter a first name as well.” Figure 2 is an excellent example of this.

## 10 What do the textbooks say?

I searched the indexes of the 19 database design,

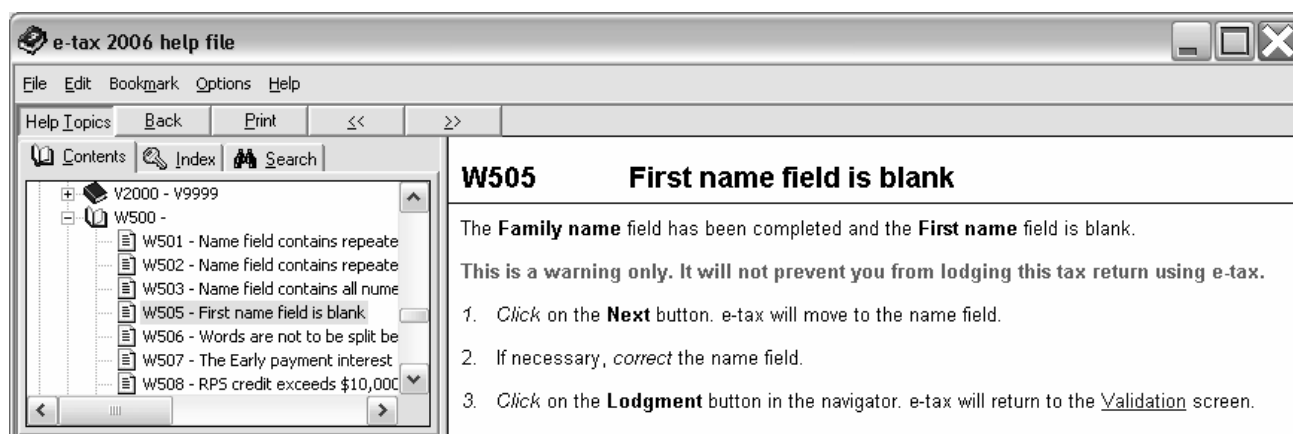


Figure 2: sensible treatment of non-fatal warning for missing first name – Australian e-tax

management, and implementation textbooks that I found on my shelves, looking for either ‘mandatory field’ or ‘required field’.

Most of the books that listed the phrases at all were implementation books, which gave extremely brief and extremely functional mentions. Examples are ‘The Required property rejects any record that does not have a value entered for this field’ (Grauer & Barber 2001) and ‘The Required property for Field objects can be used to disallow Null values in the column. Setting this property to True means that a value is required.’ (Winemiller, Roff, Heymann, & Groom 1998).

While most database design books were more likely to write about mandatory relationships or mandatory access control, one did mention mandatory fields, with an example: ‘Some columns must contain a valid value; they are not allowed to contain nulls . . . For example, every member of staff must have an associated job position.’ (Connolly & Begg 2002).

One further design book (Rob & Semaan 2000) did not list the concepts in its index, but I was pleased to note that in some sample databases provided on CD, FNAME was a required field while LNAME and INITIAL were not. I would have been more pleased if I had found this choice explained and justified somewhere in the text.

If textbooks do not encourage students and academics to consider the point of mandatory fields, it is highly likely that such fields will be assigned more or less at the whim of each developer. This is unlikely to be a good thing.

## 11 What fields should be mandatory?

With no help from textbooks and little from the academic literature, perhaps it is time that somebody proposed a purpose for mandatory fields.

I propose that a field should be set to mandatory if there is simply no point in storing a record in which that field is null.

This position garners some support in the literature. For example, in the context of internationalisation Chan & Suwanda (2000) ask ‘Which address fields are mandatory or optional for a shopper for each region (for example, first name, last name, e-mail, state, county, fax number, customer number, etc)’? They appear to assume that first name and last name can be mandatory in Canada, from where they write, but at least they acknowledge that this might not be valid the world over.

Little et al (2000) propose an exercise on ‘International Differences in Data Fields’, whose purpose is ‘to help students become aware of differences in multinational dates, addresses and telephone numbers’. Students are asked to consider a database design with twelve fields of which nine, including First Name and Last Name, are mandatory. While the exercise makes no mention of a potential problem with names, a generous interpretation of the word ‘address’ might hint that the name constraints should be considered as well.

Seiden (2002) begins with a warning of the pressure to conform, and quotes Jakob Nielsen who writes, albeit on

the question of Web page design, ‘If **80% or more** of the big sites do things in a single way, then this is the **de-facto standard** and you *have* to comply.’ Mind you, he does qualify this with ‘Only deviate from a design standard if your alternative design has at least 100% higher measured usability.’ (Nielsen 1999).

But Seiden goes on to explain why developers should not necessarily conform, and writes:

Like error messages and breadcrumbs, required fields are an indicator that that a better design may be available. Users dislike them because they reverse the basic social dynamic of commerce, which is that the customer is always right.

When you find yourself using required fields, ask yourself “Is this bit of data worth transforming the nature of my relationship with my customer?”

Ask yourself “Is there another way I can get this information? Can I wait and gather it later? What is the minimum amount of information I can collect and still function appropriately?”

Most important, ask “For whose benefit am I collecting this information?” If it’s for your own benefit, or to make the software easier to code, you probably need another design.” (Seiden 2002).

## 12 Conclusion – closing the gap

It appears evident that the use of mandatory fields is driven more by whim than by necessity. We have illustrated many examples of mandatory fields that are so poorly thought out that they either prevent the entry of valid records or force the invention of false data. We note that few textbooks on database design or implementation even consider the question of which fields should be made mandatory and why.

Computers and their software should reflect the world, not shape it. We believe, probably with good reason, that most computing educators agree with this position. So what has gone wrong? Why are the software developers we have educated forcing the world to conform with their limited vision of it as expressed in their software?

We believe that it is a simple matter of emphasis. It is not enough to lead by good example; instead it requires explicit instruction. Once we become aware of a great divergence between good design and common practice, it is our responsibility as educators to try to close the gap. In this particular instance, this would mean illustrating the problem with examples such as those presented here, and emphatically urging our students never to make a field mandatory unless that field truly is essential to the record of which it is part.

Why are we appealing to educators rather than to database designers and administrators? Because by the time a person becomes a database programmer, designer, or administrator it is too late; the pressure to conform will have become irresistible. The only way to get this

message into the database world is to start when the students are young (at least in the professional sense) and impressionable.

At a recent question-and-answer session I asked my University's new CIO if the proposed new all-encompassing staff database would cope with a staff member who had only one name. He grinned and said "I thought I recognised you, Simon! You taught me years ago, and I still remember you asking us how a system could cope with your name if it made two names mandatory. Yes, the new system will accept your name."

You see, educators *can* make a difference!

### 13 References

- Chan, Y., Suwanda, H. (2000): Designing multinational online stores: challenges, implementation techniques and experience. *Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research*, Mississauga, Ontario, Canada, 1-14, IBM Press.
- Cockburn, C. (2003): Courtesy titles – their proper use and website design guidelines. Accessed at <http://www.siliconglen.com/usability/courtesytitles.html>, August 2006.
- Connolly, T., Begg, C. (2002): *Database Systems: a practical approach to design, implementation, and management* (3rd edn), Addison Wesley.
- Grauer, R.T., Barber, M. (2001): *Exploring Microsoft Access 2000 with VBA*, Prentice Hall.
- Little, J.C., Granger, M., Adams, E.S., Holvikivi, J., Lippert, S.K., Walker, H.M., Young, A. (2001): Integrating cultural issues into the computer and information technology curriculum. *ACM SIGCSE Bulletin* **33**(2):136-154.
- Nielsen, J. (1999): When Bad Design Elements Become the Standard. *Alertbox*, November 14. Accessed at <http://www.useit.com/alertbox/991114.html>, August 2006.
- Neumann, P. (1996): Risks-Forum Digest **18**(35), as accessed at <http://www.it.bton.ac.uk/staff/lp22/CP303/case/atlanta.html>, August 2006.
- Rob, P., Semaan, E. (2000): *Databases: design, development, and deployment using Microsoft Access* (international edition), McGraw-Hill.
- Seiden, J. (2002): Question and answer: design means remembering to ask the question. *Interactions* **9**(1):11-15.
- Winemiller, E., Roff, J.T., Heymann, B., Groom, R., (1998) *Visual Basic 6 Database How-To*, Sams Publishing.





# Simple Eight Bit, Emulated Computers for Illustrating Computer Architecture Concepts and Providing a Starting Point for Student Designs

**Timothy D. Stanley, Thanh Quach Xuan, Leslie Fife, Don Colton**

School of Computing  
Brigham Young University Hawaii  
Laie, HI 96762 USA

StanleyT@byuh.edu, xq003@byuh.edu, fifel@byuh.edu, ColtonD@byuh.edu

## Abstract

Students learn better when they both hear and do. In computer architecture courses “doing” can be difficult in small schools without hardware labs hosted by computer engineering, electrical engineering, or similar departments. Software solutions exist. Our success with George Mills’ Multimedia Logic (MML) is the focus of this paper. We have found that students learn and understand more, and experience less frustration, without the additional complexity of hardware details. MML provides a graphical computer architecture solution with convenient I/O support and the ability to build and emulate a variety of computer designs. It has proven highly motivational to upper-division computer science students designing and constructing emulated computers. Student projects resulted in excellent student understanding of the detailed inner workings of computers. Students also developed better teamwork skills and produced useful training aids for the lower-division computer organization class. Designs implemented include 8-bit and 16-bit, von Neumann and Harvard architectures, from single-cycle to twelve-cycle instructions. Issues resolved during the learning process include timing, initialization, instruction set architecture, I/O, and assembler design. We provide two demonstration computers used to illustrate to students a design approach and an expected outcome in their individual design activities. One example is an eight-bit Harvard architecture with eight instructions that execute in a single clock cycle. The second is an eight-bit von Neumann architecture that has four instructions and executes each instruction in three clock cycles. This paper describes these two example computers.

**Keywords:** Computer Organization & Architecture, Emulation Software, Pedagogy.

## 1 Introduction

Effective learning comes from doing, not just hearing. Computer Architecture is an area where doing is a natural

extension to the course material. There are many approaches to teaching a lab component for Computer Architecture. These include no lab at all, emulated hardware, actual hardware, and a mix of emulated and actual. Software tools range from register- and memory-level computer simulators to high-level chip design languages. Between these extremes is George Mills’ Multimedia Logic (MML). We discuss the use of computer simulators and chip definition languages. We then introduce Multimedia Logic and our approach.

However taught, Computer Architecture is an essential part of any computer science curriculum. In the periodic curriculum report issued by a joint task force headed by the ACM and IEEE-CS, Architecture and Organization is one of 14 core bodies of knowledge (CC 2001). The 36 minimum core hours in Architecture and Organization represents more than 12% of all core topic hours (CC 2001). While not binding, this curriculum recommendation is a good indication of the central place Computer Architecture occupies in the curriculum. The core architecture topics include machine level representation of data, memory system organization and architecture, and alternative architectures (CC 2001). The teaching of each of these can be enhanced through the MML software tool. For example, alternative architectures can be clearly shown. In our examples, both Harvard and von Neumann designs are shown.

### 1.1 Computer/Logic Simulators

Many computer architecture classes use emulators. The popular textbook “Computer Organization and Design” (Paterson, and Hennessy 1994) features SPIM by James Larus (Larus 2006). It shows memory contents and registers and includes an assembler but does not simulate the datapath.

Moving closer to our goal, many computer architecture classes use the text by Null and Lobur that introduces MARIE (2003a). They may also use MarieSim (Null, and Lobur 2003b). MARIE uses a von Neumann architecture with a simple instruction set. Using MarieSim, students can write their own programs and watch them execute, seeing the effect these programs have on system state. MARIE has been implemented with a “Data Path Simulator” that highlights registers and data paths visually. This shows the steps the processor goes through when running a program. However,

students cannot build their own computers and simulate them.

At the other end of the complexity spectrum is the implementation of a simulator of an entire real architecture. Clark, Czezowski, and Strazdins implemented a simulator for the Sparc V9 (2001). Even if it had a GUI, this is of limited usefulness for a typical undergraduate course in computer architecture. The level of complexity this embodies is not a good starting point for learning computer architecture. In addition, pedagogically, learning only one machine's architecture is limiting. The Alfa-1 simulator (Wainer, Daicz, Simoni, and Wasserman 2001) also works specifically with the Sparc processor. This tool allows the user to experiment with the entire architecture, including extending it in some ways. However, the limited user interface and the restriction to a single architecture are limitations. The Alfa-1 was designed to replace tools that simulated obsolete architectures. This approach ensures that an eventual replacement for Alfa-1 will be needed.

Some simulation tools address only a single problem. The KScalar simulator (Moure, Rexachs, and Luque 2002) provides a tool for learning about microprocessors. This is, of course, only part of what must be covered in a typical computer architecture course. However, this might be a good choice for an advanced course on microprocessors or for a few labs within a larger course. A similar approach is used by Holland, Harris, and Hauck (2003). They provide an incomplete processor and have students design the missing pieces. They can simulate any non-floating-point instruction in their 8-instruction MIPS processor. This creates an opportunity for students to learn processor details. However, the student is still limited to a single processor type without floating point. Other parts of the machine architecture still must be learned in some fashion. Another single-purpose tool is SIMT (Tao, Schulz, and Karl 2003). This simulator allows the evaluation of shared-memory systems. While these tools may be very useful in specialized computer architecture courses, having to learn several unconnected single-purpose tools takes extra effort in a general computer architecture course.

## 1.2 Chip Design Languages

Several chip design languages are available.

Logisim (Burch 2002) is another design and simulation tool. A basic package, Logisim allows the user to design circuits from basic logic components and some basic devices. The primary drawbacks are the lack of a clock for timing and the somewhat primitive I/O capabilities.

Logic Works 5 from Capilano Computing is a wonderful package, but the emphasis is on detailed timing and simulation of circuits to be exported to silicon (2006). It seems more suited to advanced students that already grasp computer architecture and are looking for the next step. It does not have the input and output devices available in Multimedia logic.

DLSim by Matthew Leslie was developed while an undergraduate student with additional development

funded by Cambridge (2006). It is an open source Java program available from <http://www.sourceforge.net/>. DLSim has the capability to use macros and can display logic states propagated through the circuits. But the devices are simply blocks and the rich set of IO devices available in Multimedia logic is not available.

A number of schools have used VHDL as a design medium for computer architecture. While VHSIC is used commercially for chip design, we believe it is too abstract to visualize and is too much like a software design to be physically satisfying.

## 1.3 Multimedia Logic (MML)

The package we chose is Multimedia Logic (MML), open source free software by George Mills (2006). MML strikes a good balance. We can design and implement logic at the gate level, but still use high-level I/O operations. (This is parallel to the C programming language. C provides near-assembler access to the machine, but still includes a standard I/O library in the basic distribution.)

We have seen how a variety of tools exist to simulate imaginary and real computers based on simple and complex instruction sets. Generally these tools allow the simulation of only a single architecture and they often hide many of the underlying details that we might like to reveal. You can write programs and see the results in various registers, but the computer itself remains a black box. Alternately, some tools allow the student to access the logic gate level of design but have limited I/O capabilities.

MML allows the student to define an Instruction Set and build the enabling architecture. This means the student is not limited to von Neumann computers or the instruction sets designed by others. With an excellent graphical user interface and ASCII I/O, the tool is easy to learn and use. Because MML is open source software, functions can be added and the tool recompiled. Despite being easy to use it is capable of sophisticated designs. For example, see the work of James Larson (2006).

Multimedia Logic is a very decent environment for virtual computer construction. We were able to implement all of the necessary components in order to run programs written for MARIE. The benefits of MML for our project were extensive. Ease of use compared to physical hardware is obvious. In addition, MML uses abstraction to simplify the hardware components it offers and reduces the need to build all of one's own components, such as an arithmetic logic unit.

## 2 Architecture Course Design

The philosophy of our computer architecture course is that students will only truly understand computer architecture when they design and build a computer. Students are shown how to build a computer through a couple of designs that are the focus of this paper. We demonstrate starting from an Instruction Set Architecture and continuing through the building of emulated hardware to implement the instruction set. After working

through these example designs, students were assigned to invent an original design, including an instruction set, and the registers to support the instructions. Our students are then required to implement their design in MML (Mills 2006). We have used this approach for three years now and students have produced some wonderful designs. Students feel empowered and are highly motivated to participate in the labs and survive the debugging process. They report having a great sense of accomplishment and achieving a profound understanding of how computers work at the logic level.

### 3 Example Computer One, the von Neumann Design

We start our discussion of MML with a “Hello World” example shown in Figure 1. This seems to be the first program shown in every programming language text, and it is fun to do it in an architecture course as well.

In a recent semester, to set the stage for the design assignment, the instructor provided as an example an 8-bit, accumulator-based, von Neumann design that uses three clock cycles to execute each instruction. The von Neumann architecture uses a single memory for data and instructions. To make this computer as simple as possible, but still able to demonstrate operation with useful programs, it was designed with four instructions. The instructions are: Load the accumulator from memory, Save the accumulator to memory, Add from a memory location to the accumulator, and Jump if the last add produced a zero result. These four instructions were supplemented by two memory mapped commands, Output on Save to memory location x3F, and Halt on Save to memory location x3E. Figure 1 shows the main page of this computer running a “Hello World” program. A second page of multiplexers is shown in Figure 2.

One of the most difficult challenges in a design like this is to develop an instruction decoder state machine in “Read Only” memory. This decoder takes as inputs a step count, the op-code and the zero flag, and provides as outputs control signals to set paths through multiplexers and enable writing to memory and registers.

One very nice feature of designs done in Multimedia Logic is the ease of attaching diagnostic displays to the computer. Displays include hexadecimal, binary, and

ASCII display terminal. Also available are text displays that provide one of sixteen text strings depending on a provided four-bit binary value. These text displays are used to show the current state of the computer in the execute cycle, and the instruction currently being executed.

Even though this computer has only four instructions they cover the basics needed to illustrate an accumulator-based von Neumann architecture.

One significant learning to come from this design is the elegance of self-modifying code to implement a program like the “Hello World” program shown. An effective indirect load can be designed in the von Neumann architecture by just recursively incrementing the load

instruction. This can lead to a discussion on self-modifying code.

In the Harvard architecture discussed next, with twice as many instructions, the “Hello World” program is a series of output commands for characters stored in the data memory because the program memory can not be modified by the running program.

### 4 Example Computer Two, the Harvard Design

An alternate example, developed three years ago, just redesigned to improve readability, is a single cycle, 8-bit, Harvard architecture described in the proceedings of the ACM Workshop on Computer Architecture Education (Stanley 2005). This design, shown in figures 3 and 4, uses two memories, one, read only, for program storage and a second, read-write, for data storage. It also has a read-only memory used to decode operation codes into control line states. This design also uses two ALU devices, one to increment the program counter and a second to execute mathematical instructions. Having two ALUs enables instructions to execute in one cycle, with the program counter incrementing while the other processes are also occurring. Also, this architecture can simultaneously access data and program memory eliminating the “von Neumann Bottle neck”. But since mathematical operations are on data memory, creating self-modifying code is not possible in this design. This makes some program tasks awkward. For example, the hello world program implemented in the Harvard design consists of a series of output commands.

One of the nice features of designs in MultiMedia Logic is the freedom to lavishly include display devices to show hexadecimal values, binary values, ASCII text, and value dependent comments, like the display “OutM” which is the mnemonic for the current instruction.

In this design, as with the von Neumann design, one of the most difficult challenges is designing the operation decoder, although for the Harvard design, this task is simpler since each instruction takes just one cycle. One nice capability with both designs is the ability to single step through instructions, by pressing the “Clock Pulse” button or have the computer run by lifting the “Enable Clk” switch.

### 5 Conclusions

These two designs, while not very advanced, have proven to be very useful for teaching principles of computer architecture. They have also served as a spring board for some very elaborate designs from our students. Some of the student designs include sixteen bit computers, hardware multipliers, a 128 bit key hardware encryption unit, and a fully multiplexed sixteen register array of sixteen bit registers. Also, since hardware is not used in these designs, students can take them with them at the end of the semester.

Some comments from students taking this class include the following:

“This project was very motivating for our team. We spent many hours to insure that a quality project was built. We learned a lot through the design and assembly as well as in debugging. Our fellow students also learned from our presentation of this project.”

Another student added:

“We appreciate for this new method of teaching the computer architecture. We think that we learn much more than if we physically build computers. I personally understand more how to build and design a computer from the ground up. Although our computer is in a logical form, it does represent our knowledge and our work. Instead of spending more time on connecting wire or circuit, our team actually spends more time on the primary elements in designing, planning, debugging, and understanding how our computer works.”

Patterson says “The processor comprises two components: data path and control...” (1994). These students know what that means because they have built both.

## 6 Acknowledgements

The authors gratefully acknowledge George Mills, the author of Multimedia Logic, for making his logic simulation package available without cost and including the source code on his web site, <http://www.softronix.com/>.

The authors are also deeply grateful to the students they have taught who have been the motivation for this development. Special thanks to Elliot Manning who was the team leader of the first student team and set the bar for the following classes.

## 7 References

Burch, C. (2002): Logisim: A Graphical System for Logic Circuit Design and Simulation. *ACM Journal of Educational Resources in Computing* 2(1): 5–16.

Capilano Computing. <http://www.capilano.com/>. Accessed 1 June 2006.

Clark, B., Czezowski, A. and Strazdins, P. (2001): Implementation Aspects of a SPARC V9. *Proc. 25th Australasian Computer Science Conference*, Melbourne, Australia, 23–32.

Computing Curricula 2001 Computer Science, Final Report, ACM/IEEE-CS Task Force, 15 December 2001. [http://acm.org/education/curric\\_vols/cc2001.pdf](http://acm.org/education/curric_vols/cc2001.pdf). Accessed 9 August 2006.

Holland, M., Harris, J. and Hauck, S. (2003): Harnessing FPGAs for Computer Architecture Education. *Proc. 2003 IEEE International Conference on Microelectronic Systems Education*, 12.

Larson, J. <http://www.dst-corp.com/james/MMLogic.html>. Accessed 1 June 2006.

Larus, J. SPIM, A MIPS32 Simulator, <http://www.cs.wisc.edu/~larus/spim.html>. Accessed 1 June 2006.

Leslie, M. <http://urchin.earth.li/~mleslie/project.html>, Accessed 1 June 2006.

Mills, G. Multimedia Logic, available for free download at <http://www.softronix.com>. Accessed 1 June 2006.

Moure, J., Rexachs, D. and Luque, E. (2002): The KScalar Simulator. *ACM Journal of Educational Resources in Computing* 2(1): 73–116.

Null, L. and Lobur, J. (2003): The Essentials of Computer Organization and Architecture. Sudbury, MA, Jones and Bartlett Computer Science.

Null, L. and Lobur, J. (2003): MarieSim: The MARIE Computer Simulator. *ACM Journal of Educational Resources in Computing* 3(2): 1–29.

Paterson, D. and Hennessy, J. (1994): Computer Organization and Design, The Hardware/Software Interface, Morgan Kaufmann.

Stanley, T. (2005): An emulated computer with assembler for teaching undergraduate computer architecture. *Proc. of the Workshop on Computer Architecture Education*.

Tao, J., Schulz, M. and Karl, W. (2003): A Simulation Tool for Evaluating Shared Memory Systems. *Proc. 36th Annual Simulation Symposium (ANNS'03)*.

Wainer, G., Daicz, S., Simoni, L. and Wassermann, D. (2001): Using the Alfa-2 Simulated Processor for Educational Purposes. *ACM Journal of Educational Resources in Computing* 1(2): 111–151.

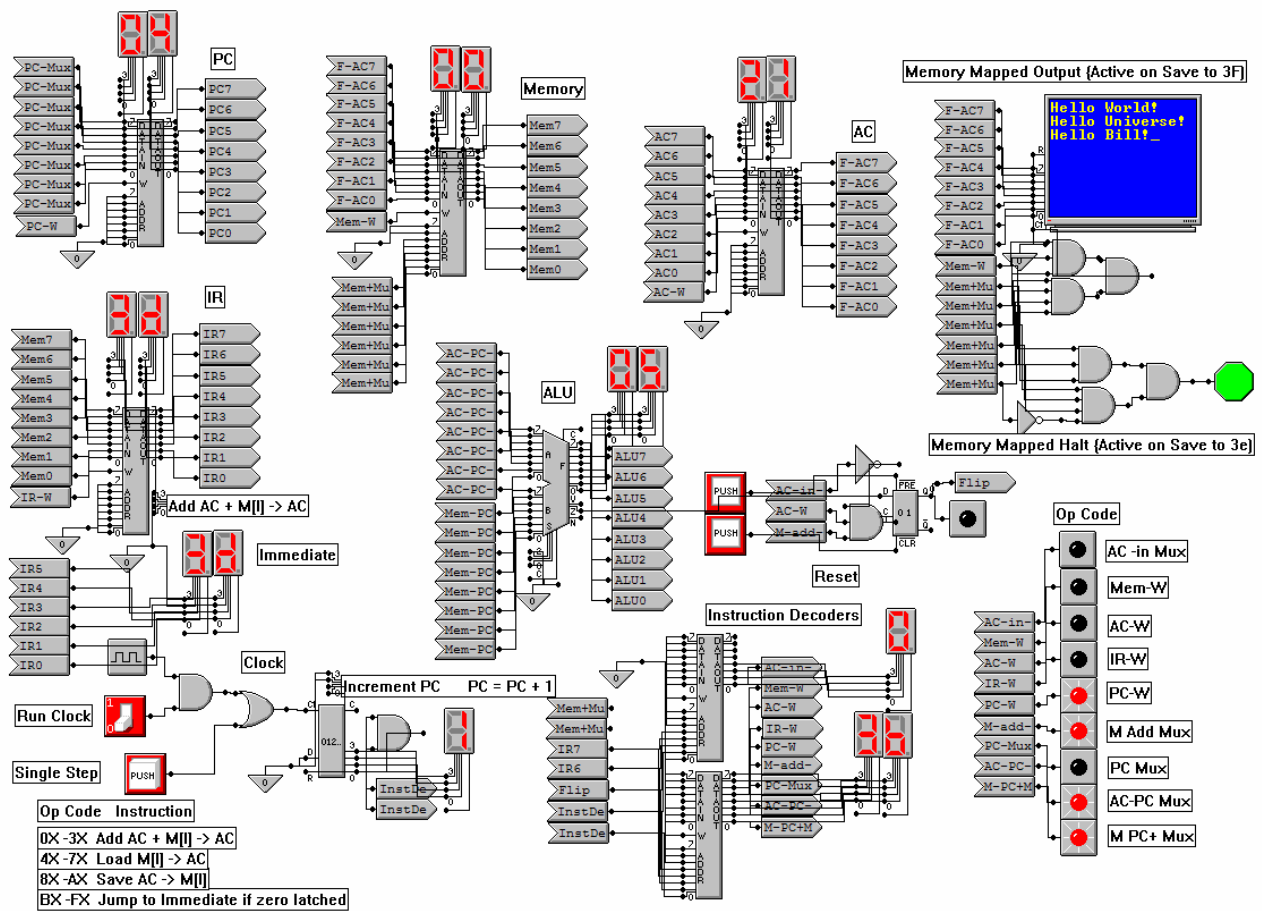


Figure 1 An example computer using an 8-bit von Neumann design

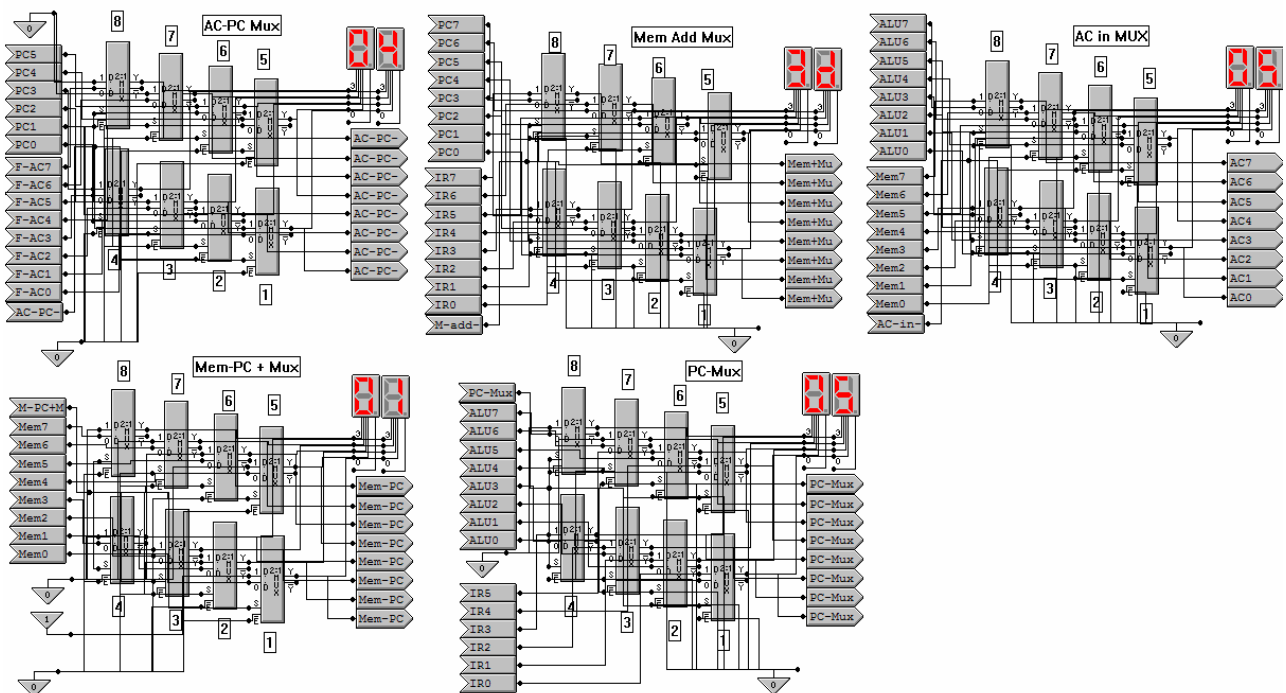
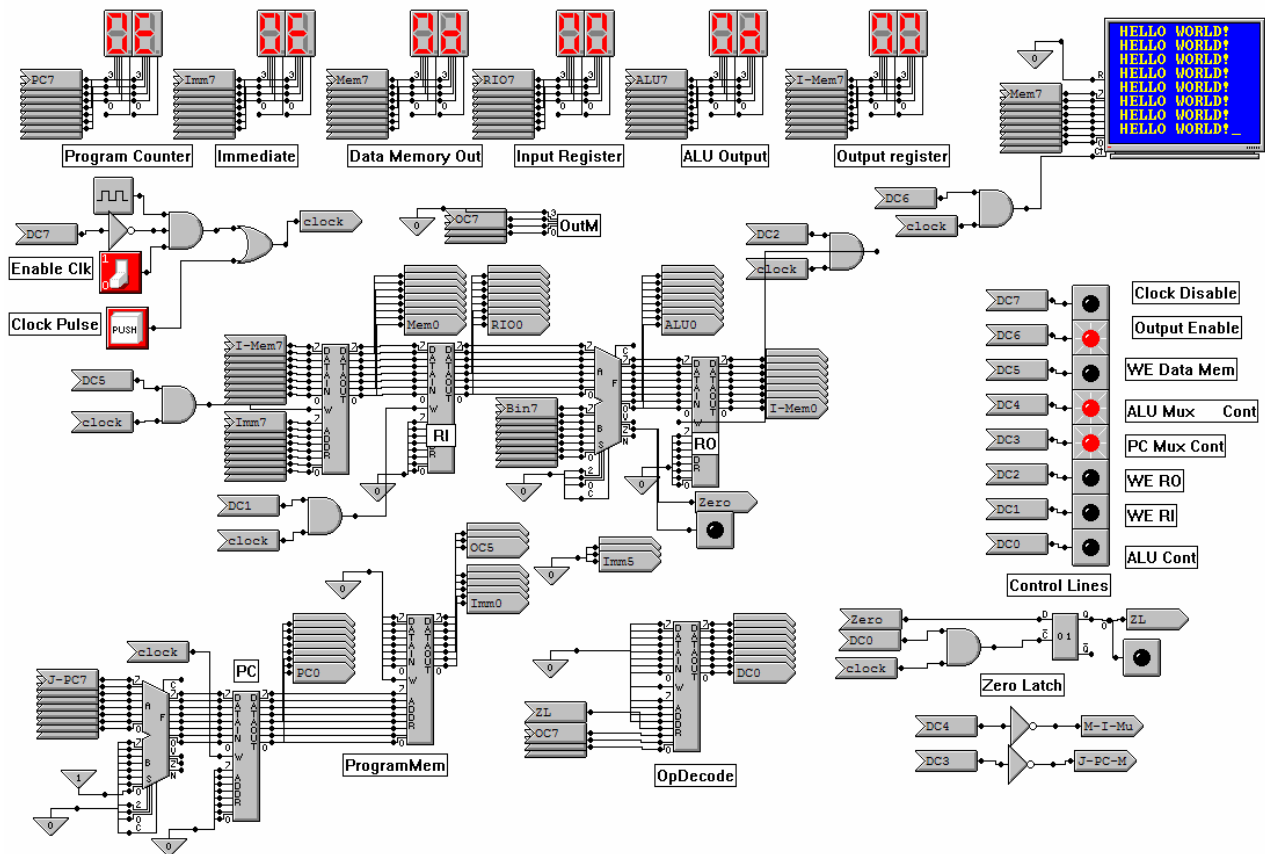
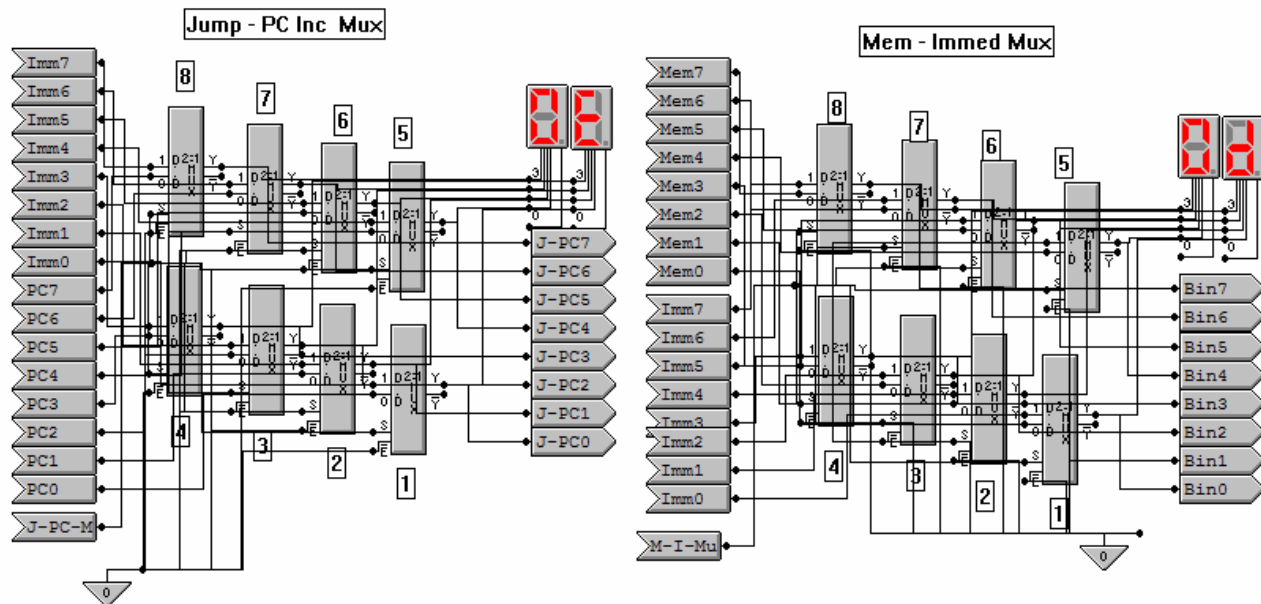


Figure 2 Multiplexer array for the 8-bit von Neumann design



**Figure 3** Example computer two, the eight bit Harvard design



**Figure 4 Multiplexer array for Harvard design**

# Evaluation of a New Assessment Scheme for a Third-year Concurrency Course

Paul Strooper

Larissa Meinicke

School of ITEE, The University of Queensland,  
Brisbane, Qld 4072, Australia,  
pstroop@itee.uq.edu.au      larissa@itee.uq.edu.au

## Abstract

In this paper, we describe and evaluate a change that was made to the assessment scheme for a third-year course on concurrent systems. The original assessment scheme consisted of tutorials, three assignments, and a final examination. The change was motivated by the fact that students often performed poorly, especially on the third assignment, and yet the average mark that they received for this and the other assignments contributed to what we felt were inflated course grades. In the changed scheme, the third assignment was made more complex and open-ended, but it was also made optional in that students can pass the course without having to do this third assignment. We evaluated the new assessment scheme by comparing the performance of students on Assignment 3 and the course for the last two years and by analysing the results of a student questionnaire.

*Keywords:* student assessment, concurrency.

## 1 Introduction

Lister and Leaney (2003) present and discuss an assessment scheme based on Bloom's taxonomy of educational objectives (Bloom et al. 1956) for a first-year programming course. The assessment scheme is motivated by the fact that in many first-year programming courses, the weaker students flounder and the stronger students are not challenged. To maximise the potential of students in such a course, Lister and Leaney argue that different assessment tasks must be used for students of different abilities and they use Bloom's taxonomy to motivate their choices. To pass the course, the students had to display knowledge and comprehension through lab exercises, a lab exam and a multiple choice exam. For a credit or distinction, students had to display application ability as well, through the completion of traditional, fairly well-defined programming assignments. Finally, for a high distinction, the students had to display the ability to synthesise and evaluate through an open-ended individual project and peer review of other students' code. Students were required to complete only the assessment tasks for the level of achievement they desired or were capable of, thus reducing the assessment load and leaving only a relatively small number of motivated students to attempt the individual project.

Box (2004) documents her experience in applying the assessment scheme used by Lister and Leaney to a second-year object-oriented programming course.

Copyright ©2007, Australian Computer Society, Inc. This paper appeared at Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, Australia, February 2007. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 66. Samuel Mann and Simon, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

Her motivation was that her class consisted of student at different skill and motivational levels, that some students felt they were not being challenged by the course material, and that many students were achieving inflated grades because of *grace* marks that they earned on assignments. Box does not provide a detailed evaluation of the course outcomes.

The first author has been involved in teaching a third-year course on concurrent systems since 1993. Up until 2003, this course was assessed in a fairly traditional way, incorporating an optional tutorial participation mark, three fairly well-defined assignments, and a final exam. However, this assessment scheme was unsatisfactory for a number of reasons:

- The students performed poorly on some of the assignments. This was especially true for the third assignment, which tended to be slightly more complex than the first two assignments and which was due at a busy and stressful time for students. The poor student performance on the third assignment also made marking this assignment a frustrating task for the teaching staff.
- Despite the poor performance on assignments, we felt that the marks for the course were inflated due to an overemphasis on relatively easy marks that could be collected by students on tutorial participation and assignments (even if students failed the third assignment, it was relatively straightforward to pick up some marks that would help them with their final grade).
- On the course and teaching evaluations, a number of (presumably stronger) students complained that the course was not challenging enough.

Motivated by the work of Lister and Leaney, a number of changes were made to the assessment scheme in 2004. The third assignment was replaced by a much more open-ended assignment, which was made optional. To do well in the course, the students had to complete all three assignments and perform well on the exam, and the tutorial participation mark was no longer used in the calculation of the final grade. For the lower grades, tutorial participation, the first two assignments, and the exam were taken into consideration, but the third assignment was not. Note that students had to decide whether to work on the third assignment during the semester, without knowing exactly how well they would end up doing in the course. However, we felt that it would not be a disadvantage to do the third assignment in terms of gaining a better understanding of the course material, even if the Assignment 3 mark did not directly count in the calculation of the final grade.

The anticipated advantages of the changes were:

- A more balanced and representative distribution of grades.

- A less stressful (but just as relevant) assessment scheme for weaker students.
- More challenging and interesting assessment tasks for stronger students.
- For teaching staff, a reduced marking load and more enjoyable marking experience for the third assignment. In addition, the more open-ended nature of the third assignment would make it much more difficult for students to plagiarise on this assignment.

In Section 2, we present a brief overview of the course and the assessment scheme that was used in the past. In Section 3, the new assessment scheme is presented, including a description of the third assignment. The evaluation of the new assessment scheme is discussed in Section 4.

## 2 Old assessment scheme

### 2.1 Course overview

COMP3402 *Concurrent and Real-Time Systems* is an introductory course on concurrent systems. Despite its title, only a small portion of the course is devoted to real-time systems. The course is typically taken by third- or fourth-year Information Technology or Software Engineering students, and can also be taken by Masters students (but typically the number of Masters students enrolled is small).

Since 2001, we have used the textbook *Concurrency: State Models and Java Programs* by Magee and Kramer (1999) in the course. The book takes a model-based approach to the design and implementation of concurrent programs, with models specified in FSP (a variant of CSP) that can be analysed using the LTSA tool and then implemented in Java.

### 2.2 Assessment

The contact hours consist of three lectures per week, one group tutorial, and one open tutorial. In addition, there are three assignments and a two-hour final exam. In the group tutorials, which form part of the assessment, students have to hand in an individual solution to a “warm-up” exercise (typically these are very simple problems, which the students should be able to solve in 5–10 minutes) at the start of each tutorial, and then a group solution to one of the problems on the tutorial handout. There is no tutorial handout for the open tutorials, which can be used to get assistance with any issues related to the course.

Assignment 1 involves modelling a fairly well-defined problem (in 2004, a taxi dispatcher and a number of taxis and customers) in FSP and then analysing it using LTSA. Assignment 2 is a programming assignment, in which the students have to implement a bounded buffer using semaphores (the semaphore implementation is given to them) and then to use these bounded buffers to implement a dataflow network. Assignment 3 is also a programming assignment, in which the students typically have to implement the system that was modelled in Assignment 1. Up until 2003, the design of the implementation and the detailed input and output was specified in the assignment handout, and parts of the solution were provided to the students so that they could focus on the “interesting” aspects.

combined mark	grade
85–100	7 (high distinction)
75–84	6 (distinction)
65–74	5 (credit)
50–64	4 (pass)
45–49	3 (conceded pass)
20–44	2 (fail)
0–19	1 (fail)

Table 1: Conversion from marks to grades

### 2.3 Determining grades

To calculate the student grades up until 2003, the three assignments contributed 10% each, the tutorial participation mark contributed 10%, and the final exam mark contributed 60%. If students performed better on the final exam than on tutorial participation (e.g., if they did not attend any of the group tutorials), then the final exam contributed 70% and the tutorial participation mark was not counted. The resulting combined mark was then converted to a grade as shown in Table 1.

### 2.4 Bloom’s taxonomy

In terms of Bloom’s taxonomy, the tutorial and the final (open-book) exam questions emphasise comprehension and application, with many questions asking the students about properties of small concurrent models or programs, and more complicated questions asking them to modify existing or write new (small) concurrent models and programs.

The assignments are relatively well-defined and address the students’ application and analysis abilities on medium-sized problems. Shneider and Gladikh (2006) have indicated that it can be difficult to consistently use Bloom’s taxonomy to determine levels in programming assignments. While it could be argued that the assignments contain some tasks at the evaluation and synthesis levels, it was felt that the tasks on the assignments are not challenging or creative enough for third-year students to be classified at higher levels.

As indicated above, there were a number of perceived problems with this assessment scheme, which motivated the changes introduced in 2004 and discussed below.

## 3 New assessment scheme

The main change to the assessment tasks in 2004 was that Assignment 3 was made more open-ended and a small component of its assessment was associated with the students evaluating their own design and implementation.

### 3.1 New Assignment 3

The following description is taken from the assignment handout (recall that Assignment 1 in 2004 involved the modelling of a taxi dispatcher and a number of taxi and customers): *For this assignment, you must design and implement a system that meets at least the requirements from Part 3 of Assignment 1, but extend them where appropriate (for example, your dispatcher should be able to queue many calls from customers when no taxis are available). One extension that you must make is to add time to the simulation so that the output shows at what point in time events (such as taxis going on/off duty, calls from*



customers to the dispatcher, etc.) occur in the simulation.

To encourage the students to start thinking about the assignment and to provide feedback on their initial thoughts, the assignment was split in two parts. In Part 1, the students had to describe the proposed user interface and the design. For the user interface, they had to describe the exact format of the input to the program and the output produced. They were encouraged to specify a GUI, but were not forced to do so. The design had to be described in terms of the classes and methods they planned to implement, as well as the concurrent aspects of the implementation.

Part 1 of Assignment 3 was actually due before Assignment 2 was due, and was marked on a pass/fail basis. If the student failed Part 1, then they had one week to resubmit it based on the feedback we had given them (and a general message that we posted to indicate common errors and problems in Part 1).

Part 2 of Assignment 3 consisted of the implementation of the system, an updated description of the user interface and design, a list of changes, and an evaluation of their own solution (e.g. describing any known flaws or deficiencies, what worked well, what they would do differently, etc.). The final mark for Assignment 3 was determined by their mark on Part 2, which was assessed based on the quality of the presentation (10%), the user interface (10%), the design (10%), the evaluation of their own solution (15%), the quality of the code (10%), a demonstration of their program (30%), and our overall impression of their “product” (15%). More detailed descriptors for each of these criteria were handed out to the students.

Part 2 was due in the penultimate week of the semester and all assignments were marked in demo sessions during the last week of semester. In the demo sessions, which typically lasted about 30 minutes, students were given the opportunity to briefly demonstrate their programs and then the markers asked questions about various aspects of their system (e.g. running through certain scenarios, questions about aspects of the code, etc.). To ensure consistency, all demo sessions were assessed with two markers present. Although all the assignments were assessed by the end of the week, it did take a lot of effort (certainly more effort than marking Assignment 3 in previous years).

### 3.2 New grading scheme

The main changes to the grading scheme in 2004 were as follows:

- Assignment 3 was made optional, but students had to perform well on it to get a grade of 6 or 7 on the course. Tutorial participation was not taken into account to determine if students should get a grade of 6 or 7.
- For grades of 5 or lower, Assignment 3 was not used in the calculation of the combined mark for the course, but tutorial participation could be counted for up to 10%.
- Cut-offs were introduced for the final exam mark (for most grades) and for Assignment 3 (for grades of 6 and 7).

In particular, there were two ways to calculate a combined mark for the course:

$$M1 = 0.15A1 + 0.15A2 + 0.6Ex + 0.1\max(Ex, T)$$

$$M2 = 0.1A1 + 0.1A2 + 0.2A3 + 0.6Ex$$

where  $A1$ – $A3$  represent the marks on Assignments 1–3,  $Ex$  represents the exam mark, and  $T$  represents

$M1$	$M2$	$Ex$	$A3$	grade
-	$\geq 85$	$\geq 75$	$\geq 75$	7 (high distinction)
-	$\geq 75$	$\geq 70$	$\geq 50$	6 (distinction)
$\geq 65$	-	$\geq 60$	-	5 (credit)
$\geq 50$	-	$\geq 50$	-	4 (pass)
$\geq 45$	-	$\geq 40$	-	3 (conceded pass)
$\geq 20$	-	-	-	2 (fail)
-	-	-	-	1 (fail)

Table 2: Grading scheme for 2004

the tutorial participation mark. Then the final grade was determined by the highest grade in Table 2 for which the component results satisfy all the criteria.

### 3.3 Bloom’s taxonomy

In terms of Bloom’s taxonomy, the change was intended to address the the synthesis and evaluation aspects of Bloom’s taxonomy through the modified Assignment 3, while maintaining the other assessment tasks (tutorials, first two assignments, and the final exam) to address the lower levels. To address the synthesis and evaluation aspects, Assignment 3 was left much more open-ended than the other assignments and included a small assessment component in which the students had to evaluate their own design and implementation.

Our new approach to assessment differs from the one adopted by Lister and Leaney (2003) and Box (2004), since we require all students, even those at a pass level, to be able to demonstrate application and analysis level skills. This difference is motivated by the fact that our assessment scheme, unlike that of Lister and Leaney (2003) and Box (2004), is for a third-year, not a first- or second-year, course. Students who are at a more advanced stage of their program should be capable of demonstrating these more sophisticated characteristics.

## 4 Evaluation

In this section, we present the results of comparing the students’ performance on assignment 3 and the course in 2003 and 2004. We also present the results of an in-class survey carried out in 2004. Finally, we present our own perceptions of the changes and the subsequent changes that we made in 2005.

The number of students enrolled in 2004 (52) was significantly lower than the number of students enrolled in 2003 (77). We believe that this is attributed to an overall drop in the number of IT and SE students and the fact that there was a time clash in 2004 with the lectures from a big third-year IT course that was discovered too late. In other words, we do not think that the new assessment scheme had a detrimental affect on student numbers and we did not see a higher than normal attrition rate in the course either. Unless otherwise noted, the frequencies shown in the bar graphs in this section are expressed as percentages, to facilitate comparison between 2003 and 2004.

### 4.1 Student performance on Assignment 3

Figure 1 shows the distribution of Assignment 3 marks in 2003 and 2004.

In 2004, 41 out of 52 (79%) students enrolled submitted Part 1 of Assignment 3. Only 16 of these passed and 25 failed. We felt that the two main reasons for this was that we had not given clear enough

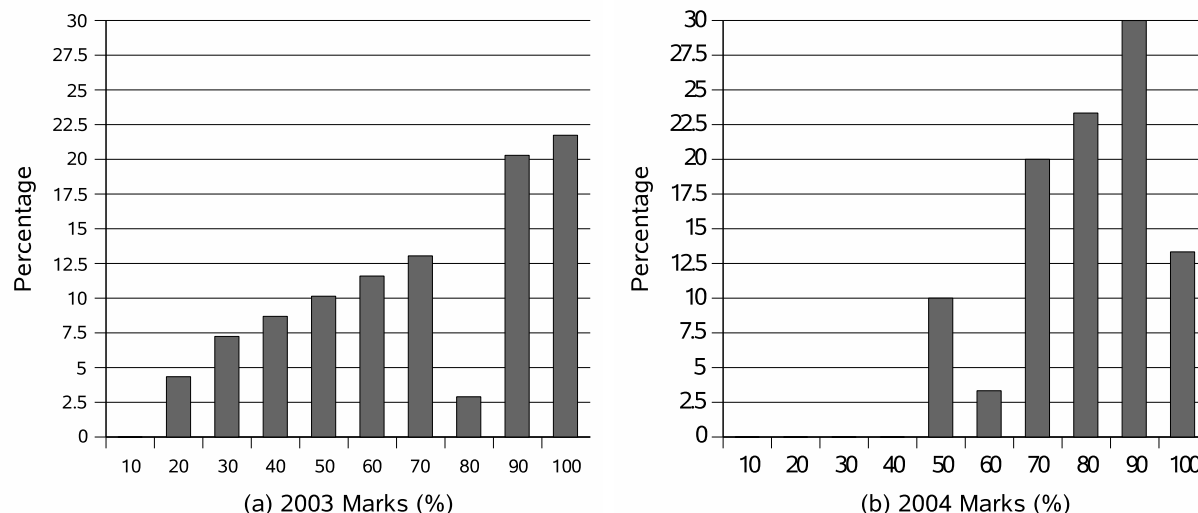


Figure 1: Assignment 3 marks in 2003 and 2004

instructions on what we wanted and a number of students had not spent enough time thinking about their user interface and design. In addition to individual feedback to the students, we posted a general message about the common errors and what we expected in the resubmissions. The students were then given a week to resubmit, and 22 out of 25 students resubmitted. Of these, 15 passed the second time and 7 still failed (2 of these resubmitted a second time and finally passed). This meant that in total 33 students passed Part 1 of the assignment. Of those, 30 students submitted Part 2 (this represents 58% of the students enrolled in the course). Note that these are the only students who received a mark for Assignment 3 in 2004.

In 2003, 69 out of 77 (90%) students enrolled submitted Assignment 3. Out of the 8 who did not submit the assignment, 3 submitted no assignments, 4 submitted only 1 assignment, and 1 submitted the other 2 assignments, so it is fair to say that most students at least attempted Assignment 3.

As can be seen from this data, students performed fairly poorly on Assignment 3 in 2003, which was one of the motivations for changing the assessment scheme in the course. Of the 69 students who submitted, 18 failed the assignment and many others performed poorly. Despite the fact that the assessment scheme in 2004 was tougher, the students who submitted Part 2 of the assignment did better than in 2003. Only one student failed, three students received marks in the range of 20–21 out of 40, and the others received a mark of 25 out of 40 or better.

We also compared the performance of the students in both years for Assignments 1 and 2, and for the exam, to ensure that there were no significant differences between the two groups. With the exception of Assignment 1, on which the students performed slightly better in 2003 (which we believe to be due to the requirements for that assignment being more straightforward), the distributions for the marks were similar in both years.

## 4.2 Student performance on the course

We present and discuss the distribution of grades in 2003 and 2004. In order to understand the differences in performance we investigate possible differences in groups by comparing their results on assignment 1, assignment 2 and the exam, and we examine how changing the marking scheme in various ways may have affected the resulting grades.

### 4.2.1 Distribution of grades

The distribution of student grades for the course in 2003 and 2004 are shown in Figure 2.

The 2003 data shows a relatively large number of students with a grade of 6 (and also with a 7), compared to the students with a grade of 4 or 5. As indicated earlier, we felt that these grades were inflated and at least partially caused by the students being able to collect relatively “easy” marks during the semester. The grade distribution for 2004 is more what we would expect to see from an average class. Note also that, as indicated above, the students performed similarly on the first two assignments and on the exam (for example, the average mark on the exam in 2003 was 36.8 out of 60, whereas in 2004 it was 34.8 out of 60).

### 4.2.2 Factors affecting grades

Since we made multiple changes to the assessment scheme simultaneously, we wanted to ensure that the change in grade distribution was not caused solely by the introduction of exam or assignment cut-offs or the fact that we no longer counted tutorial participation for a grade of 6 or 7.

We first checked what the grades would have been in 2004 if there had been no exam cut-offs. This would not have changed the grade distribution very much: there would be 1 less grade of 2, 2 less grades of 4, and 3 more grades of 5 (the same number of 1s, 3s, 6s and 7s). This is not surprising, as the exam cut-offs are really not that stringent. In fact, the exam cut-off of 75% for a grade of 7 is not really a cut-off at all, because it is impossible to get a combined mark of 85% or higher on the course if you do not get at least 75% on the final exam. An explicit cut-off was still included in the assessment scheme for simplicity and to indicate the importance that we attributed to the final exam.

We also checked the cut-offs for Assignment 3, but these did not make any difference to the grades for any of the students in the course.

Finally, we wanted to check what the significance of the removal of the tutorial participation marks for grades of 6 and 7 was. However, it was not clear how we could use the 2004 data for this, since it was not clear how the tutorial marks should be incorporated into the calculation for the final grade (drop the exam mark to 50%, reduce the weighting for one of the assignments, or some other way). Instead, we used the

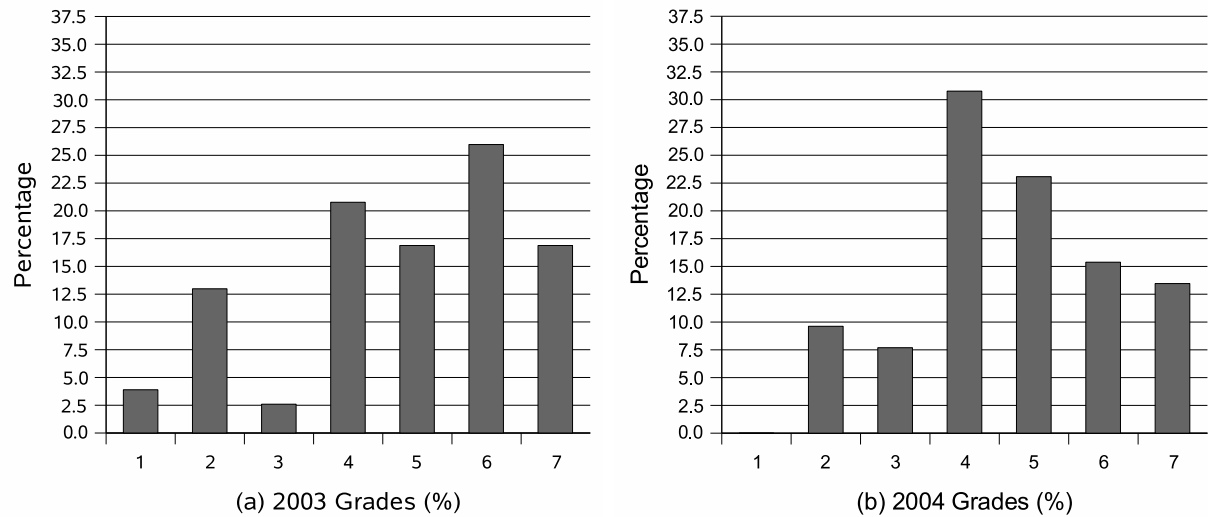


Figure 2: Distribution of grades in 2003 and 2004

data from 2003 and checked to see what the distribution of grades would have looked like if we would not have allowed tutorial participation to count for grades of 6 or 7 (thus to get a grade of 6 or 7, the exam was counted for 70% of the final mark). Although this change reduced the percentage of 6s and 7s from what it was previously, there still is a significant number of 6s in the course (close to 25%), so the change related to the tutorial participation mark does not seem to account for the difference in the grade distribution between 2003 and 2004.

#### 4.2.3 Factors affecting Assignment 3 choice

We had a closer look at the students who did not submit Assignment 3. One of the concerns with the new assessment scheme is that some “good” students may simply be too lazy to submit Assignment 3 and underperform as a result. We checked the exam marks of all the students who did not submit Assignment 3, and there was only one who did very well on the exam with a mark of 54 out of 60. The next highest mark for this group of students was 36 out of 60. The student who got a mark of 54 out of 60 also did well on the first two assignments and clearly could have received a 6 or a 7 on the course if he had submitted the third assignment. However, his combined mark for the course including Assignment 3 (where we counted his mark for Assignment 3 as 0) was only 74%, which is only a grade of 5. In other words, the marking scheme did not penalize him, but clearly it may have discouraged him from attempting the third assignment.

We also checked that for those students who did Assignment 3, but still only received a grade of 4 or 5 on the course, if it would have made a difference if we had allowed them to use the second marking scheme ( $M_2$  in Section 3) for grades lower than 6. There were two students whose combined mark  $M_2$  was higher than  $M_1$ , but for both these students  $M_2 = 61$  and  $M_1 = 60$ , so it did not make a difference in the grade that they would have received.

A number of students commented to us after their Assignment 3 demonstrations that they really enjoyed the experience of working on a more open-ended assignment. It allowed them to “get into trouble” and then they had to find their own way out. A number of students also commented that they felt that they had a much deeper understanding of the problems associated with concurrency. Most importantly, however, some of these students did not think they

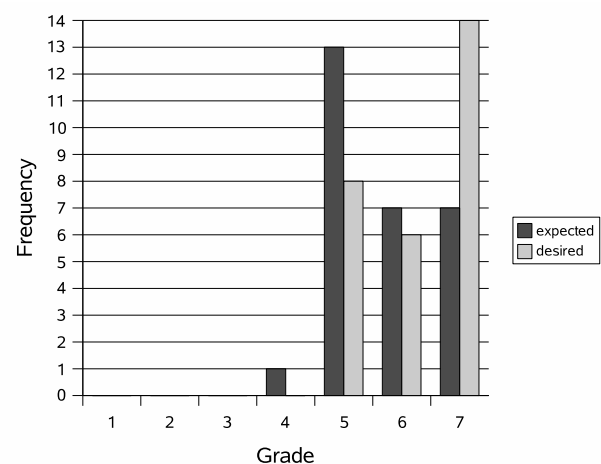


Figure 3: Expected and desired grades for 2004

would get a 6 or 7 in the course, but still appreciated the experience of working on the third assignment.

#### 4.3 Student survey

In the last week of semester, which was before most of the Assignment 3 demos, we carried out a survey to obtain feedback on the new assessment scheme. Out of the 52 students in the course, 28 completed the survey.

The first part of the questionnaire asked if the students completed Part 1 and/or Part 2 of Assignment 3. If not, they were asked why. Out of the 28 students who completed the survey, 24 had completed Part 1 and 21 had completed Part 2. The main reasons for not handing in the assignment were lack of time and failing Part 1 (or a combination of both).

Since students have to decide whether or not to work on Assignment 3 before they knew how well they would do in the course, we were interested to find out how accurately students could predict their final grade (which could influence their decision whether or not to do the assignment). So we asked students about the grade that they expected to get in the course and the grade they were hoping to get. This data is shown in Figure 3 (note that in this case, the vertical axis represents the number of students, and not the percentage of students).

	SD	D	N	A	SA
I think this year's scheme is fair	0	2	8	14	4
I like this year's scheme	0	7	6	11	4
I think last year's scheme is fair	0	0	6	16	6
I like last year's scheme	0	1	7	12	8

Table 3: Student rating of old and new assessment schemes

We also asked the students if they were willing to identify themselves on the questionnaire, so that we could compare their predicted with their actual grades (22 out of 28 students allowed us to do this). When we compare the expected with the actual grades, one student overestimated their final grade by 3 (expected a 5 and received a 2), another one by 2 (expected a 7 and received a 5), and all others were within 1 grade (8 overestimated by 1 grade, 10 accurately predicted their grade, and 2 underestimated by 1 grade). This data suggests that most students were able to predict fairly accurately how well they will do in this course before knowing how well they did on Assignment 3 or the exam.

The last two sections of the questionnaire asked the student to rate the assessment schemes in 2003 and 2004 (the 2003 assessment scheme was explained on the questionnaire). For both schemes, we asked them to rate their level of agreement for the following statements:

1. I think the assessment scheme is fair, and
2. I like the assessment scheme.

For both questions, we used a five-point scale: strongly agree (SA), agree (A), neutral (N), disagree (D), or strongly disagree (SD). In addition, they were asked to provide comments on either of the assessment schemes, or any other comments related to assessment.

Note that while it is normally not reasonable to ask students to evaluate an assessment scheme to which they have not been exposed, we felt that this was not an issue in this case because the old assessment scheme is a standard assessment scheme that is used in a number of our courses.

The data for the ratings of the above statements is shown in Table 3. The students rated the old scheme higher both in terms of thinking it is fair and liking it. The difference in the ratings is somewhat bigger for the issue of liking the assessment schemes than it is for rating the fairness. However, even for the new scheme, 18 out of 28 (64%) students agreed or strongly agreed with the statement that they thought the scheme was fair, and 15 out of 28 students (53%) agreed or strongly agreed that they liked it. A more detailed analysis of the data reveals that 7 students thought the new scheme was fairer than the old, 14 thought the old scheme was fairer, and 7 rated them the same. Similarly, 6 students liked the new scheme better, 15 liked the old scheme better, and 7 rated them the same.

The following is a summary of the most interesting general comments by the students. The students could write any comments they wanted, and several students included multiple comments. Below we paraphrase and group their comments, so that we can indicate the number of students who included similar comments in their feedback.

- 5 students liked the more open-ended and challenging nature of Assignment 3;
- 3 liked the flexibility offered by making Assignment 3 optional;

- 4 thought Assignment 3 should have been defined better;
- 3 thought Assignment 3 should also be counted for grades lower than 6;
- 3 did not like the exam cut-offs;
- 1 thought the exam was weighted too heavily;
- 3 thought it was (too) hard to achieve a 6 or a 7 with the new scheme;
- 4 thought tutorial participation should count for all grades; and
- 2 thought tutorial participation should never be counted.

#### 4.4 Own evaluation

The main concerns we had with the changes were the large number of students who failed Part 1 of Assignment 3 and the fact that some of the user interfaces in the final submissions were hard to use and comprehend.

We were satisfied with most other aspects of the changes:

- many of the students who did the third assignment seemed to enjoy the experience,
- although it was even more time-consuming to mark the third assignment than in 2003, it was more enjoyable because of the interactive nature of the demo sessions and the fact that the standard of solutions was better, and
- we felt that the distribution of final grades better reflected the students' ability than in previous years.

#### 4.5 Changes for 2005

We continued with the new assessment scheme in 2005 with the following changes.

1. We dropped the cut-offs for Assignment 3, because they did not seem to make a big difference in the final grades.
2. We also allow students to count  $M2$  (and thus Assignment 3) for any grade, not just for grades of 6 and 7. As indicated above, this would not have made a difference in 2004 because only 2 students had a slightly higher mark  $M2$  than  $M1$ , but it could make a difference in theory. Moreover, this addressed a concern raised by some of the students and we also felt that by making this change, it was less likely that the assessment scheme would discourage students from attempting Assignment 3 because it would not count for them anyway.
3. In response to a requirement from our University to distinguish the assessment requirements for undergraduate and masters courses, we always used marking scheme  $M2$  to calculate the final grade for masters students. We thus always counted Assignment 3 for masters students, and we did not consider tutorial participation in their final grade.
4. For Assignment 3 itself, we provided more detailed instructions for Part 1 the assignment, including a template document to clarify the type of information that we were expecting. We also increased the portion of the assessment attributed to the user interface from 10% to 15%

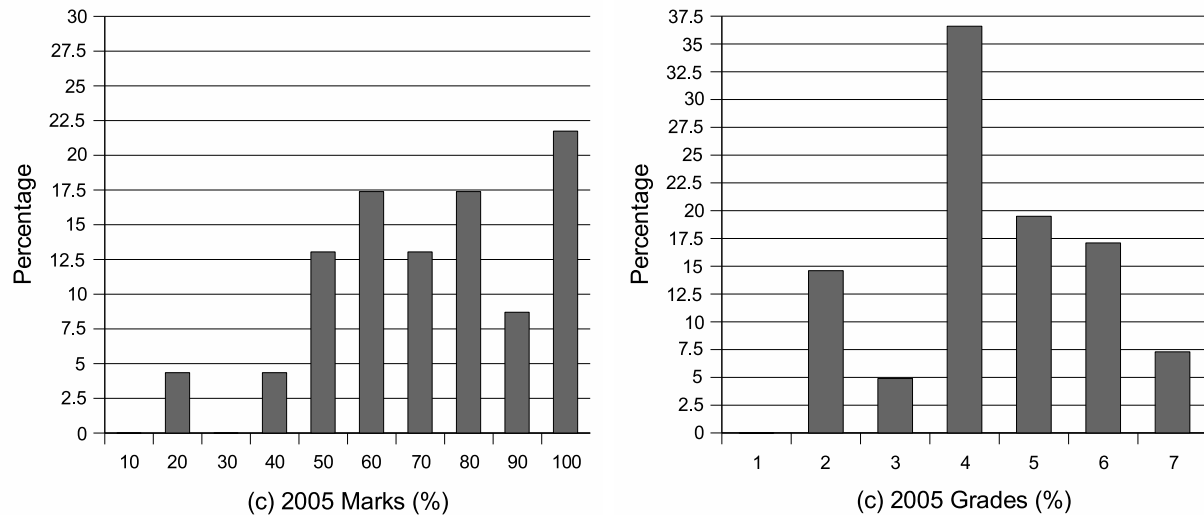


Figure 4: Assignment 3 marks and distribution of grades in 2005

and correspondingly decreased the proportion attributed to the students evaluation of their own solution from 15% to 10%.

Despite the comments by some of the students, we kept the exam cut-offs because we felt they allowed us to ensure that students achieve at least a certain standard in the course (e.g., for a grade of 4 or 5, up to 40% of a student's combined mark  $M1$  consists of their marks on the first two assignments, on which students can "collaborate" extensively, and the tutorial participation mark). Similarly, we continued with not counting tutorial participation in the  $M2$  mark, because we felt that it is reasonable for grades of 6 or 7 to not include any "easy" marks.

#### 4.6 Results from 2005

We did not perform a student survey or a detailed analysis in 2005, but we briefly discuss the results from the 2005 version of the course below. The distribution of marks for Assignment 3 and the distribution of final grades for the 2005 version of the course are shown in Figure 4.

In 2005, there were 41 students enrolled in the course, 33 undergraduate students and 8 masters students (compared to 52 in 2004, of whom 6 were masters students). Of those, 29 submitted Part 1 of Assignment 3, 26 of whom passed the first time and 2 passed after a resubmission, for a total of 28 students who passed Part 1. This compares favourably to the 2004 data, when only 16 out of 41 students passed Part 1 the first time and 33 out of 41 students passed eventually. Of the 28 students who passed Part 1, 23 submitted Part 2 of the assignment (56% of the students enrolled, compared to 58% in 2005). The average assignment mark in 2005 was 68.5% compared to 76.2% in 2004. Part of the reason for this was that 5 out of 23 students failed the assignment, including 2 out of 8 masters students (one of whom received a mark of only 8 out of 40). Part of this may be attributable to the fact that masters students were required to submit Assignment 3 as part of their assessment.

The overall performance of the students in 2005 was similar to that in 2004. The average mark on the final exam was similar (35.3 out of 60 in 2005 compared to 34.8 out of 60 in 2004), and the grade distribution for the course in 2005 was also similar.

We again checked if there were students in 2005 who we thought could have done much better if they

had chosen to do Assignment 3. There were 3 students who scored at least 40 out of 60 on the final exam and who had not attempted Assignment 3. Of those, one student would have received a grade of 6 if they had just passed Assignment 3, the second student would have only received a grade of 6 if they had received full marks (40 out of 40) for Assignment 3, and the third student would not have received a 6 even with full marks for Assignment 3.

Based on these results, we are currently going ahead with the same assessment and grading scheme in the 2006 version of the course and a similar scheme was used in a third-year Compilers course at our university.

#### 5 Concluding remarks

We have motivated, presented, and discussed the change in the assessment scheme for a course on concurrent systems. The main changes were making the third assignment much more open-ended and optional, using different ways of combining marks for different grades, the introduction of cut-off scores for the exam and third assignment, and not counting tutorial participation for grades of 6 or 7. In terms of the anticipated advantages discussed in the introduction, we feel that the new assessment scheme has resulted in a more balanced and representative distribution of grades, the feedback of both weaker and stronger students on the third assignment has been positive, and although the scheme has not resulted in a reduced marking load, the marking has become more enjoyable (and we are fairly certain that there was no problem with plagiarism on the third assignment). The student feedback on the overall assessment scheme was not quite as positive, but we did not feel that the criticism was sufficient to warrant a substantial change. As a result, we continued with the new assessment scheme in 2005 and 2006 with only a few minor changes.

#### Acknowledgments

We would like to thank Margaret Wojcicki and the anonymous referees for their valuable feedback on earlier versions of this paper. We would also like to thank the students enrolled in the course for their feedback on the assessment schemes.

## References

- Bloom, B. et al. (1956), *Taxonomy of educational objectives: the classification of educational goals — Handbook 1: Cognitive Domain*, David McKay Company.
- Box, I. (2004), Object-oriented analysis, criterion referencing, and bloom, in 'Proc. of the 6th Conference on Australian Computing Education', pp. 1–8.
- Lister, R. & Leaney, J. (2003), First year programming: Let all the flowers bloom, in 'Proc. of the 5th Australasian Computer Education Conference (ACE2003)', pp. 221–230.
- Magee, J. & Kramer, J. (1999), *Concurrency: State Models and Java Programs*, John Wiley & Sons.
- Shneider, E. & Gladkikh, O. (2006), Designing questioning strategies for information technology courses, in 'Proc. of the 19th Annual Conference of the National Advisory Committee on Computing Qualifications (NACCQ)', pp. 243–248.

# Holistic assessment criteria – Applying SOLO to programming projects

**Errol Thompson**

Department of Information Systems  
Massey University  
PO Box 756, Wellington 6140, New Zealand  
E.L.Thompson@massey.ac.nz

## Abstract

How you define your assessment criteria should influence the way the students approach the assignment. Does this mean that if we use a holistic criterion-based assessment strategy that students will look more holistically at the topic rather than focussing on the pieces for which they think they can gain satisfactory marks? A holistic set of assessment criteria for programming assignment work based on the SOLO taxonomy is presented, and reflections on the use of this approach over three years are discussed.

**Keywords:** Assessment, programming, criterion-based assessment.

## 1 Introduction

The initial moves to a holistic approach to grading for programming assignment work was reported in Thompson (2004) where the holistic approach was compared with a scoring / weighting rubric (Maki, 2004). The institutions, in which this work was carried out, used criterion-based strategies for essay and report writing assignments that endeavoured to assign higher grades for critical thinking. There was difficulty translating these to programming assignment work where the emphasis seemed to be on satisfying the required functionality according to a set of predefined programming standards.

The SOLO taxonomy (Biggs and Collis, 1982) provided a solution. In his book on improving learning in the university context, Biggs (1999) provided an example of how he applied the taxonomy to an essay style assessment. This sparked a number of trials of different SOLO based grading criteria for both essays and programming exercises in the context of object-oriented software development. These trials were completed with second and third year papers.

Biggs (1999) focuses extensively on the use of the SOLO taxonomy in university education. He contends that the strategy led to essays that integrated knowledge rather

than simply focussing on a single aspect or providing a shopping list of concepts relevant to the topic. Biggs' criteria seemed to provide a better criteria than the concepts of width, depth, and distance that had been used for assessing learning journals (Thompson, 1998, Thompson, 1997, November, 1996, November, 1997). Using learning journals in a distance education course revealed that some students were capable of integrating knowledge while others simply rewrote material from references or notes with limited reflection or integration.

In examining programming assignments, it was possible to observe the shopping list style answer approach to writing code. The criteria for these assignments didn't encourage code reuse or have any emphasis on the structure of the code. The program needed to be syntactically correct, implement a required set of functionality, and utilise reasonable programming practices. Although the marking criteria included the ability to use modules (i.e. subroutines and functions), this didn't lead to the students integrating code or reducing code duplication. Modules tended to be large and perform multiple tasks. The use of documented testing strategies was also given little importance by the students with often buggy and incomplete code for all functionality being handed in for marking. Students appeared to see programming as an exercise in completing as much of the required valid data path functionality without concern for the overall integrity or quality of the product.

The criteria described in this paper attempts to address these issues by drawing on the concepts underlying the SOLO taxonomy. Other experience with using the SOLO taxonomy in evaluating responses to program reading questions of novice programmers is documented in Whalley et al. (2006), Lister et al (2006), and Thompson et al. (2006). In these papers, the SOLO taxonomy was selected to analyse the data gathered from a series of programming related questions. These questions were not written with the SOLO Taxonomy in mind.

This paper reports on the use of SOLO to define a set of criteria that are given to the students to influence their approach to the programming task and to assess the work that the students present for marking. Biggs (1999) and Hattie and Purdue (1998) describe this type of usage.

## 2 Methodology

The holistic marking criteria were originally introduced to papers in 2002 taught. They have been used for essay,

---

Copyright © 2007, Australian Computer Society, Inc. This paper appeared at the Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, Australia, January 2007. Conferences in Research in Practice in Information Technology, Vol. 66. Samuel Mann and Simon Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

design and programming assignments. These papers have been at all levels within a degree program.

The lecturer involved initially evaluated the marks obtained against the previous marking strategies used for these papers (Thompson, 2004). In that paper, it was shown how the strategy had limited impact on the grade of a good student (A+) but could cause a lower grade student (C - C+) to fail unless they changed their approach to the assessment. The lecturer concerned argues that this is precisely the outcome that was wanted from the holistic criteria.

This paper endeavours to use qualitative data drawn from the lecturer's journal to identify whether the students' approach to the assignment was changed as a result of using the assessment strategy? The lecturer recorded key questions raised by students about the assignments and marking strategies. These were recorded following the lecture or laboratory sessions.

### 3 SOLO categories

The SOLO taxonomy (Biggs and Collis, 1982) provides a series of categories based on the structural relationship of the material being presented. The categories of the SOLO taxonomy are defined as:

Category	Description
Prestructural	a) Misses the point of the exercise or plagiarises from other material. b) The presented information has little or no relevance to the requested requirements.
Unistructural	a) Focus on one conceptual issue or naming things. b) Shows minimal understanding by only giving serious consideration to one feature or requirement.
Multistructural	a) List of items but no relationship between items. b) The emphasis here is on "knowledge-telling" (i.e. look at how much I know).
Relational	a) Shows understanding through integrating concepts and ideas. b) Understands how to apply the concept to a familiar problem.
Extended Abstract	a) Relates an existing concept or principle in such a way that they are able to handle unseen problems. b) Questioning and going beyond existing principles

**Table 1: SOLO categories**

The prestructural category includes the criterion that the essay or code has to meet a minimum standard of presentation. This doesn't mean the assessor is looking for all the grammatical or spelling errors in an essay. An essay has to be able to be read in reasonable time without

the reader being caught by obvious grammatical problems. There are tools that the student should learn to use to help them verify the grammar or spelling. The issue is whether the information is presented in a way that makes sense to the reader. If grammar or spelling gets in the way then the student hasn't applied the basic tools that are available.

### 4 Programming SOLO categories

This section outlines the criteria based on those used in a second year programming paper that emphasised test-driven development and refactoring. The assignment was provided as three iterations over the twelve week period of the course. Each iteration added another feature that reused some of the existing functionality either explicitly, such as validation rules for input data, or implicitly through the need to use closely related processing structures or techniques such as access to a database. The iterations also tried to introduce a requirement for different programming constructs or techniques.

The objective of the paper was to introduce the students to the techniques of test-driven development and refactoring. They all had passed previous introductory and intermediate level programming papers. Limited emphasis was placed on learning new language or framework features.

In the criteria for the assignment, some introductory notes were included to emphasise the different focus used in the assessment criteria. These notes were:

- 1) Exceeding the minimum requirement in one area for a grade will not see a higher grade awarded. You must show that you are applying all the principles consistently to be awarded a higher grade.
- 2) Features in the following criteria relates to all aspects of the assignment. That is, the programs required functionality, the design of the user interface, and the implementation of an automated testing strategy.
- 3) Programming standards include the use of good code layout, variable names, and the elimination of code duplication. The code should be readable with the minimum of internal comments. That is, it should be self documenting.
- 4) Form design should endeavour to be consistent with the conventions of windows based applications.
- 5) The tab sequence on the form should follow a logical pattern.

The first of these notes was intended to discourage the students focussing on implementing all the functional requirements and ignoring the requirement to use test-driven development and refactoring. Simply completing more functionality would not gain a better mark if the required practices were ignored. This was further emphasised by the second note.



As well as the program code, the students were asked to provide a document that described the reasoning for their design. They were not encouraged to produce external detailed design documentation but were encouraged to use coding standards that promoted readability and self-documentation of code. The emphasis on internal documentation was placed on comments that would help explain why a particular programming approach was used rather than a simple description of what the code was doing. Comments that simply stated the obvious (i.e. adds the two values together) were discouraged.

#### 4.1 Base Standard

It was also stated in the assignment brief that programs that did not compile or run would be returned without being marked. It was not the marker's task to fix such problems. As a programming paper, the students had access to a compiler to validate syntax and were being encouraged to focus on incremental development rather than attempting all functionality and getting none of it working. There should be no obvious faults in the submitting code. The emphasis in the assignment instructions is to have completed and tested features rather than to have started all the features but have few of them completed.

#### 4.2 Inadequate work (Prestructural)

This category could also be defined as absolute fail (E grade) in terms of the assigned grade. In line with the SOLO taxonomy, a student graded into this category was showing that they did not understand the task or what was expected of them.

##### 4.2.1 Issues

The primary focus of this category was that the student either showed inadequate knowledge or had completed an inadequate amount of work. In line with the concepts of plagiarism in essay writing, this category included simply copying example code. Such copying would normally produce a program that failed to deliver the working functionality unless the plagiarism involved copying another student's code. During the lectures, code examples were given out that illustrated a range of solutions to particular types of coding problems and had been used to discuss good coding practice and possible design options for different aspects of coding business applications. At no point were the students given a full system solution so any copying of solutions would involve the selection of sample code that would not adequately combine into a solution.

As well as ruling out copying, these criteria specified the minimum standard required to be considered as making a serious attempt at the programming exercise. The target of 30% represented slightly less than one iteration for the assignment. Inadequate progress was seen as being an indicator of an inability to tackle the programming tasks.

A minimum standard was also presented for coding, user interface design, and application structure. If they were

given minimum consideration by the student then the assignment submission was dismissed as unacceptable.

##### 4.2.2 Criteria

The inadequate work (prestructural) criteria were defined as:

*Copying code or no understanding of programming issues.*

- Application attempts to copy example code with minimal changes
- Application is unrelated to requirements
- Application delivers less than 30% of the required functionality.
- No attempt has been made to apply programming or user interface design standards or good application structures.

#### 4.3 Single aspect (Unistructural)

This category was considered a marginal fail (D grade). The student showed some understanding but was operating an inadequate level to be able to participate in a programming environment.

##### 4.3.1 Issues

The reason for a student's work to be graded in this category was that they had focussed on one aspect of the assessment. This might have been that one feature of the system had been implemented or that one aspect of the required implementation tasks was utilised. For example a student may have concentrated on the design of the user and ignored the implementation of processing logic or a testing strategy. If a student completed all the functionality according to the specifications but wrote no automated tests or documented no testing strategy then this was regarded as focusing on a single aspect of the assignment task.

In attempting to define this category, two dimensions were taken into account. These were the features or scope of the system and the range of programming techniques or constructs that should be used. Focussing on only one iteration or feature set of the system was regarded as a single aspect. Likewise, focussing on a single programming technique or construct was regarded as a single aspect.

Because of the way that the iterations were defined, students who only completed one iteration or who were only beginning the second iteration were unlikely to demonstrate anything other than a single aspect approach to the assignment.

##### 4.3.2 Criteria

The single aspect (unistructural) criteria were defined as:

Shows a *limited understanding* of programming and application development issues. Parts of features are implemented without ensuring successful operation. Some of the issues considered include:

- Application partially operates with significant obvious problems.
- Application delivers 30-50% of required features as specified in the requirements.
- Programming standards, application structures, and user interface design standards are not applied consistently.

#### 4.4 Disjoint project (Multistructural)

The greatest grade range was assigned to the disjoint project (multistructural) category. In defining this category, the grade band was split into two categories.

The lower band of this category represented those students who were only just making the standard in more than one aspect of the project. They were assigned a grade that would give them a marginal pass for the assessment (C grade). These students might be able to participate in projects where they can focus on a specific aspect and not have to deal with all aspects of a project.

In contrast those in the higher band were endeavouring to satisfy the standards in most aspects of the project. However, they were not seeing these aspects as related or integrated. Each was handled as though independent of each other. Many of these students would make good journeymen on programming projects where they could follow the lead of others. These students were assigned an adequate pass grade (B grade).

##### 4.4.1 Lower disjoint project band issues

The lower disjoint project band focussed on the use of a limited range of concepts and techniques or on implementing limited functionality. If a student attempted two iterations in a manner where each iteration was considered as single aspects then their mark would fall into this lower category band. An attempt to complete more than one iteration was needed for the student to be graded above this band. One iteration if completed fully utilising all the required programming techniques and with automated test would be at the bottom end of this grade band. The issues of integration of code between iteration requirements would not be illustrated in the student's solution.

This lower category might also be represented in a project where the student had completed the full scope of the project but only partially addressed the testing requirements or user interface design issues.

##### 4.4.2 Lower band criteria

The lower disjoint project (multistructural) criteria were defined as:

Is able to *complete a working* piece of code to a base standard. The completed features are implemented but without ensuring consistency in operation or implementation. Some of the issues considered include:

- Application operates without obvious problems (i.e. does not crash when executed).
- Application delivers 50 to 60% of required features as specified in the requirements.

- Application inconsistently applies programming standards and user interface design standards.

##### 4.4.3 Upper band issues

In contrast to the lower disjoint project category band, the higher disjoint project category recognised that a large amount of the system functionality might be implemented but that there was limited or no integration of that functionality. At least two iterations of the assessment project would need to be attempted to be considered for a grade in this band. The project may have attempted all iterations but each iteration appeared in the presented work as if it was a totally independent programming project or included a high level of code duplication because the student had not seen the commonality that was possible in developing the solution. The required system specifications might have been fully met but the user has what appeared to be three totally independent projects.

##### 4.4.4 Upper band criteria

The higher disjoint project (multistructural) category was defined as:

Is able to *complete a working* piece of code to a base standard. The completed features are implemented as if they don't belong together. Some of the issues considered include:

- Application operates without obvious problems (i.e. does not crash when executed).
- Application delivers 60 to 70% of required features as specified in the requirements.
- Application consistently applies programming standards and user interface design standards.

#### 4.5 Unified project (Relational)

This is the level that we wanted most students to operate at. These students were assigned a grade that indicated their high level of competency as a programmer (A grade). At this level, they were seeing the relationships in what they were attempting to achieve although not stretching beyond their current knowledge. It would be expected that students operating at this level would be able to deal with most programming tasks that utilised familiar techniques or practices. They might struggle where novel solutions were required.

##### 4.5.1 Issues

In this category, the student has completed the assessment work to a level that shows integration of the iterations and utilisation in a coordinated manner of a wide range of programming techniques and constructs for the intended purpose. Two iterations of the project needed to be completed and a portion of the third iteration attempted in order for the student to be considered in this grade band.

At this level, the student has identified the commonality in the functionality of the code and has refactored the code to eliminate duplication. The student has also recognised the need to apply all the programming techniques consistently to achieve the projects objectives.

#### 4.5.2 Criteria

The unified project (relational) criteria were defined as:

Is able to *apply* the programming concepts taught and consistently *uphold* the standards and structures from example code. The completed features are implemented in a way that demonstrates the integration of ideas and of the system being developed. As above plus:

- Application delivers over 70% of the required functionality as specified in the requirements.
- Application structure is clean and matches standards.
- Application is documented externally to ease understanding.
- Minimal duplication of code and good reuse between different functional components.

#### 4.6 Outstanding work (Extended Abstract)

This category was reserved for those students who had exceeded the expectations for the assessment. These were the possible innovators and ideas people of a project. As such they were assigned a grade that reflects excellence (A+ grade).

##### 4.6.1 Issues

At the lowest end of this category would be the student who has completed all the iterations with fully integrated code and utilising the full range of programming techniques and constructs. Ideally, they should have used some constructs that were not explicitly taught in the paper. The focus here was on going beyond just doing the basics. The last iteration provided the students with the opportunity to extend their learning if they wished and to use some alternative techniques not explicitly covered in the paper but which they should have been able to learn based on the learning foundation given in the paper.

The outstanding student in this category would be using additional programming techniques and constructs, and have demonstrated an ability to argue for their inclusion in the project. The student was not expected to add functionality beyond what was requested but to show that they had evaluated and explored new techniques and constructs without formal instruction or guidance. They may also have demonstrated an ability to identify weaknesses in the requirements or design specification.

##### 4.6.2 Criteria

The outstanding work (extended abstract) criteria were defined as:

*Shows initiative* to experiment with new ideas and is able to *present a meaningful argument* for a revised approach. Achieves what is specified in an integrated way and addresses issues beyond those clearly stated in the assignment. As above plus ...

- Application delivers over 80% of the required functionality.
- Application design shows integration of task components.
- Utilises programming techniques and constructs that were not explicitly taught in the paper

- Documents reasoning for choice of approach to application design and coding.
- Documents task integration issues.

#### 4.7 General comment

The objective in writing the criteria presented here was to give the students a clear indication of the intent for each of the marking criteria and to provide some basis for them to check whether they had achieved a specific standard. Those operating at the outstanding work category level didn't need a checklist but those at the lower levels often seemed to need explicit checklists of what was expected.

### 5 Cases

The previous section introduced the concepts of using the SOLO taxonomy for defining the assessment criteria for programming projects. The criteria described are those used in a particular paper. They need to be adapted for each paper. In this section, the application of the criteria is discussed in relation to specific student cases. The described cases are based on experience over the last two years.

#### 5.1 "How can I be sure that I have done enough?"

This first case related to the difference between criteria that defined as a checklist or scoring / weighting rubric (Maki, 2004) and the holistic approach of the SOLO type categories. The students had phrased this as "how many marks will I get if I only do ...?" or "how can I be sure that I have done enough to get a .. grade?"

These questions still occurred with scoring / weighting rubrics but seemed to be even more pronounced when the SOLO taxonomy criteria are used. The scoring / weighting rubric provided a form of checklist which the students used to check off whether they thought they had done enough work. When pushed they accept that the SOLO categories had given them this information and that it was simply that it was not in a form that they are used to using.

This did lead to the protest that this was not how they were assessed in other papers. In other papers, they felt they could leave out practices required by the assessment. They contended that the criteria didn't give them flexibility in how they approached the task. To some extent this was true, but in reality they had considerable flexibility in how they created the solution and in the type of tests that they utilised. What they didn't have flexibility in was providing the required functionality and the proof that the functionality was working as required.

Part of the reason for this difficulty was that the SOLO taxonomy criteria were only being used in papers taught by one lecturer. Students were not familiar with this style of criteria or approach to assessment. Despite having talked through the criteria, the style of assessment, and the desire to help them understand professional practice, the students had difficulty understanding or accepting the expectations of the criteria.

The students were indoctrinated with the assessment practices of the department and institutional culture. This indoctrination wasn't simply for the criteria but for the nature of assessment and ability to negotiate changes to due dates, and criteria.

This problem wasn't directly related to the use of the SOLO taxonomy. It applied to the use of any alternative assessment strategy or assessment criteria. To use an alternative assessment strategy or criteria requires making the effort to assist students to understand the expectations.

## 5.2 “Do I really need to do that?”

This case had some similarities to the previous case except that this wasn't based on issues of inflexibility but more whether something was really needed. In the assignment for which the above criteria were used, the completion of a feature involved both the code and automated tests. This included elements of data validation. Under a conventional marking matrix, students knew that they could ignore the automated tests and still pass the assessment possibly with a very good mark. They also knew that as long as they had the primary processing path working (i.e. valid data) and testing for that path that they would pass in other papers. With the SOLO strategy, completion of functionality meant they needed to be more thorough in their work including dealing with invalid data in order to obtain the same grade. It was necessary to complete the full functionality of a feature and not simply what seemed to deliver the core functionality.

Students in this category either obtained a lower grade than they would have under a scoring / weighting rubric (Thompson, 2004) or they adapted to the new criteria and completed the additional tasks to obtain a similar grade.

## 5.3 “I need to implement this now”

When a new iteration was handed out there was in the students thinking a need to add new functionality even if the previous functionality was still incomplete and untested. The result often was code that was full of problems and faults. The more functionality they tried to add the more problems and faults occurred often leading to code that failed to run at all. The shift to ensuring the current iteration is complete before moving on had to be continually reinforced and in some cases demanded before help or assistance was given to solving the more difficult problems. This problem was also occurring with conventional scoring / weighting rubrics but students found it easier to ignore the problems and still pass.

This was less of a problem in an offering where individual iterations were taken in and checked for a satisfactory level of completion before the next iteration was given out. The marking strategy handled this situation with minimal adjustment since the student who could not complete the first iteration would not gain a pass grade. Not completing the current iteration put a ceiling on the mark obtained for the project and reinforced the objective of the marking strategy.

When iterations had to reach a certain standard before the student was given the requirements for the next iteration, this led to those students who are struggling to reach the standard protesting that they were being disadvantaged because they were not getting the opportunity to work on the next iteration. These students sometimes obtain copies of the next iteration from a student who had achieved the standard in the belief that if they implemented more functionality they would get a better mark. They failed to recognise the importance of applying key practices or reaching required levels of performance before moving on. They also failed to recognise that failure is part of the learning process. Steve McConnell (2004) says

“Great designers usually have experience on failed projects and have made a point of learning from their failures. They try out and discard more alternatives. They are often wrong, but they discover and correct their mistakes quickly. They have the tenacity to continue trying alternatives after others give up.”

Students who refused to accept that their work is not up to standard were failing to learn the important lessons that would enable them to become better programmers and to gain the freedom that they desired in programming.

A marking strategy based on assigning a portion of marks for each iteration still portrayed the idea that if they didn't meet the required minimum standard for this iteration, they could always pick up marks for the next. The message desired from the SOLO strategy is that each iteration must be completed successfully. This is an attitude that is portrayed in the velocity calculations of agile methods such as extreme programming where a story is not included in the velocity if it is not complete. If the student was not allowed to move forward to the next iteration until they have completed the current to the required level, then they were learning that work needs to be completed before it counts. The velocity count is all or nothing for stories. It isn't an estimate of “I have 50% complete so I am making progress”.

## 5.4 “I have all the functionality there”

This case relates to the students who completed all the iterations as though they were totally independent applications. In this situation all the functionality was there and was fully tested. The problem was that there was no recognition of common features (i.e. repeated validation) and duplication of coding occurred.

In the SOLO taxonomy grading system, this student only received a B grade and some protested that they should be receiving at least an A if not an A+. From a teaching perspective, the B grade was what they should have received because they were not integrating their work but the students failed to see the importance of this integration activity.

It is easy to blame the students for this style of thinking but sometimes it reflects the way that the students have been taught to program. The use of methods came after they were taught all the logic structures. It wasn't taught as part of the process of dividing your code into

functional components or for implementing duplicate logic. Rather it was simply a tool that you used when directed.

A similar attitude exists to the use of objects. Because the student has been taught procedural logic first, they see the problem in procedural terms rather than as interacting objects. When a student is asked why a form is overloaded with logic and why they have not created business objects that contain testable functionality, they respond that it was the way they have learnt to program.

Felleisen et al (2001) and Proulx and Gray (2006) propose teaching strategies that seem to reverse conventional wisdom but lead to students thinking in terms of class and method design ahead of logic design. This approach to teaching may address the issues raised by this case.

### 5.5 Testing isn't a programmer's responsibility

Test-driven development argues that the automated test should be written before the code is written. This does mean that the programmer needs to have an idea of how the program is to be designed before they write the test. Students took this a step further and argued that they didn't know what the test would be until they had written the code or that it was not possible to write tests for the code because the program could only be tested as a whole or that the programmer wasn't responsible for testing their code.

The result of this attitude was that automated test code was only submitted for a small segment of the code or the provided tests were superficial. Some functionality was implemented but not tested or there were failing tests in the automated test suite.

This was a problem of attitude rather than reality. In order to write the code, the programmer has to have some expectation of what the program will do. If the programmer doesn't know how the code should respond to certain inputs or what should be returned as a result of some coding sequence then they don't understand the requirements. Writing an automated test is the ultimate level of formalisation of the requirements.

Edsger W Dijkstra (2000) said "A programmer has to be able to demonstrate that his program has the required properties. If this comes as an afterthought, it is all but certain that he won't be able to meet this obligation: only if he allows this obligation to influence his design, there is hope that he can meet it. Pure a posteriori verification denies you that wholesome influence and is therefore putting the cart before the horse..."

Being able to develop a testing strategy based on the requirements shows a clear understanding of what is required. It is only at this point that the programmer has a clear idea of what is required and has, as Dijkstra said, allowed the obligation to "to demonstrate that his program has these properties" to influence the design. Simply gathering numerical proof as evidence of correct operation denies this influence.

McBreen (2001) argues that "Software craftsmen have a real interest in automated testing because of their investment in their reputations."

Should the grade assigned to the student reflect an "investment in their reputation" or should it be an indicator of a good attempt even if they failed to prove the integrity of their code? Students who failed to accept this message received a lower grade than they would have using a scoring / weighting rubric (Thompson, 2004)

This complaint wasn't a failure of the SOLO taxonomy marking criteria but rather failure of our teaching strategy.

## 6 Conclusion

This paper outlines criteria that have been used for programming assignments. The cases described show issues that arose and how these issues came from perspectives that were challenged by the criteria.

Did the strategy cause the students to change to a more holistic approach to their assignment work? Clearly there was resistance to change and some students refused to change (cases 5.2 and 5.3). Some of this resistance was caused by lack of familiarity with the approach to assessment reflected in the criteria (case 5.1). Other resistance came from a feeling that they could not select what they wanted to focus on (case 5.2). This reflected a resistance to a holistic approach to the task. Further resistance came from issues of understanding the process of software development and what it meant to have completed an iteration (case 5.3). This wasn't necessarily resistance to a holistic assessment approach to the task. In two of the cases, the resistance came from perspectives of what it means to complete functionality (case 5.4) or to ensuring that code worked as required (case 5.5). Although this might be seen as resistance to a holistic approach to the task, the teacher considered the approach to teaching as a possible influence on the student attitudes. Were the students taught from the beginning to think in terms of code reuse and with an obligation to fully understand the requirements in the form of tests? The lecturer records that test-driven development was taught but wondered whether the students understand how to translate requirements to tests and what was involved in that process? In all of these cases, the students were thinking about what was expected of them and the criteria were influencing the judgement of the students.

The lecturer in recording the instances tended to focus on the difficulties and recorded less of the positives. This was so that the lecturer could endeavour to address these issues in the next offerings of the papers. The positives that were recorded reflected comments of appreciation for the lecturer's teaching style or what had been learnt. These notes showed that the students recognised that this lecturer's papers were difficult but that the students felt that they learnt a significant amount.

## 7 Discussion

Like any marking criteria, there needs to be alignment with the specification of the task. In the SOLO taxonomy marking strategy and the cases, there is a consistent strategy of rewarding valued programming practices and placing of an emphasis on the quality of the solution. The alternative may be to emphasise achieving a solution with quality as an option. A further study on this is to be completed as part of research currently being carried out on improving learning in programming.

The SOLO taxonomy based marking criteria have been used with programming assignments that involved the implementation of a reasonable sized program. Can it be applied to smaller projects where there may not be so much duplication or shared functionality in the code? The research work suggests that we can use it to evaluate student responses to reading and comparing code segments (Whalley et al., 2006, Thompson et al., 2006) and in evaluating the strategies used to solve code reading problems (Lister et al., 2006). Work is continuing on establishing the use of the SOLO taxonomy to assess the writing of code segments for exam conditions.

What is clear is that there is a need to ensure the project includes issues that enable solutions that are dependant on more than a single or limited range of concepts. Larger projects should provide for the possibility of implementing a solution as independent pieces or as an integrated whole. Biggs (1999) describes how the SOLO taxonomy can be used to write assessment items.

## 8 References

- Biggs, J. B. (1999) *Teaching for quality learning at University*, Buckingham, Open University Press.
- Biggs, J. B. & Collis, K. F. (1982) *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*, New York, Academic Press.
- Dijkstra, E. W. (2000) Answers to questions from students of Software Engineering. From <http://www.cs.utexas.edu/users/EWD/ewd13xx/EWD1305.PDF>.
- Felleisen, M., Flatt, M., Findler, R. B., Gray, K. E., Krishnamurthi, S. & Proulx, V. K. (2001) *How to design class hierarchies: Object-oriented programming and computing*.
- Hattie, J. & Purdie, N. (1998) The Solo model: Addressing fundamental measurement issues. In Dart, B. & Boulton-Lewis, G. M. (Eds.) *Teaching and learning in higher education*. Camberwell, Vic, Australian Council of Educational Research.
- Lister, R., Simon, B., Thompson, E., Whalley, J. & Prasad, C. (2006) Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. *Innovation and Technology in Computer Science Education (ITiCSE 2006)*. Bologna, Italy.
- Maki, P. (2004) *Assessing for learning: Building a sustainable commitment across the institution*, Sterling, VZ, Stylis Publishing.
- McBreen, P. (2001) *Software craftsmanship: The new imperative*, Boston, Addison Wesley.
- McConnell, S. (2004) *Professional software development: shorter schedules, higher quality products, more successful projects, enhanced careers*, Boston, Addison Wesley.
- November, P. (1996) Journals for the journey into deep learning: a framework. *Higher education research and development*, 15, 115-127.
- November, P. (1997) Learning to teach experientially: a pilgrim's progress. *Studies in Higher Education*, 22, 289-299.
- Proulx, V. K. & Gray, K. E. (2006) Design of class hierarchies: An introduction to OO program design. *Inroads - The SIGCSE Bulletin*, 38, 288-292.
- Thompson, E. (1997) *71253 Systems Design and Development*, Lower Hutt, The Open Polytechnic of New Zealand.
- Thompson, E. (1998) Delivering education that satisfies industry. *NACCQ National Conference*. Auckland, National Advisory Committee on Computing Qualifications.
- Thompson, E. (2004) Does the sum of the parts equal the whole? In Mann, S. & Clear, T. (Eds.) *Proceedings of the seventeenth annual conference of the National Advisory Committee on Computing Qualifications*. Christchurch, New Zealand, National Advisory Committee on Computing Qualifications.
- Thompson, E., Whalley, J., Lister, R. & Simon, B. (2006) Code Classification as a Learning and Assessment Exercise for Novice Programmers. In Mann, S. & Bridgeman, N. (Eds.) *The 19th Annual Conference of the National Advisory Committee on Computing Qualifications: Preparing for the Future — Capitalising on IT*. Wellington, National Advisory Committee on Computing Qualifications.
- Whalley, J., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, A. & Prasad, C. (2006) An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. In Tolhurst, D. & Mann, S. (Eds.) *Eighth Australasian Computing Education Conference (ACE2006)*. Hobart, Tasmania, Australia, Australian Computer Society Inc.

# Engaging Undergraduates in Discussions about Ethics in Computing

Brian R. von Konsky Jim Ivins Susan J. Gribble

Curtin University of Technology  
Department of Computing  
Software Engineering Education Research Group  
GPO Box U1987, PERTH WA Australia

bvk@cs.curtin.edu.au

## Abstract

Third-year computing students enrolled in a software engineering subject were introduced to the Australian Computer Society (ACS) Code of Ethics in the context of a computing professional's obligation to manage quality, safety and reliability. Following an introductory lecture, case study scenarios were interactively discussed during class. Immediately afterwards, students were surveyed to assess their self-perceived and actual ability to apply the ACS Code to another similar scenario. Of the 68 students who gave their informed consent to participate in the study, 34% reported being fully comfortable with applying the ACS Code of Ethics, while 63% were somewhat comfortable. In justifying multiple-choice options for dealing with a new case study scenario, 37% provided a good justification for their choice, 48% provided a poor justification, and 15% provided no justification. A further qualitative analysis of the responses suggests the need for formal assessment of ethics in computing education, and highlights the importance of improving the perceived relevance of ethics to students and the need for in-depth treatment of ethical issues.

*Keywords:* Ethics, Teaching and Learning, Engagement

## 1 Introduction

Information and Communication Technology (ICT) professionals are faced with ethical challenges as society adopts new and increasingly complex tools and technologies. These ethical challenges have the potential to dramatically impact consumers of software products and services, the wider community, and colleagues in the workplace.

Recognizing the importance of the ethical and professional responsibilities of ICT professionals, the Australian Computer Society (ACS) requires professionally accredited undergraduate ICT programs to include content related to ethics and the social implications of ICT (Hart, 2003).

Despite common agreement that it is important to expose undergraduate students to the ethical issues associated with their profession, there is a lack of consensus regarding whether ethics should be taught as a separate subject or within the context of technical ICT subjects (Dudley-Sponaule and Lidkte, 2002, Martin and Huff, 1997), and whether it is necessary to formally assess learning experiences related to ethics in technical subjects (UWA, HREF).

This paper reflects on some of these issues, reporting on the experience associated with making the transition from not assessing ethical issues, to formally examining them in a third-year software engineering subject.

## 2 Background

Various approaches for teaching ethics to students of Information Technology (IT), Computer Science (CS), and Software Engineering (SE) have been described in the literature. These include the use of written reflective exercises, team debates, and group discussions conducted in the classroom (Towell and Thompson, 2004). Such learning experiences are often structured around case studies drawn from news stories reported in the popular press (Bower, 2006), from the professional literature (Kumagai, 2004), from the personal experience of the instructor, or from fictitious scenarios designed to highlight the ethical issues faced by professionals working in large organizations with many stakeholders (Epstein, HREF). Often, these learning experiences are conducted in the context of a Code of Ethics from a relevant professional society.

In Australia, the ACS is the recognised professional society for ICT professionals. All ACS members agree to adhere to a Code of Ethics (ACS, HREFa), which is contained within the Society's National Regulations (ACS, HREFb). The application of the ACS Code of Ethics to fictitious scenarios was used as the framework for teaching computing ethics as reported in this paper.

Other professional societies such as the Association for Computing Machinery (ACM) and the British Computer Society (BCS) have their own ethics codes. However, these codes all share common attributes of professionalism, honesty, and social responsibility (Burmeister, 2000). Consequently, the results reported in this paper should still be applicable outside of Australia.

The ACS Code of Ethics identifies guiding ethical principals related to competency, honesty and professional development. It further requires ACS

---

Copyright © 2007, Australian Computer Society, Inc. This paper appeared at the Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, Australia, January 2007. Conferences in Research in Practice in Information Technology, Vol. 66. Samuel Mann and Simon Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

members to consider the impact of their work on society and the profession they represent, and to “... *act with professional responsibility and integrity in... dealings with the community and clients, employers, employees and students*” (ACS, HREFa).

In particular, an ethical framework is set out in six categories that embody the Society’s core values and ideals. These are found in the opening sections of the Code, with further elaboration provided in subsequent clauses. These six categories are listed below for ease of reference (ACS, HREFa)

**“... Priorities**

*I must place the interests of the community above those of personal or sectional interests.”*

**“... Competence**

*I must work competently and diligently for my clients and employers.”*

**“... Honesty**

*I must be honest in my representations of skills, knowledge, services and products.”*

**“... Social Implications**

*I must strive to enhance the quality of life of those affected by my work.”*

**“... Professional Development**

*I must enhance my own professional development, and that of my colleagues, employees and students.”*

**“... Information Technology Profession**

*I must enhance the integrity of the information technology profession and the respect of its members for each other.”*

## 2.1 Applying the ACS Code of Ethics

Burmeister (2000) wrote a seminal article in which the ACS Code of Ethics was applied to various case study scenarios. The scenarios involved issues related to intellectual property, privacy, confidentiality, quality in professional work, fairness and discrimination, liability for unreliability, software risks, conflicts of interest, and unauthorised access to data. For each scenario, the article discussed the how the Code of Ethics can be applied and suggested other sources for further reading.

More recently, the ACS Code of Ethics Project has provided many new and interesting scenarios, cross referenced with the relevant categories of the Society’s core values and ideals, and with the relevant clauses of the Code, all presented in a convenient tabular format (ACS, HREFc). Additionally, ethical cases studies drawn from the popular press have appeared in *Information Age*, a publication of the ACS (Bowern, 2006).

## 2.2 Engagement

Recently, it has been suggested that the extent to which students engage with learning experiences should be considered when evaluating the effectiveness and quality of teaching and learning activities (Coates, 2005). In the context of the present study, this suggests that students

who are not engaged during class discussions of case studies framed in the context of an ethical code of practice may not be in a position to constructively comment on their ability to apply that code.

This may be particularly important in learning experiences conducted in large classroom environments. However, this does not need to be the case. For example, student-student and student-instructor interaction have been shown to be effective pedagogical approaches to teaching and learning, where students learn through engagement in a shared learning experience, even in large lecture environments (Mazur, 1997).

## 3 Methodology

The study reported in this paper was conducted in a third-year software engineering subject that is a core requirement for students undertaking the Bachelor of Engineering (Software Engineering), Bachelor of Science (Computer Science) and Bachelor of Science (Information Technology) degrees at an Australian university. The initial lecture introduced the ACS Code of Ethics from the standpoint of managing software quality and safety, and included interactive case study discussions. Subsequent topics of a more technical nature included quality management standards, verification and validation, and requirements analysis using formal methods and graphical modelling languages. All topics were presented in the context of managing project quality and safety.

During the initial lecture, application of the ACS Code of Ethics was interactively discussed using scenarios originally posed by Burmeister (2000). The instructor moderating the case study discussion attempted to create an engaging and collegial environment to encourage students to participate in the discussion, and was careful to keep the dialogue focussed on ethical issues rather than technical aspects of the case studies (Sanders, 2005).

The ability of students to independently apply the ACS Code of Ethics to a new scenario was evaluated using a short survey and feedback form. Students completed the survey and feedback form at the end of the two-hour lecture and interactive group discussion. Using the survey and feedback instrument, the ability of students to apply the Code was compared to self-perceptions regarding their ability to do so.

The Burmeister article was required reading. Students were instructed to use it to guide further reflection and self-study in preparation for a mid-semester test, which formally assessed their ability to apply the Code.

Additionally, students were asked to reconstruct the intended Learning Outcomes for the entire third-year subject during a practical session held late in the semester. The purpose of this exercise was to evaluate the extent to which students considered ethics to be a formal part of the subject, or merely an add-on to the technical content.

This study was conducted with the approval of the Human Research Ethics Committee at the authors’ institution.



## 4 Results

Of the 95 students enrolled in the subject, a total of 68 students or 71% of the class gave their informed consent to participate in the study. A further breakdown by degree and final mark is given in Table 1.

Degree	Consent	No Consent	Total
CS	26	8	34
IT	19	11	30
SE	10	4	14
CS Double	6	2	8
Other	7	2	9
Total	68	27	95

Final Mark	Consent	No Consent	Total
9	0	1	1
8	6	1	7
7	15	7	22
6	21	7	28
5	14	4	18
F	12	7	19
Total	68	27	95

**Table 1. Breakdown of the cohort by degree program (top) and final mark (bottom) for consenting and non-consenting students.**

A Chi-squared test of association showed that the consenting and non-consenting groups were statistically similar with respect to degree program ( $\chi^2=1.62$ , d.f.=4,  $p>0.05$ ) and final mark ( $\chi^2=4.49$ , d.f.=5,  $p>0.05$ ). The high percentage of students agreeing to participate in the study, together with the statistical similarity between the consenting and non-consenting groups, suggested that the consenting group was representative of the class as a whole. All further data presented in this paper were obtained only from students giving their informed consent to participate.

### 4.1 Ethical Scenario 1 (Formative Task)

Immediately following an initial lecture on the ACS Code of Ethics and the subsequent interactive discussion of relevant case studies (Burmeister, 2000), students were asked to complete a feedback form which requested identifying information and included the following text:

*This is not a test. There is no “right” answer. Completing this part of the form will not impact your mark in this unit.*

*I feel that I can apply the ACS code of ethics as a result of participating in this lecture.*

(Circle One):    Yes    No    Somewhat

Most students believed that they were able to do this with at least some degree of confidence, with 34% responding “yes”, 63% responding “somewhat”, 1% responding “no”, and 1% providing no answer.

The feedback form also presented students with the following hypothetical scenario and choices for potential courses of action.

*You studied Java during your undergraduate course. You are now a graduate, who was recently hired by a company that traditionally used Java. However, the company has decided to switch to a different OO programming language on a new project to which you’ve been assigned. You’ve never used the new language. The project has a very aggressive schedule and a tight budget. You want to make a good impression on your new boss and colleagues. In light of the ACS Code of Ethics do you*

*(circle one)*

- Ask to be shifted to another project that uses Java*
- Read up on the language but don’t tell anyone about your lack of experience*
- Discuss it with the boss and ask for additional training*
- Send an anonymous message to the client warning them of the lack of experience on the part of some members of the company*
- Do the best you can and hope no one notices that you don’t know the new language*
- Other (specify) \_\_\_\_\_*

*Write one or two sentences that explain your answer.*

It was expected that students would justify the selected response by making an ethical argument positively impacting one or more stakeholders in one or more of the six categories from the ASC Code of Ethics. Less than five minutes were allocated to complete the form, so students were only asked to provide brief explanations.

A reasonable justification for selecting choice a) would be to avoid misrepresenting skills and technical abilities in a manner that would potentially have a negative impact on project outcomes and on colleagues, employers, shareholders, and clients.

Alternatively, a reasonable justification for selecting choice c) would be the ethical responsibility of an ICT professional to remain current in ones discipline through professional development, with the effect of being able to serve the expectations of ones employer by learning the new language, leading to other positive outcomes for the client.

As shown in Table 2, almost 80% of the students selected “c) Discuss it with the boss and ask for additional training” in response to the given scenario, while 6% selected “a) ask to be shifted to a different project that uses Java”. 12% proposed an alternative answer, but many of these were a combination of choices a) and c).

Choice	N	%
a)	4	6
b)	1	1.5
c)	53	78
d)	0	0
e)	1	1.5
f)	8	12
No response	1	1.5

**Table 2. Multiple-choice responses to Formative Task**

As explained below, with respect to the one or two sentences that students were asked to provide to explain their answer, 37% provided a good justification, 48% provided a poor justification, and 15% provided no justification.

The justifications were deemed to be good if they considered ethical issues associated with one or more stakeholders and/or one or more of the six ACS Code of Ethics categories. For example, one student who selected choice c) wrote: *“You have an ethical obligation to notify your employer of your shortcomings, or lack of experience.”* In this answer, the student acknowledged the role of the employer as a significant stakeholder and was attempting to be honest about skills and abilities.

Another student wrote: *“It will allow the boss to realise that I am honest, willing to learn and hardworking. Of course also, willing to constantly upgrade myself.”* This student not only acknowledged an employee’s responsibility to the boss, but also one’s personal responsibility to undertake further training and professional development. A third student provided similar justification, but acknowledged a wider range of stakeholders, writing: *“I would have to act responsibly and let my boss know of my situation. It would be in the best interests of myself, the company and indirectly the end-consumer.”*

Another student considered choice a) but ultimately picked c), writing: *“You could ask to be moved to a project which uses Java, but then you will never be able to gain the additional skills and knowledge.”* This brief statement suggests that the student considered the acceptable alternatives in light of the Code, but ultimately selected one alternative over another to have the best positive career outcome. Another student apparently agreed with this assessment, writing *“I would discuss it with my boss instead of asking to be shifted. This is because while a) is also ethical, I still have to act professionally.”* Both students appear to be referring to an ICT professional’s responsibility to undertake professional development as outlined in the Code.

Poor explanations generally did not consider ethical issues impacting stakeholders, and did not consider any of the categories from the ACS Code of Ethics. Instead, these explanations tended to be based on technical justifications. For example, one student who selected choice c) justified this by writing *“Moving from one OO language to another is usually not significant as long as you understand the fundamental concepts”*. Another wrote: *“The design of the software should be such that his inexperience won’t impact the final product.”*

While most students believed that they could apply the ACS Code of Ethics to some extent following the lecture and subsequent group discussion, over half of the students provided a poor justification, or no justification for their choice of response.

## 4.2 Ethical Scenario 2 (Assessment Task)

To determine if further private study would improve the ability of students to apply the ACS Code of Ethics, copies of the paper by Burmeister (2000) were distributed to students as required reading. In particular, students were advised to use the paper in preparation for the Mid-Semester Test, which posed the following scenario:

*“You are the Chief Technical Officer for a company that has been awarded the contract for a software-based system to automate pumping stations in a new waste treatment network to be implemented in Western Australia. At the onset, you were confident that your firm had the necessary expertise to undertake the project. Mid-way through the project, the Board of Directors agreed to a change in requirements involving a new technology to facilitate telecommunication between the pumping stations. Your firm has limited experience with telecommunications systems, a fact that was not disclosed when the client introduced the change in requirements. You are concerned that the change will have a significant impact on the quality of the delivered system that could potentially result in the spillage of untreated waste into the Swan River. The budget is tight and there are severe contractual penalties for delivering the system late. Stage 1 of the system has already been implemented, but has not yet been tested in the field. The company directors and relevant government ministers are unaware of your concerns. You have a responsibility to serve the interests of all stakeholders, including your shareholders, colleagues, and the citizens of Western Australia.”*

*“Apply the ACS Code of Ethics, describing potential actions that could be taken in light of the Code and considering the interests of the various stakeholders.”*

In answering this test question, students were expected to write a short essay describing their preferred outcome, providing greater detail than in the previous exercise. As before, it was expected that solutions to the test question would discuss the impact on stakeholders in light of relevant categories from the Code, in a manner similar to that demonstrated by Burmeister (2000). However, students were not required to memorize the ACS Code of Ethics, or expected to cite specific clauses in formulating their response to the question.

In addition to considering many of the same types of issues discussed in the previous scenario, it was expected that the social implications of a sewage spill and the reputation of the ICT industry would be considered in formulating an appropriate answer.

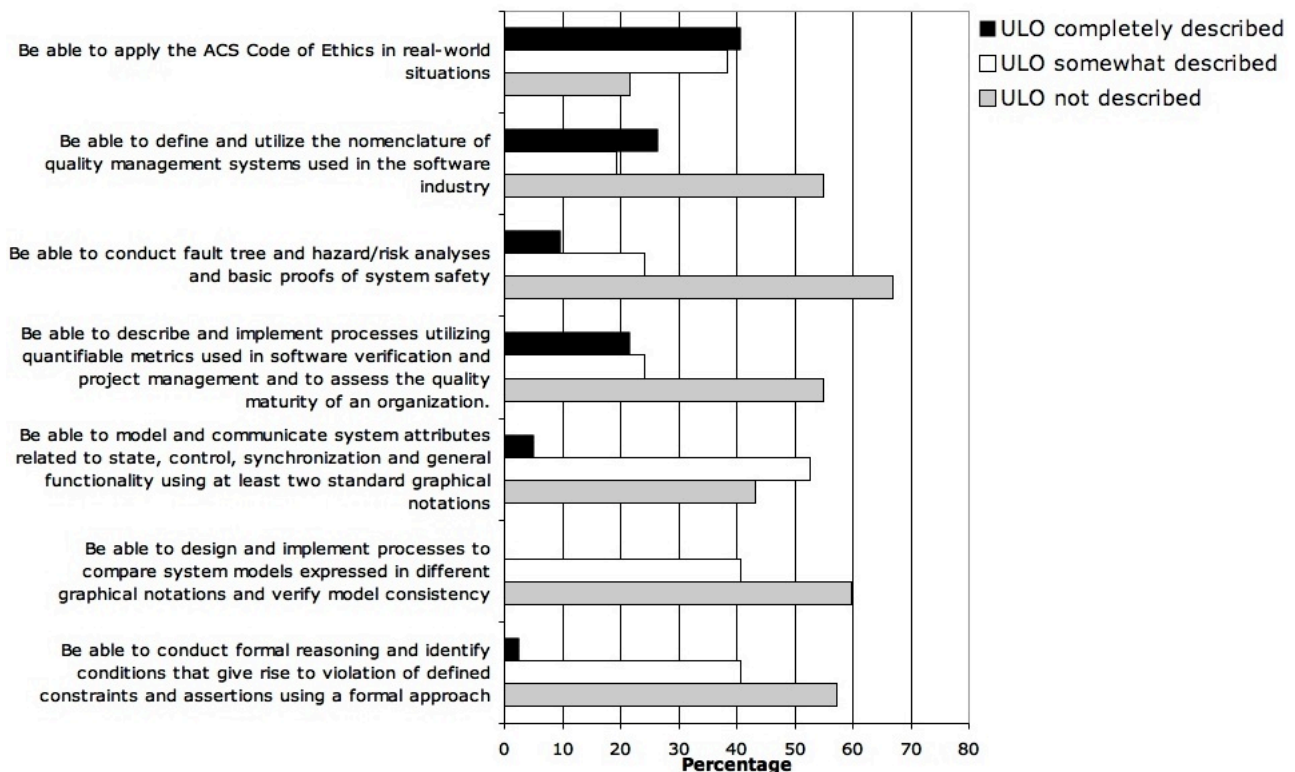


Figure 1. The extent to which students were able to list and describe each Unit Learning Outcome (ULO) in their own words.

In response to the test question, one student wrote the following:

*"The ACS Code of Ethics says we should not put our own advancement ahead of the S.E. community or the community at large. If no action is taken the company is putting their own immediate financial gain ahead of the societal consequences, i.e. pollution."*

*"The code says we shall not behave in a manner which will bring the SE community/profession into disrepute. Now, if the system were to fail due [to] the engineers' lack of core knowledge then the industry would indeed be brought into disrepute."*

*"The engineers have an ethical obligation to disclose all relevant information to their clients. If the inexperience and potential danger is not disclosed, this is a direct contradiction of the code."*

*"My suggested action is to inform the client of the potential risk of failure... Inform them that due to the requirements change a contract variation is needed in order to deliver a system which best and fully meets the needs of the client and the community...."*

Another student suggested three potential actions. However, in the subsequent discussion these were largely analysed from a business perspective based on the financial implications, rather than from an ethical standpoint. The student wrote:

*"Potential action 1: Inform [the] Board of Directors of your lack of experience in this area, outlining the risks and effects on budget and schedule, and let them decide whether to continue."*

*"Potential action 2: Pull out of the project, citing lack of experience in this field."*

*"Potential action 3: Hire subcontractors with experience in this field, and cut expenses and time from other aspects of the project."*

While hiring sub-contractors was an interesting suggestion, the subsequent analysis did not discuss this or the other alternatives in any detail. Neither did the analysis discuss the alternatives in the context of the ACS Code of Ethics. Instead, the essay analysed potential actions in terms of the financial ramifications. Additionally, the student failed to take a stand or recommend a specific action from the three nominated possibilities.

Marks were awarded for identifying the appropriate stakeholders and discussing ethical outcomes affecting them in light of the ACS Code of Ethics. The average mark for this test question was 5 out of 9 marks, with a standard deviation of 3. While some students did an excellent job of analysing the test scenario in the context of the ACS Code of Ethics, many students took a technical or financial approach to the analysis, making minimal reference to the Code or its various categories. This was despite the view expressed in the earlier feedback instrument, that students were either fully or

somewhat confident with conducting a case study analysis in the context of the Code.

During the 11<sup>th</sup> week of instruction in a 12 week teaching semester, students were asked to list each Unit Learning Outcome (ULO) for the subject in their own words, based on the various learning experiences in which they participated, and without referring to the official list of ULOs provided to students at the beginning of the semester. Responses were analysed to determine if each of the seven outcomes was completely, partially, or not described by each student. Results are shown in Figure 1, with those ULOs generally addressed early in the semester shown towards the top, and ULOs generally addressed later in the semester shown towards the bottom.

41% of the students were able to describe the Unit Learning Outcome: "Be able to apply the ACS Code of Ethics in real-world situations". A further 38% were able to describe it somewhat, writing a ULO statement that said something about ICT ethics, but not necessarily about the ACS Code of Ethics or their ability to apply it. 21% of the students did not describe this ULO at all. This suggests that most students recognised knowledge of ICT ethics to be an outcome of their participation in the subject, alongside the technical content.

Students generally did a better job of articulating the Learning Outcomes associated with ethics and other material presented early in the semester. In contrast, they had greater difficulty describing the more technical Learning Outcomes that were presented later in the semester. While this may be due to the fact that students had already prepared and undertaken formal assessment (i.e. the mid-semester test) for the earlier material, it is generally consistent with the earlier view regarding the level of comfort students associated with applying the Code to various scenarios.

## 5 Discussion

During previous semesters, the Burmeister (2000) case studies have been discussed during the first lecture of the subject described in this paper. Prior to 2006, however, this material was not formally assessed (von Konsky *et al.*, 2006). Indeed, others have suggested that assessing attainment of ethics learning outcomes is not always necessary (UWA, HREF):

*"For example, a discussion about ethical issues may be purposely raised in a tutorial or laboratory session, but there may be no attempt to formally assess whether or not students have achieved the outcome 'appraisal of ethical issues related to the subject matter'. The discussion itself may serve the important purpose of raising pertinent questions for ongoing reflection and consideration."*

However, this study has shown that student opinions regarding the attainment of ethics learning outcomes do not always reflect a student's ability to apply a Code of Ethics in practice. By itself, the experience reported here suggests that these subjective opinions should not be

taken to demonstrate outcome attainment for the purposes of course review or accreditation.

Further, it appears that many students will make so-called "ethical arguments" solely on the basis of technological and financial considerations. If a student has substantial prior experience with technology or commerce, as is often the case, the experience reported here suggests that this will take priority over a guiding Code of Ethics. In other words, while the technical aspects of a scenario are important, these must be guided by an ethical code of practice to assist in making appropriate decisions. Otherwise, the technical predisposition of ICT students may adversely bias the tradeoffs that are sometimes associated with difficult ethical situations.

These findings further suggest the need to expand related learning experiences beyond those described here, in order to find additional ways to make this material more relevant to students. Indeed, in a question regarding how learning could be improved in the subject, a high distinction student wrote the following on the end of semester questionnaire: "*Less ethics or more relevant use of them. I realize it is probably a requirement, however in the current form it seems highly impractical and of little use to me.*" While it is unclear if other students shared this view, the comment is well founded.

## 6 Conclusion

While ICT ethics is sometimes taught in a subject dedicated to ethics and the social impact of ICT, it is also frequently taught *in situ* in the context of other subject material, as is the case here. Indeed, attempts have been made to construct a pedagogical framework that supports teaching ethics throughout the computer science curriculum (Martin and Huff, 1997).

Whether it is better for ICT ethics to be dealt with in a dedicated subject or spread across the curriculum is unclear. What is clear, however, is that teaching ethics in a manner that is conducive to formal assessment, and that is seen to be practical and relevant by students, is vital to ensure that the next generation of ICT professionals is well prepared to face the ethical challenges of a technologically and socially dynamic world.

## 7 References

- ACS (HREFa): ACS Code of Ethics,  
last accessed 16 October 2006,  
<http://www.acs.org.au/index.cfm?action=show&conID=coe>
- ACS (HREFb): ACS National Regulations,  
last accessed 16 October 2006  
[https://www.acs.org.au/about\\_acs/docs/NATREGSvFeb2005.pdf](https://www.acs.org.au/about_acs/docs/NATREGSvFeb2005.pdf)
- ACS (HREFc): Case Studies on Ethics,  
last accessed 16 October 2006,  
[http://www.acs.org.au/publication/docs/ACS\\_CaseStudiesFinal.pdf](http://www.acs.org.au/publication/docs/ACS_CaseStudiesFinal.pdf)
- Bowern, M. (2006): Ethics: part of being a professional, *Information Age*, June/July 2006, pp 51-52.

- Burmeister, O. (2000): Applying the ACS Code of Ethics, *Journal of Research and Practice in Information Technology*, **32**(2): 107-120.
- Coates, M. (2005): The value of student engagement for higher education quality assurance, *Quality in Higher Education*, **11**(1): 25-36.
- Dudley-Sponaule, A., and Lidtke, D. (2002): Preparing to teach Ethics in a computer science curriculum, *International Symposium on Technology and Society, 2002, ISTAS'02*, pp. 121-125.
- Epstein, M. (HREF): The Case of the Killer Robot, last accessed 16 October 2006, <http://onlineethics.org/cases/robot/robot.html>
- Hart, B. (2003): Accreditation of Courses at the Professional Level, Guidelines for Applicants, The Australian Computer Society, last accessed 16 October 2006, <http://www.acs.org.au/accreditation/accreditationmanual.doc>
- Kumagai, J. (2004): The whistle-blower's dilemma, *IEEE Spectrum*, **41**(4):53-55.
- Martin, C.D., and Huff, C.W. (1997): A conceptual and pedagogical framework for teaching ethics and social impact in computer science, *27<sup>th</sup> Annual Frontiers in Education Conference*, **1**:479-483.
- Mazur, E. (1997): *Peer Instruction: a user's manual*, Sydney, Prentice Hall Series in Educational Innovation, ISBN 0-13-565441-6.
- Moskal, B., Miller, K., and Smith King, L.A. (2002): Grading essays in computer ethics: rubrics considered helpful, ACM SIGCSE Bulletin, *Proceedings of the 33rd SIGCSE technical symposium on Computer Science Education*, SIGCSE '02, **34**(1):101-105.
- Sanders, A.F. (2005): A discussion format for computer ethics, ACM SIGCSE Bulletin, *Proceedings of the 36th SIGCSE technical symposium on Computer Science Education*, SIGCSE '05, **37**(1): 352-355.
- Towell, E., Thompson, J. B. (2004): A further exploration of teaching ethics in the software engineering curriculum, *Proceedings of the 17<sup>th</sup> Conference on Software Engineering Education and Training*, CSEE&T'04, pp 39-44.
- UWA (HREF): Outcomes Based Education and Assessment at UWA, last accessed 16 October 2006, [http://www.catl.uwa.edu.au/\\_data/page/77897/OBE\\_and\\_Assessment.pdf](http://www.catl.uwa.edu.au/_data/page/77897/OBE_and_Assessment.pdf)
- von Konsky, B.R., Loh, A., Robey, M., Gribble, S.J., Ivins, J., and Cooper, D. (2006): The Benefit of Information Technology in Managing Outcomes Focused Curriculum Development Across Related Degree Programs, 8th Australasian Computing Education Conference, ACE 2006, *Conferences in Research and Practice in Information Technology*, D. Tolhurst and S. Mann, Eds., **52**:235-242.



# Decoding Doodles: Novice Programmers and Their Annotations

**Jacqueline Whalley**

School of Computer and Information  
Sciences  
Auckland University of Technology  
Private Bag 92006, Auckland 1020,  
New Zealand  
+64 (9) 921 9999  
jacqueline.whalley@aut.ac.nz

**Christine Prasad**

School of Computing and Information  
Technology  
Unitec New Zealand  
Private Bag 92025, Auckland, New  
Zealand  
+64 (9) 815 4321  
cprasad@unitec.ac.nz

**P. K. Ajith Kumar**

Bay of Plenty Polytechnic  
Tauranga,  
New Zealand  
+ 64 (7) 5440920  
Ajith.Kumar@boppoly.ac.nz

## Abstract

This paper reports on the annotations made by novice programming students on an exam script. We investigate the level of reasoning that the students achieve when answering a short answer question using the SOLO taxonomy and relate this to the type and number of annotations they made. The questions and annotations were classified and the relationship between question type, student performance and the tendency to annotate was explored.

**Keywords:** Computing education, novice programming, annotation, doodle, cognitive taxonomies.

## 1 Introduction

It is no secret that, despite the best efforts of teachers in our discipline, students continue to face many challenges when learning computer programming. Recently, multi institutional multi national (MIMN) studies, as described by (Fincher *et al.* 2005), have provided the CSEd community with a new means of gaining insight into the issues faced by novice programmers. MIMN studies such as “the McCracken Group” (McCracken *et al.* 2001), “the Leeds Group” (Lister *et al.* 2004), Bootstrapping (Petre *et al.* 2003), Scaffolding (Fincher *et al.* 2004) and BRACE (Fincher *et al.* 2005) have highlighted the difficulties that novice programmers face when learning computer programming. Within the CSEd community MIMN studies have facilitated hypothesis generation and data collection. They have also strengthened research methodology through a sharing of ideas and knowledge.

In this paper, we report on an analysis of data gathered by an MIMN study, the BRACElet project. This project was designed to investigate the reading comprehension skills

of novice programmers. The data analysis presented in this paper focuses on the annotations or “doodles” that accompanied student responses to multiple-choice questions (MCQ’s). In particular, we aim to answer the following questions:

- Is there a relationship between students’ reasoning skills according to the different levels of the SOLO taxonomy and their use of doodles?
- Is there a relationship between student achievement and their frequency of doodling?
- Is there a relationship between the level of difficulty of a question and the likelihood of doodles being used to solve the problem?
- Are students more likely to doodle on certain types of questions?

In the discussion that follows we put the study into context, provide some insights into the use of doodles by novice programmers and in doing so identify future avenues for investigation.

## 2 Background

The Leeds Group defined a “doodle” as diagrams and annotations that experienced programmers write or draw when faced with the task of determining the function of the code. With respect to student responses to MCQs, they referred to a “doodle” or annotation as any marking a student makes on his or her exam paper. They categorised the doodles into 12 classes. This group found that if students traced through the code, they were more likely to get the correct answer.

McCartney *et al.* (2004) further analysed the Leeds Group’s data using the same categorization of doodles as the original study. They looked at the interrelationships between the kinds of annotations used by students, the difficulty and type of individual questions, and the institutions where the students were tested. They found that performance improved when students annotated their tests. Overall, they concluded that any annotation was better than none.

Fitzgerald, Simon and Thomas (2005) identified 19 strategies that novice programmers use to solve MCQ’s.

One of the strategies was doodling. They made a clear distinction between tracing code and other doodles. They claimed that “tracing is an entire process that may or may not involve doodling”. All students in their study employed a range of strategies, and the choice of strategy adopted was influenced by the problem type. Moreover, students often employed strategies poorly, which they considered an indicator of fragile knowledge.

For the purposes of our study we developed a new set of problems and designed the test script so as to encourage students to annotate them. The problem set used contained MCQ’s and short answer questions. These questions were classified within a cognitive framework that was based on the revised Blooms taxonomy (Anderson *et al.* 2001) and an adaptation of the SOLO taxonomy (Biggs 1999). The responses to the two short answer questions were analysed using SOLO to determine the reasoning ability of each of the students in the study. Additionally, the student responses to the MCQs were compared with expert responses from teachers of programming. Information on the project and its question set, administration and analysis has been documented in Whalley *et al.* (2006), Thompson *et al.* (2006), and Lister *et al.* (2006).

### 3 Data Collection

The data collected consisted of student answers to nine MCQs and two short answer questions. Analyses of the MCQs (Whalley *et al.* 2006), the short answer question 10 (Whalley *et al.* 2006, Lister *et al.* 2006) and the classification question 11 (Thompson *et al.* 2006) have already been reported. In this paper we are investigating the use of annotations on the student scripts.

This analysis looks at the scripts of 71 students from a single institution that participated in this study. These students had either nearly completed or just completed the first semester of their first programming course. In all cases the problem set counted towards the student’s final grade and were taken under examination conditions in a single session.

The students were encouraged to annotate their scripts and plenty of space was provided on the problem sheet with each problem appearing on a separate sheet to allow for doodling.

### 4 Data Coding

The doodles on the students’ scripts were analysed by the authors using a set of defined categories. The coding schema (Table 1) used for the classification of the student script annotations was a modified version of the schema described by McCartney *et al.* (2004).

As a result of the inclusion of new types of questions (for a full discussion of question types see section 5.2) a new doodle type emerged that is a ‘range doodle’ (R). This type of doodle was observed only for question 4 where students were required to determine whether or not a number was within a given range (see Appendix). This doodle helps identify numbers that are valid within a given range and seem to occur with conditional

statements that include a Boolean operator. An example from a student script is shown in Figure 1.

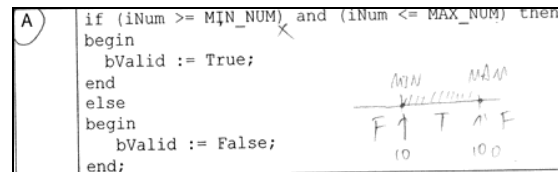


Figure 1: A category ‘R’ doodle

Some of the categories used by McCartney *et al.* (2004) were not of interest for this analysis for example the category E (Extraneous Marks, marks that appear ambiguous or meaningless) was not used. Extraneous marks in our coding were treated as blanks.

Rank	Name	Code	Description
High	Synchronised trace	S	Shows values of multiple variables changing, generally in a table
	Trace	T	Shows the values of a variable as it changes (more than 1 value for at least 1 variable)
	Odd Trace	O	Appears to be a trace but neither S nor T
Med	Keeping Tally	K	Some value being counted multiple times (specific variable not indicated)
	Number	N	Shows a single variable value, most often in comparison
	Position	P	Picture of correspondence between array indices and values
	Range	R	A doodle that depicts a range of valid and invalid values
	Computation	C	An arithmetic or Boolean computation.
Low	Alternate answer	A	Student changed their answer
	Ruled Out	X	One or more alternative answers have been crossed out. The answer appears to have been arrived at by elimination
	Underlined	U	Part of a question underlined for emphasis
	Blank	B	Either no annotations for this question or extraneous markings such as dots, lines and so forth.

Table 1: Annotation Categorisation



For the purposes of our analysis a further layer of classification, namely rank, has been added. The ranking allows for the clear separation of doodles based on our evaluation of the doodles importance in relation to program comprehension. This ranking provides a distinction between tracing and other doodles.

The ranking separates highly meaningful doodles and moderately meaningful doodles from low-relevance doodles. In this context low-relevance doodles are the result of an action taken after the thought process has been completed. They record the result rather than the process (for example U, X, A). On the other hand, meaningful doodles illustrate the process of thinking and are written as the thinking takes place and assist the thought processes that construct an answer.

McCartney *et al.* (2004) provided a similar layer of abstraction for their coding: (Blank, Some Tracing (S, T and O), Elimination (A or X), Other (everything else)). Their layers of abstraction were constructed to create disjoint classes of doodles. Some of the categories included in their “Other” layer have a moderate degree of usefulness as a program comprehension doodle. Others do not, for example U and E and this is why we developed a new grouping of doodles.

## 5 Results

If every student had doodled once on every question there would have been 639 annotations. When we counted any combination of doodle types on a single question as one annotation, the total number of observed annotations was 289 giving a frequency over all the questions of 44.5%.

McCartney *et al.* (2004) found that the percentage of questions with annotations varied from 28% to 92% over the 12 participating institutions in their study. The percentage of doodles obtained in our study was lower than that obtained in 10 of the 12 institutions in the McCartney study. This was despite the fact that the students in our study had been encouraged to annotate their scripts and had been taught synchronised tracing as a formal process. However, we should not be surprised by student reluctance to adopt tracing and code annotation as a technique in light of Thomas, Ratcliffe and Thomasson’s experiences. In 2004 they reported on an occasion where, in an exam, students were encouraged to doodle and yet almost two thirds of the students turned in a paper with no annotations.

On most occasions where our students did doodle they tended to use a single type of doodle (350 times, 60%) and using more than two types in combination was rare (12%). Figure 2 shows the frequency with which combinations of doodle types were used.

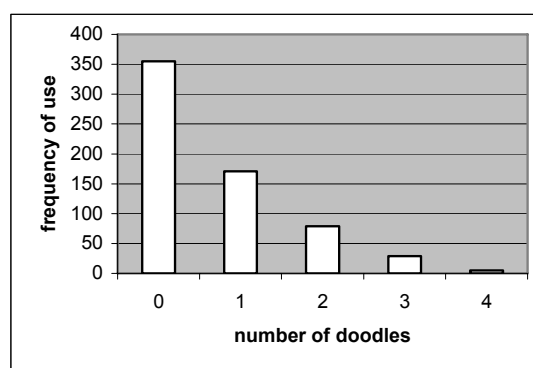


Figure 2: Doodle combinations

The most commonly used doodle combination was N and T. The combination of NT was used in isolation 60% of the time otherwise it was observed in combination with other type(s)

### 5.1 Performance and doodles

The Leeds group concluded that performance in both FC (fixed code) and SC (skeleton code) questions improves when students annotate their tests. We compared the overall achievement of our students on the 9 MCQs with the extent to which they used doodles in order to further investigate the relationship between doodling and achievement (Figure 3).

The extent to which students in the top quartile of achievement used doodles was compared with the use of doodles by students in the second and third quartiles<sup>1</sup> and students in the bottom quartile. Only ten students failed to doodle on any of the questions.

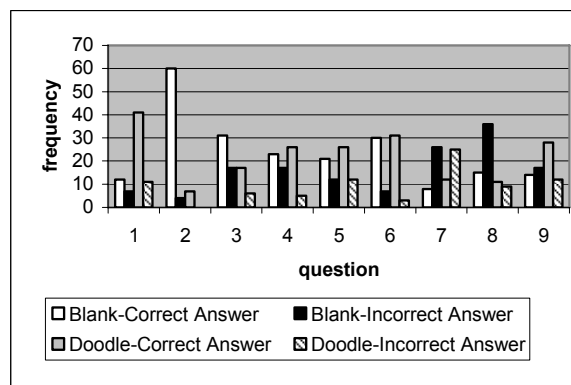


Figure 3: Comparison of the success of students who doodled and those who didn't

Six of the students who failed to annotate at all were in the bottom quartile and 4 were in the middle two quartiles. All students in the top quartile had made at least one annotation.

<sup>1</sup> Data for students in the middle two quartiles were combined because a large number of scores fell at the median score of 6 and the two quartiles could not be meaningfully separated.

A one-way ANOVA of the three groups ( $F = 4.48$ ,  $p < 0.02$ ) showed that higher achievers were more likely to doodle. This provides support for the conclusions reached by McCartney *et al.* (2006).

Students in the lower quartile averaged 2.35 doodles, the middle quartiles 4.26 and top quartile 4.88. However, the only statistically significant difference (Tukey HSD test) was between the top quartile and the bottom quartile ( $p < 0.01$ ). The difference between the bottom quartile and the middle two quartiles very nearly reached the 0.05 level of significance, but the difference between the middle quartiles and the top quartile was well short of significance. A significant but weak correlation  $r = 0.282$  ( $t = 2.44$ ,  $p < 0.02$ ) was found between the number of doodles and student scores.

## 5.2 Replicated Leeds Questions

Two of the MCQs used in this study (problems 1 and 9; see Appendix) were taken directly from the Leeds working group instrument (5 and 2 respectively; Lister *et al.* 2004). These two questions were included to enable some direct comparative analysis.

As reported previously (Whalley *et al.* 2006) the students found question 1 to be the third easiest question in the set while question 9 was the fifth easiest question. Student performance on these two questions was comparable with that observed by the Leeds group.

Seventy-three percent of our students doodled on question 1. This was by far the highest usage of doodles when compared with the other questions. Similarly in the Leeds study they found that 88% of their students doodled on this question and that it elicited the highest number of doodles.

Twenty-eight of our students used some form of trace and 6 used the formally taught synchronised trace (S). Most students who answered this question used a combination of doodle types. Of those students who used tracing only 3 used tracing in isolation. Question 1 was the only question where tracing was the predominant form of doodle type. This use of tracing as an annotation demonstrated that students could effectively trace code but they chose not to employ it as a strategy to the same extent for any other question.

Fifty-six percent of our students doodled on question 9 but 79% of students in the Leeds study doodled on this question (Table 2). Thirty-six percent (26 students) of our students used some form of high-ranking trace doodle. We have no definitive explanation for the difference observed on this question. Perhaps our students doodled less because they met this question in the second half of the problem sheet.

## 5.3 Question types and doodle style

The BRACElet problem set contained four types of questions, two of which had been previously defined as fixed code and skeleton code (McCartney *et al.* 2004). We introduced two new question types that we defined as change of representation and change of logic.

Fixed code (FC) problems are defined as questions where a student is given a code fragment and asked to predict the result after executing that code.

Skeleton code (SC) problems are defined as questions where the student is given a code fragment and asked to complete the code so that it will perform a specified task.

Change of representation (CR) problems are defined as questions where the student is given a code fragment and is asked to identify the same program in an alternative representation or visa versa. For example, question 2 in our problem set provides a code fragment and 4 potential solutions in the form of flow diagrams, one of which represents the code's logic as a flow diagram. In question 3 this is reversed, the stem is a structure diagram and the 4 possible answers are code fragments.

Change of logic (CL) problems are defined as questions where the student is given a code fragment and the solution is a code fragment that should give the same result but the logic of the algorithm has been altered (or reversed). Question 4 is a CL problem.

At first glance (Table 2) our students do not appear to show such a marked difference in the frequency of doodling between FC and SC type questions as was observed by the Leeds group. For FC questions we found a range of 47-73% annotations while the Leeds group saw a range of 66-88% (mean = 57%, (77%)). For SC questions we found 31-50% of students used annotations while in the Leeds study they saw a range of 38-55% (mean = 41%, (41%)). So while our results still agree with the Leeds study in that FC questions resulted in more doodles than SC type questions we did not see such a marked divide.

CL questions appear to encourage doodling at about the same level as SC type questions although, because of the nature of the CL question different doodle types were observed. In particular a new doodle was observed, the range doodle (Figure 1).

	Question								
	1	2	3	4	5	6	7	8	9
count	52	7	24	31	37	34	36	22	40
%	73 (88)	10	33	43	51	47	50	31	56 (79)
type	FC	CR	CR	CL	FC	FC	SC	SC	FC

**Table 2: Number and percentage of annotated questions. The number in parentheses is the percentage reported by McCartney *et al.* (2004) for the same question.**

CR type questions did not encourage doodling (10-33% annotated, %mean = 22) and indeed did not encourage tracing. Of the students who doodled on questions 2 and 3 the main doodle types observed were X, U and A that are all categorised as low ranking doodles. These doodle types are recognised strategies for arriving at an answer to MCQs in any discipline.

The students doodled 3 times more frequently on question 3 than on question 2 although the questions were similar. The most likely reason is because question 3 was the one they found to be more difficult question (question 3 was the 5<sup>th</sup> hardest while question 2 was the easiest).

Type	Hi S, T & O	Med K, N, P, R & C	Low A, U & X	Blank B
FC	29 (81)	25 (72)	4 (11)	42 (120)
SC	13 (18)	15 (22)	12 (17)	60 (85)
CR	3 (4)	1 (1)	18 (25)	79 (112)
CL	0 (0)	28 (20)	15 (11)	56 (40)

**Table 3: Percentages of annotations for each question type. Numbers in parentheses present the counts.**

Table 3 shows that certain types of questions are more likely to elicit certain doodle types. FC questions elicited more high level and medium level doodles than any other type of question. While FC, CL and SC questions encouraged students to use doodles that have some level of programming strategy, CR questions resulted in MCQ low ranking doodles that were a result of the MCQ question format. The low ranking doodles do not directly contribute to the students understanding of the code itself.

#### 5.4 Reasoning vs. tendency to doodle

The tenth question in the problem set was a short answer question. This question was analysed (for a full description see Whalley *et al.* (2006) and Lister *et al.* (2006)) using a set of categories (Table 4) based on the SOLO taxonomy (Biggs and Collis 1982). This taxonomy allowed us to arrive at a level of reasoning for each student. Using this reasoning classification for each student that attempted question 10 we were able to investigate whether or not a student's ability to reason has some correlation with their tendency to doodle.

Our SOLO classification of students answers to question 10 found that there were 22 relational (R), 28 multistructural (M) and 13 unistructural (U). Where no answer was provided and where a prestructural response was given the students were omitted from this section of the analysis. There was no difference found between the level of reasoning of the students, namely R, M and U, and the probability that they would use doodles to help obtain an answer ( $F = 0.03239$ ,  $p = 0.725$ ) in the 9 MCQs. This was unexpected. It had been assumed that multistructural and unistructural students would have a stronger tendency to doodle than unistructural and relational students. Multistructural and unistructural students in general are unable to see the overall purpose of the code and analyse a code fragment at best line by line. We had expected that they would have needed to use high and medium level doodles to arrive at an answer.

SOLO category	Description
Relational [R]	Classification based on a precise summary of what the code does as a whole
Multistructural [M]	Classification based on two or more language constructs of the code
Unistructural [U]	Classification based on one language construct
Prestructural [P]	Classification is related to a code characteristic such as position on page or number of lines of code but not to an understanding of the code.
Blank [B]	No classification performed, the student did not attempt the question.

**Table 4: SOLO categories for question 10**

Are relational students more likely to doodle on more difficult questions? Because relational students are considered to think more like an expert (Lister *et al.* 2006) we expected that they would only need to doodle on the more difficult questions where the overall purpose of the code fragment was not clear. The percentage of the relational students who doodled was compared with the other students on each question (Table 5).

Question	1	2	3	4	5	6	7	8	9
Relational	80	15	30	40	60	50	60	25	55
Others	71	9	36	51	53	49	51	29	58

**Table 5: Comparison of tendency to doodle by relational and non-relational students**

On question 7, the most difficult question, the relational students did indeed doodle more. However, on question 8 the second most difficult question, the non-relational students doodled more. This supports the first finding that there was no difference between the level of reasoning that a student used in a short answer question and the probability that they would use doodles to help solve an MCQ. Furthermore we can conclude that in this survey there was no relation between the difficulty of the question, the relational thinking level as classified on question 10 and the use of doodles.

Davis (1993) found that experts spend more time annotating than novices. If, as we argue, relational students have a higher level of expertise than the other students then we would expect them to doodle more and especially to use tracing more as a strategy for problem solving. However our findings show that the higher-level thinking students did not adopt this technique any more than other students.

Because we had only one question to arrive at a SOLO classification for each student we cannot draw a strong conclusion from these findings. Further investigation is required where questions are designed to elicit the SOLO classification for each student.

## 6 Conclusion

Our study extended the Leeds study by adding a short answer question (question 10) to the research instrument. This question allowed us to use a recognised cognitive framework to determine the level of reasoning of each student. We found that while most of the students worked at the multistructural level a subset functioned at the relational level, a level of thought that we recognise as being similar to the level an expert would reach when solving a problem. With students assigned a level of reasoning we were able to investigate whether or not the way a student approached problem solving affected their use of doodles. We found that there seemed to be no relation between the level of reasoning and the use of annotation by novice student programmers.

This finding was surprising. We expected that M students would find that they needed to use doodles to solve any but the simplest problems and that the higher-level thinkers would have started to use tracing as a strategy when faced with a difficult problem.

We have confirmed many of the findings of the Leeds working group. Our number of students was larger than the number investigated by the Leeds group and therefore provides weight to their findings. By duplicating two of their questions and designing new questions we have been able to take a fresh look at many of their conclusions. Indeed our problem set not only contained different questions but also identified two new question types. Even though our question set was more diverse in question type we confirmed three key findings of the Leeds group:

1. Doodles: code annotations on paper tend to help students arrive at the correct answer.
2. Tracing: students are more likely to trace on fixed code questions than skeleton code questions.
3. Higher achievers are more likely to doodle.

Additionally, we found that certain types of questions elicit certain types of doodles. In particular certain types of questions encourage students to use doodles that are relevant to understanding the code whereas other types or questions do not.

Although we grouped our questions by type and rank it might be argued that certain doodles actually arise because of the constructs in the code rather than the question type. For example it could be argued that tracing would be a more appropriate strategy when the code fragment contained a loop.

Finally, the analysis in this paper highlights a number of interesting questions:

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

## 7 References

- Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Raths, J. and Wittrock, M. C. (eds). (2001): *A taxonomy for learning and teaching and assessing: A revision of Bloom's taxonomy of educational objectives*. New York, Addison Wesley Longman Inc.
- Biggs, J. B. (1999). *Teaching for quality learning at University*. Buckingham, Open University Press.
- Biggs, J. B. and Collis, K. F. (1982): *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. New York, Academic Press.
- Davies, S. (1993): Externalising information during coding activities: effects of expertise, environment, and task. In *Empirical Studies of Programmers - Fifth Workshop*. 42-61. Cook, C., Scholtz, J. and Spohrer J. C. (eds.). Palo Alto, California, Ablex Publishing.
- Fincher, S., Petre, M., Tenenberg, J., Blaha, K., Bouvier, D., Chen, T-Y., Chinn, D., Cooper, S., Eckerdal, A. and Johnson, J. (2004): A multi-national, multi-institutional study of student-generated software. In *Proc. Kolin Kolistelut - Koli Calling*, Koli, Finland, 20-28.
- Fincher, S., Lister, R., Clear, T., Robins, A., Tenenberg, J. and Petre, M. (2005): Multi-institutional, multi-national studies in CSEd research: Some design considerations and trade-offs. *Proc. of the First International Computing Education Research Workshop (ICER'05)*, Seattle, USA. 111-121, ACM Press.
- Fitzgerald, S., Simon, B. and Thomas, L. (2005). Strategies that Students Use to Trace Code: An Analysis Based in Grounded Theory, *Proc of the 2005 international workshop on Computing education research*. Seattle, USA. 69-80. New York, ACM Press.
- Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Mostrom, J. E., Sanders, K., Seppala, O., Simon, B., & Thomas, L. (2004): A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *SIGCSE Bulletin*. 36(4): 119-150.
- Lister, R., Simon, B., Thompson, E., Whalley, J. L. and Prasad, C. (2006): Not seeing the forest for the trees: novice programmers and the SOLO taxonomy, *Proc. of the 11th annual SIGCSE*

conference on Innovation and Technology in Computer Science Education, Bologna, Italy. 118-122, ACM Press.

McCartney, R., Moström, J. E., Sanders, K. and Seppala, O. (2004): Questions, Annotations, and Institutions: observations from a study of novice programmers. *Proc. of the Fourth Finnish/Baltic Sea Conference on Computer Science Education*. Koli, Finland, 11-19.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagen, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I. and Wilusz, T. (2001): A Multi-Institutional, Multi-National Study of Assessment of Programming Skills of First-year CS Students. *SIGCSE Bulletin*. **33**(4): 125-140.

Perkins, D., Hancock, C., Hobbs, R., Martin, F. and Simmons, R. (1989): Conditions of Learning in Novice Programmers. In *Studying the Novice Programmer*. 261-279. Soloway E. and Spohrer J. C. (eds.). Hillsdale, New Jersey, Lawrence Erlbaum Associates.

Petre, M., Fincher, S. and Tenenberg, J. et al. (2003): "My criterion is: Is it a Boolean?": A card-sort elicitation of students' knowledge of programming constructs. *Technical Report 6-03*, University of Kent, Kent, United Kingdom.

Thomas, L., Ratcliffe, M., and Thomasson, B. (2004): Scaffolding with Object Diagrams in First Year Programming Classes: Some Unexpected Results. *Proc. of the 35<sup>th</sup> Technical Symposium on Computer Science Education (SIGCSE 2004)*, Norfolk, VA USA, 250-254.

Thompson, E., Whalley, J. L., Lister, R. and Simon, B. (2006): Code Classification as Learning and Assessment Exercise for Novice Programmers. *Proc. of the 19th Annual Conference of the National Advisory Committee on Computing Qualifications*. Wellington, New Zealand, 291-298

Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P. K. A., and Prasad, C. (2006): An Australasian Study of Reading and Comprehension Skills in Novice Programmers, using the Bloom and SOLO Taxonomies. In *Proc. of the Eighth Australasian Computing Education Conference (ACE2006)*, Hobart, Australia, **52**: 243-252.

## Appendix

The questions have been reformatted to fit into the journal layout.

### Question 1 (Question 5 : Leeds).

Consider the following code fragment:

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

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

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

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

### Question 9 (Question 2: Leeds).

Consider the following code fragment:

```
int[] x1 = {1, 2, 4, 7};
int[] x2 = {1, 2, 5, 7};
int i1 = x1.length-1;
int i2 = x2.length-1;
int count = 0;
while ((i1 > 0) && (i2 > 0)){
    if ( x1[i1] == x2[i2] ){
        ++count;
        --i1;
        --i2;
    }else if (x1[i1] < x2[i2]){
        --i2;
    }
    else
    {
        // x1[i1] > x2[i2]
        --i1;
    }
}
```

After the above while loop finishes, "count" contains what value?

- a) 3
- b) 2
- c) 1
- d) 0

**Question 4: a CL question**

A change of logic question where the Boolean logic is reversed but the code purpose remains the same.

This segment of code checks to see if a number is within a given range. Note:

- **iMIN\_NUM** and **iMAX\_NUM** are both constant integers, where **iMAX\_NUM > iMIN\_NUM**.
- **iNum** is an integer variable;
- **bValid** is a Boolean variable.

```
if ((iNum<iMIN_NUM) || (iNum>iMAX_NUM)) {
    bValid = False;
}else{
    bValid = True;
}
```

Which of the following pieces of code will give exactly the same result as the code above?

a)

```
if ((iNum>=iMIN_NUM) && (iNum<=iMAX_NUM)) {
    bValid = True;
}
```

else

```
{
    bValid = False;
}
```

b)

```
if((iNum>iMIN_NUM) && (iNum<iMAX_NUM)){
    bValid = True;
}else{
    bValid = False;
}
```

c)

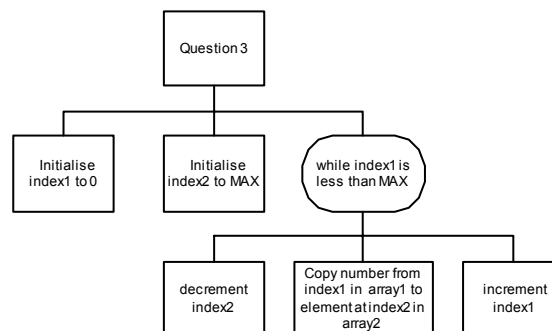
```
if ((iNum>iMIN_NUM) || (iNum<iMAX_NUM)){
    bValid = False;
}else{
    bValid = True;
}
```

d)

```
if ((iNum<=iMIN_NUM) || (iNum>=iMAX_NUM)){
    bValid = False;
}else{
    bValid = True;
}
```

**Question 3: a CR question**

Study the following structure diagram:



Array1 and Array2 are both arrays containing iMAX integers. Which of these code segments correctly implements the logic shown in the above diagram?

a)

```
iIndex1 = 0;
iIndex2 = iMAX;
while (iIndex1 <= iMAX)
{
    iIndex2--;
    iArray1[iIndex1] = iArray2[iIndex2];
    iIndex1++;
}
```

b)

```
iIndex1 = 0;
iIndex2 = iMAX;
while (iIndex1 < iMAX)
{
    iIndex2--;
    iArray1 [iIndex1] = iArray2
    [iIndex2];
    iIndex1++;
}
```

c)

```
iIndex1 = 0;
iIndex2 = iMAX;
while (iIndex1 <= iMAX)
{
    iIndex2--;
    iArray2 [iIndex2] = iArray1
    [iIndex1];
    iIndex1++;
}
```

d)

```
iIndex1 = 0;
iIndex2 = iMAX;
while (iIndex1 < iMAX)
{
    iIndex2--;
    iArray2 [iIndex2] = iArray1
    [iIndex1];
    iIndex1++;
}
```

For the full problem set please see the BRACElet project website at <http://online.aut.ac.nz/Bracelet/repository.nsf/>

## Author Index

Avram, Chris, 97

Berglund, Anders, 97

Berry, Marsha, 55

Bower, Mat, 97

Box, Ilona, 97

Carbone, Angela, 3, 97

Ceddia, Jason, 11

Colton, Don, 141

Cope, Chris, 97

D'Souza, Daryl, 27

Davey, Bill, 97

de Raadt, Michael, 3, 97

De Roeck, Anne, 35

Dobbie, Gillian, 107

Doyle, Bernard, 19, 97

Fekete, Alan, 89

Fife, Leslie, 141

Fitzgerald, Sue, 97

Gribble, Susan J., 163

Haley, Debra T., 35

Hamer, John, 43, 107

Hamilton, Margaret, 27, 55

Harris, Michael C., 27

Herbert, Nicole, 63, 73

Ivins, Jim, 163

Karahasanović, Amela, 81

Kay, Judy, 3, 89

Kell, Catherine, 43

Komisarczuk, Peter, 125

Kumar, P. K. Ajith, 171

Kutay, Cat, 97

Li, Lichao, 89

Lister, Raymond, 3, 19, 97

Litchfield, Andrew, 3

Lutteroth, Christof, 107

Luxton-Reilly, Andrew, 107

Mann, Samuel, iii, 115

Mannila, Linda, 97

Meinicke, Larissa, 147

Pauling, Joel W., 125

Pears, Arnold, 97

Peltomäki, Mia, 97

Petre, Marian, 35

Prasad, Christine, 171

Raban, Richard, 3

Roe, Paul, 3

Santamaria, Daniel, 3

Sheard, Judy, 3, 11, 97

Shepherd, John, 3

Simon, iii, 97, 133

Smith, Lesley, 115

Solomon, Andrew, 3

Spence, Fiona, 43

Stanley, Timothy D., 141

Strooper, Paul, 147

Sutton, Ken, 97

Thomas, Pete, 35

Thomas, Richard, 3

Thomas, Richard C., 81

Thompson, Errol, 155

Tibbey, Grant, 11

Traynor, Des, 97

Tutty, Jodi, 97

Venables, Anne, 97

von Kinsky, Brian R., 163

Wang, Zhong, 73

Whalley, Jacqueline, 171

Xuan, Thanh Quach, 141

## Recent Volumes in the CRPIT Series

ISSN 1445-1336

Listed below are some of the latest volumes published in the ACS Series *Conferences in Research and Practice in Information Technology*. The full text of most papers (in either PDF or Postscript format) is available at the series website <http://crpit.com>.

### Volume 53 - Conceptual Modelling 2006

Edited by Markus Stumptner, *University of South Australia*, Sven Hartmann, *Massey University, New Zealand* and Yasushi Kiyoki, *Keio University, Japan*. January, 2006. 1-920-68235-X.

Contains the proceedings of the Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Tasmania, Australia, January 2006.

### Volume 54 - ACSW Frontiers 2006

Edited by Rajkumar Buyya, *University of Melbourne*, Tianchi Ma, *University of Melbourne*, Rei Safavi-Naini, *University of Wollongong*, Chris Steketee, *University of South Australia* and Willy Susilo, *University of Wollongong*. January, 2006. 1-920-68236-8.

Contains the proceedings of the Fourth Australasian Symposium on Grid Computing and e-Research (AusGrid 2006) and the Fourth Australasian Information Security Workshop (Network Security) (AISW 2006), Hobart, Tasmania, Australia, January 2006.

### Volume 55 - Safety Critical Systems and Software 2005

Edited by Tony Cant, *University of Queensland*. April, 2006. 1-920-68237-6.

Contains the proceedings of the 10th Australian Workshop on Safety Related Programmable Systems, August 2005, Sydney, Australia.

### Volume 56 - Vision in Human-Computer Interaction

Edited by Roland Goecke, Antonio Robles-Kelly, and Terry Caelli, *NICTA*. November, 2006. 1-920-68238-4.

Contains the proceedings of the HCSNet Workshop on the Use of Vision in Human-Computer Interaction (VisHCI 2006).

### Volume 57 - Multimodal User Interaction 2005

Edited by Fang Chen and Julien Epps *National ICT Australia*. April, 2006. 1-920-68239-2.

Contains the proceedings of the NICTA-HCSNet Multimodal User Interaction Workshop 2005, Sydney, Australia, 13-14 September 2005.

### Volume 58 - Advances in Ontologies 2005

Edited by Thomas Meyer, *National ICT Australia, Sydney* and Mehmet Orgun *Macquarie University*. December, 2005. 1-920-68240-6.

Contains the proceedings of the Australasian Ontology Workshop (AOW 2005), Sydney, Australia, 6 December 2005.

### Volume 60 - Information Visualisation 2006

Edited by Kazuo Misue, Kozo Sugiyama and Jiro Tanaka. February, 2006. 1-920-68241-4.

Contains the proceedings of the Asia-Pacific Symposium on Information Visualization (APVIS 2006), Tokyo, Japan, February 2006.

### Volume 61 - Data Mining and Analytics 2006

Edited by Peter Christen, *Australian National University*, Paul J. Kennedy, *University of Technology, Sydney*, Jinyong Li, *University of Southern Queensland*, Simeon Simoff, *University of Technology, Sydney* and Graham Williams, *Australian Taxation Office*. December, 2006. 1-920-68242-2.

Contains the proceedings of the Australasian Data Mining Conference (AusDM 2006), Sydney, Australia. December 2006.

### Volume 62 - Computer Science 2007

Edited by Gillian Dobbie, *University of Auckland, New Zealand*. January, 2007. 1-920-68243-0.

Contains the proceedings of the Thirtieth Australasian Computer Science Conference (ACSC2007), Ballarat, Victoria, Australia, January 2007.

### Volume 63 - Database Technologies 2007

Edited by James Bailey, *University of Melbourne* and Alan Fekete, *University of Sydney*. January, 2007. 1-920-68244-9.

Contains the proceedings of the Eighteenth Australasian Database Conference (ADC2007), Ballarat, Victoria, Australia, January 2007.

### Volume 64 - User Interfaces 2007

Edited by Wayne Piekarski, *University of South Australia*. January, 2007. 1-920-68245-7.

Contains the proceedings of the Eighth Australasian User Interface Conference (AUI2007), Ballarat, Victoria, Australia, January 2007.

### Volume 65 - Theory of Computing 2007

Edited by Joachim Gudmundsson, *NICTA, Australia* and Barry Jay *UTS, Australia*. January, 2007. 1-920-68246-5.

Contains the proceedings of the Thirteenth Computing: The Australasian Theory Symposium (CATS2007), Ballarat, Victoria, Australia, January 2007.

### Volume 66 - Computing Education 2007

Edited by Samuel Mann, *Otago Polytechnic* and Simon Newcastle *University*. January, 2007. 1-920-68247-3.

Contains the proceedings of the Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, Australia, January 2007.

### Volume 67 - Conceptual Modelling 2007

Edited by John F. Roddick, *Flinders University* and Annika Hinze, *University of Waikato, New Zealand*. January, 2007. 1-920-68248-1.

Contains the proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM2007), Ballarat, Victoria, Australia, January 2007.

### Volume 68 - ACSW Frontiers 2007

Edited by Ljiljana Brankovic, *University of Newcastle*, Paul Coddington, *University of Adelaide*, John F. Roddick, *Flinders University*, Chris Steketee, *University of South Australia*, Jim Warren, *University of Auckland*, and Andrew Wendelborn, *University of Adelaide*. January, 2006. 1-920-68249-X.

Contains the proceedings of the ACSW Workshops - The Australasian Information Security Workshop: Privacy Enhancing Systems (AISW), the Australasian Symposium on Grid Computing and Research (AUSGRID), and the Australasian Workshop on Health Knowledge Management and Discovery (HKMD), Ballarat, Victoria, Australia, January 2007.

### Volume 72 - Advances in Ontologies 2006

Edited by Mehmet Orgun *Macquarie University* and Thomas Meyer, *National ICT Australia, Sydney*. December, 2006. 1-920-68253-8.

Contains the proceedings of the Australasian Ontology Workshop (AOW 2006), Hobart, Australia, December 2006.

### Volume 73 - Intelligent Systems for Bioinformatics 2006

Edited by Mikael Boden and Timothy Bailey *University of Queensland*. December, 2006. 1-920-68254-6.

Contains the proceedings of the AI 2006 Workshop on Intelligent Systems for Bioinformatics (WISB-2006), Hobart, Australia, December 2006.