Special Issue: SAICSIT '99
## Contents

**Preface**  
P. Machanick .................................................. 1

**Research Articles**

**Active Learning: Issues and Challenges for Information Systems and Technology**  
RD Quilling, GJ Erwin and O Petkova ........................................ 5

**A Generic Modelling Framework for Interactive Authoring Support Environments**  
Paula Kotze ............................................................. 15

**An Application of a Framework for Evaluation of the Factors Affecting Software Development Productivity in the Context of a Particular Organisational Environment**  
O Petkova and JD Roode ............................................... 26

**An Information-Theoretic Semantics for Belief Change**  
T Meyer ................................................................. 33

**A Complexity Metrics Model for Software Correction**  
A Törn, T Andersson and K Enholm .................................. 40

**A Conceptual Design for High-Volume Data Processing of Warehouse Database into Multidimensional Database**  
Paisarn Trakulsuk and Vichit Avatchanakorn ......................... 49

**A Pragmatic Approach to Bitemporal Databases: Conceptualization, Representation and Visualisation**  
Chiyaba Njovu and WA Gray ...................................... 58

**A Building Recognition System**  
SP Levitt and B Dwolatzky ........................................... 68

**Computer Programming and Learning to Write**  
John Barrow ......................................................... 77

**Co-operating to Learn using JAD Technologies**  
TA Thomas .............................................................. 87

**Critical Success Factors for the Implementation of DSS at a Selection of Organisations in Kwazulu/Natal**  
URF Averweg and GJ Erwin ........................................ 95

**Enhancing the Predictability of Two Popular Software Reliability Growth Models**  
Peter A Keiller and Thomas A Muzzuchi ......................... 105
Generalised Unification of Finite Temporal Logic Formulas
Scott Hazelhurst .................................................. 110

Harmonizing Global Internet Tax: A Collaborative Extranet Model
E Lawrence and B Garner ........................................ 119

Improving Object Oriented Analysis by Explicit Change Analysis
Lui Yu, Siew Chee Kong, Yi Xun and Miao Yuan ..................... 128

Reconciling the Needs of New Information Systems Graduates and Their Employers in Small, Developed Countries
Rodney Turner and Glenn Lowry .................................. 136

Shortest Delay Scheduling Algorithm for Lossless Quality Transmission of Stored VBR Video under Limited Bandwidth
Fei Li, Yan Liu, Jack Yiu-Bun Lee and Ishfaq Ahmad .................. 146

Software Croma Keying in an Immersive Virtual Environment
Frans van der Berg and Vali Lalioti ................................ 155

Some Automata-Theoretic Properties of $\cap$-NFA
Lynette van Zijl and Andries PJ van der Walt ...................... 163

Statistical Analysis of an Automated Computer Architecture Learning Environment
JT Waldron, J Horgan and G Keogh .................................. 168

The CILT Multi-Agent Learning System
Herna L Viktor ....................................................... 176

The Development of a Generic Framework for the Implementation of Cheap, Component-Based Virtual Video-Conferencing System
Soteri Panagou and Shaun Bangay .................................. 185

The Role of Experience in User Perceptions of Information Technology: An Empirical Examination
Meliha Handzic and Graham Low ..................................... 194

What are Web Sites Used for: Cost Savings, Revenue Generating or Value Creating?
Man-Ying Lee ........................................................ 201

New Ideas Papers
Approaches to Video Transmission over GSM Networks
Bing Du and Anthony Maeder ....................................... 210

From Information Security Baselines to Information Security Profiles
Rossouw von Solms and Helen van der Haar ......................... 215

Grounded Theory Methodology in IS Research: Glaser versus Strauss
J Smit ................................................................. 219

Introducing a Continuum of Abstraction-Led Hierarchical Search Techniques
Robert Zimmer and Robert Holte .................................... 223
Multimedia as a Positive Force to Leverage Web Marketing, with Particular Reference to the Commercial Sector  
*Stan Shear* .................................................................................................................. 229

Understanding HCI Methodologies  
*Peter Warren* ............................................................................................................. 234

Electronically Published Papers  

<table>
<thead>
<tr>
<th>Experience Papers</th>
</tr>
</thead>
</table>
| A Java Client/Server System for Accessing Arbitrary CANopen Fieldbus Devices via the Internet  
  *Dieter Bühler, Gerd Nusser, Gerhard Gruhler and Wolfgang Küchlin* ........................................... 239 |
| An Object-Oriented Framework for Rapid Client-side Integration of Information Management Systems  
  *Ralf-Dieter Schimkat, Wolfgang Küchlin and Rainer Krautter* ...................................................... 244 |
| Distributed Operating Systems: A Study in Applicability  
  *Jürgen Prange and Judith Bishop* .................................................................................................. 249 |
| Formal Verification with Natural Language Specifications: Guidelines, Experiments and Lessons so far  
  *Alexander Holt* ....................................................................................................................... 253 |
| Introducing Research Methods to Computer Science Honours Students  
  *Vashti Galpin, Scott Hazelhurst, Conrad Mueller and Ian Sanders* ............................................ 258 |
| Visualising Eventuality Structure  
  *ST Rock* .............................................................................................................................. 264 |

Electronically Published Papers  
SAICSIT'99

South African Institute of Computer Scientists and Information Technologists

Annual Research Conference 17-19 November 1999

Prepare for the New Millennium

Is there life after y2k?

Mount Amanzi Lodge, Hartebeespoort, South Africa

Sponsors

http://www.cs.wits.ac.za/

PHDS research/PHDS.html

Think different.
Preface

Philip Machanick, Overall Chair: SAICSIT’99

Running SAICSIT’99, the annual research conference of the South African Institute for Computer Scientists and Information Technologists, has been quite an experience.

SAICSIT represents Computer Science and Information Systems academics and professionals, mainly those with an interest in research. When I took over as SAICSIT president at the end of 1998, the conference had not previously been run as an international event. I decided that South African academics had enough international contacts to put together an international programme committee, and a South African conference would be of interest to the rest of the world.

I felt that we could make this transition at relatively low cost, given that we could advertise via mailing lists, and encourage electronic submission of papers (to reduce costs of redistributing papers for review).

The first prediction turned out to be correct, and we were able to put together a strong programme committee.

As a result, we had an unprecedented flood of papers: 100 submitted from 21 countries. As papers started to come in, it became apparent that we needed more reviewers. It was then that the value of the combination of old-fashioned networking (people who know people) and new-fashioned networking (the Internet) became apparent. While the Internet made it possible to convert SAICSIT into an international event at relatively low cost, the unexpected number of papers made it essential to find many additional reviewers on short notice. Without the speed of email to track people down and to distribute papers for review, the review process would have taken weeks longer, and it would have been much more difficult to track down as many new reviewers in so little time.

Even so, the number of referees who were willing to help on short notice was a pleasant surprise.

The accepted papers cover an interesting range of subjects, from management-interest Information Systems, to theoretical Computer Science, with subjects including database, Java, temporal logic and implications of e-commerce for tax.

In addition, we were very fortunate in being able to invite the president of the ACM, Barbara Simons as keynote speaker. Consequently, the programme for SAICSIT’99 should be very interesting to a wide range of participants.

We were only able to find place in the proceedings for 36 papers out of the 100 submitted, of which only 24 are full research papers. While this number of papers is in line with our expectation of how many papers would be accepted in each category, we did not have a hard cut-off on the number of papers, but accepted all papers which were good enough, based on the reviews. Final selection was made by myself as Programme Chair, and Derrick Kourie, as editor of the South African Computer Journal. Additional papers are published via the conference web site.

We believe that we have put together a quality programme, and hope you will agree.

Acknowledgments

I would like to thank the South African Computer Journal production team, Andries Engelbrecht and Herna Viktor, respectively from the Department of Computer Science and Informatics, University of Pretoria, for their work on producing the proceedings.

The reviewers listed overleaf did an excellent job: many wrote very detailed reports, sometimes after being called in on very short notice. Inevitably, there were some glitches resulting from the unexpected workload, but the buck stops with the programme chair: I promise to do better next time.

I would also like to thank my own department for putting up with the extra work and expense that running a conference entails. I tried not to burden them with too much extra work, but our secretaries, Zahn Gowar and Leanne Reddy, inevitably had to take on some extra work. John Ostrowick provided valuable assistance with design of our web pages and call for papers poster. Carol Kernick, who handles our finances and membership records, did a fine job of keeping up with the demands of the conference.

Finally, I would like to thank our sponsors, whose contribution made this conference been possible:

- PricewaterhouseCoopers – sponsored generous prizes and the conference banquet
- National Research Foundation (NRF) – provided financial support
- University of the Witwatersrand – provided financial support
- Programme for Highly Dependable Systems, University of the Witwatersrand – provided financial support
- Standard Bank – provided financial support
Editorial

- Apple Computer - provided equipment for the conference
- Qualica - provided technical support including helping with the conference web site

Web Site

For more information about SAICSIT, including a pointer to the conference site, see <http://www.saicsit.org.za>.

Referees

- Department of Computer Science, University of Pretoria
  - Derrick Kourie
  - Bruce Watson
  - Vali Lalioti
  - Andries Engelbrecht
  - Ivan Mphahlele
- German National Research Center for Information Technology - GMD
  - Gernot Goebbels
- School of Math Stat and IT, Natal University Pietermaritzburg
  - Peter Warren
- Graduate School of Business, University of Cape Town
  - Kurt April
- CSIR
  - James Jardine
- Department of Computer Science, Rhodes University
  - Shaun Bangay
  - Peter Clayton
  - John Ebden
  - Richard Foss
  - George Wells
  - Peter Wentworth
- Information Technology Division, Rhodes University
  - Caro Watkins
- Department of Information Systems, Rhodes University
  - Brenda Mallinson
  - Dave Sewry
- Department of Informatics, University of Pretoria
  - Carina de Villiers
  - Herna Viktor
  - Niek du Plooy
  - Elsje van Rooyen
  - Machdel Matthee
  - Alan Abrahams
  - Jackie Phahlamohlaka
- School of Information Technology & Engineering, University of Ottawa
  - Stan Szpakowicz
  - Dwight Makaroff
- Department of Information Systems, Victoria University
  - Glenn Lowry
  - Peter Shackleton
  - Tas Adam
  - Alastair Wallace
  - Stephen Burgess
  - Dave Burgess
  - Julie Fisher
  - Jerzy Lepa
  - Geoff Sandy
  - Rod Turner
  - Alastair Wallace
  - Andrew Wenn
- Peninsula School of Computing and Information Technology, Monash University
  - Ainslie Ellis
- School of Information Management and Systems, Monash University
  - Angela Carbone
- School of Computer Science & Software Engineering, Monash University
  - Dianne Hagan
  - Judy Sheard
- School of Management Information Systems, Deakin University
  - John Lamp
  - David Mackay
  - Philip Joyce
Editorial

- Marie van der Klooster
- Bill Hewett
- Jennie Carroll
- Rodney Carr

- Computer and Automation Research Institute, Hungarian Academy of Sciences
  - Ferenc Vajda

- Department of Computer Science, University of the Witwatersrand
  - Ian Sanders
  - David Lubinsky
  - Conrad Mueller
  - Yinong Chen
  - Bob Baber
  - Vashti Galpin
  - András Salamon
  - Scott Hazelhurst
  - Philip Machanick
  - Zoltán Fazekas

- Department of Electrical Engineering, University of the Witwatersrand
  - Barry Dwolatzky
  - Farzin Aghdasi

- Department of Mathematics, University of the Witwatersrand
  - Jonathan Burgess

- School of Law, University of the Witwatersrand
  - Victoria Bronstein

- Edward Nathan & Friedland Attorneys, Johannesburg
  - Justine White

- Department of Information Systems & Technology, University of Durban-Westville
  - Geoff Erwin

- Department of Computer Science, Clemson University, Clemson, SC
  - Karen Hay

- Logical SA
  - Philip Green

- Department of Computer Science, University of Bristol
  - Alan Chalmers

- Department of Systems and Computer Science, Howard University, Washington, D.C
  - Todd Shurn
  - John Trimble

- Department of Computer Science, University of Cape Town
  - Sonia Berman

- King’s College, London
  - Peter Wood

- Department of Information Systems, University of Cape Town
  - Paul Licker
  - Steve Erlank
  - Alemayehu Molla

- Department of Department of Mathematics and Applied Mathematics, University of Cape Town
  - Ingrid Rewitzky
  - Peter Jipsen
  - Renkuan Guo

- Department of Information Systems, University of the Western Cape
  - Andy Bytheway

- Department of Electrical Engineering and Computer Science, The George Washington University, Washington, DC
  - Rachelle Heller

- School of Biophysical Sciences and Electrical Engineering, Swinburne University of Technology
  - Ian Macdonald

- Electrical and Electronic Systems Engineering, Queensland University of Technology
  - Anthony Maeder

- Computer Science Department, Åbo Akademi
  - Aimo Törn
  - Kaisa Sere
  - Mats Aspnas

- Division of Informatics, University of Edinburgh
  - Graeme Ritchie
  - Arturo Espinosa-Romero
  - Jon Oberlander
Editorial

- Chris Brew
- Alexander Holt

• Department of Computer Science and Engineering, Helsinki University of Technology
  - Tommi Junttila
  - Nisse Husberg

• Technical University of Budapest, Department of Measurement and Information Systems
  - Andras Pataricza

• NASA-Goddard Space Flight Center
  - Nigel Ziyad

• Department of Computer Science & Information Systems, University of Port Elizabeth
  - Janet Wesson
  - Leon Nicholls

• Computer Technology Department, Indiana University Purdue University Indianapolis
  - Tim Price

• Department of Electrical Engineering and Computer Science, The University of Michigan
  - Trevor Mudge

• Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
  - David Forsyth

• School of Computer Science, University of Birmingham
  - Mark Ryan

• Faculty of Mathematics, Computer Science, Physics & Astronomy, University of Amsterdam
  - Carlos Areces

• Department of Computer Science, Universidade do Vale do Rio dos Sinos - UNISINOS Rio Grande do Sul
  - Marcelo Walter

• School of Information Technology & Mathematical Sciences, University of Ballarat
  - Binh Pham
Abstract

Programming is often seen in terms of mathematical models. An additional perspective, that of programming as a means of communication between the programmer and the computer, can also be useful, particularly in teaching novices. Modern writing theories provide a way of exploring this perspective and, along with a developmental model, suggest that recognising a) the qualitative difference between novice and expert cognition, b) the role of coaching in helping a novice become expert and c) the student's zone of proximal development can provide important insights into teaching programming. Unisa, a correspondence-based distance teaching university incorporates these factors in a distance coaching model consisting of the four phases of Observation, Coaching, Abstraction and Application. Students start learning a new concept through writing and experimenting with programs. Depending on their level of understanding, students take different paths through the study material and receive different levels of coaching. From this experiential basis students are led to develop appropriate abstract models of the concepts involved and then apply this understanding to practical problems. This approach is compared briefly to other approaches to teaching novices to program.

Keywords: Teaching programming, information systems teaching, teaching methodology, distance teaching, writing

Computing Review Categories: K.3.2

1 Introduction

Learning to program is hard. As Maheshwari [13] points out, the transition from novice to expert programmer typically takes ten years. Much of the academic teaching of computing stems from its mathematical basis. This ranges from very obvious aspects such as formal program verification through to less obvious aspects such as the different types of diagram used for analysis and design. But programming is not only a mathematical process. It is also about communication and learning to communicate. And so the programmer needs to learn the language of computers and how to use it effectively as well. This perspective, of learning to communicate with an external entity, and the possibility of drawing on the wealth of knowledge in teaching natural language, is considered less often in programming tuition [15]. There are several parallels between learning to write a good expository essay in English and learning to write a good program and this paper explores some of these for insights into teaching programming. (As Robertson & Lee [15] suggest, another parallel is the teaching of second natural languages.)

With natural languages the communication process occurs between two humans. With computers it occurs between a human and a computer, either between programmer and computer or between computer and user. This human-computer communication is highly structured, following clear and relatively inflexible patterns, and the closest parallel in the human-human realm is probably expository ('academic') writing.

Since it can, at times, be useful for a discipline to look beyond its own boundaries, this paper starts by giving a background to writing theory. This brings to light a) the need to see writing as a process, b) the cognitive differences between novices and experts, and c) the existence of a number of different expert writing patterns. A model of student development is then presented briefly to help clarify ways of teaching a novice to become an expert. Since expert characteristics are largely domain-independent [18], this approach is used to develop by analogy a teaching model that concentrates on bridging the novice/expert programmer divide. This model has been used for several years at Unisa1 and its latest incarnation, in the form of a textbook for teaching introductory programming to Information Systems students, is described briefly [3].

2 Writing Theory

2.1 Writing As A Process

For many years, the emphasis in writing instruction was on the final, written document and focused on micro-issues such as syntax. About twenty years ago, the emphasis began to shift from writing as a product to writing as a process. Hayes & Flower [7, 9] proposed a highly influential model making it clear that much expert writing starts with careful planning, is goal-directed, and moves forward in distinct phases that finally culminate in the written document. During writing, many constraints are operational more than a human can satisfy simultaneously and so the expert writer is skilled in partitioning the writing

1The University of South Africa, an open university that teaches through correspondence.
task into subtasks where certain constraints can be put into abeyance while a selected few become dominant. For example, while the writer is busy planning a document concerning about spelling and grammar are temporarily put to one side. Writing is also often not a linear process. The interaction between constraints mean that revision and iteration between phases are inevitable. Hayes and Flower also proposed sets of production rules and flow charts to describe the psychological processes governing these subtasks and the interactions between them.

These concepts are not unfamiliar in the computing world. The repeated experience that one cannot simply sit down and start writing a large computer program and then produce a functional and reliable product led to the rise of structured top-down programming. Planning was seen to be crucial and distinct phases ranging from requirement analysis and system specification through analysis and design to implementation and testing were identified. Each of these phases concentrates on certain aspects and so, for example, concern over programming techniques is largely suppressed during system analysis.

Educators in both these fields realise the importance of goal-oriented problem-solving. They see the production of the end product as far more than simply writing a document or a program, and realise the need for working in phases that concentrate on a subset of activities appropriate to the current phase. Both groups also realise that learning to write a document or program requires considerably more than merely syntax (e.g. [4, 11, 13, 19]).

### 2.2 Knowledge Transformation

As structured programming did in computing, Hayes and Flower's model dramatically changed the way writing was taught. However, it still came in for criticism, and significantly so by Bereiter and Scardamalia [5] who felt that Flower and Hayes did not distinguish sufficiently between 'natural' and 'hard' problems, between novice and expert. They proposed two models of writing. One was called 'Knowledge Telling', a 'natural' way of writing and the only model a novice knows. In some situations knowledge telling is very effective and even expert writers use it for writing in areas they understand well. However, for poorly understood, novel or difficult problems the expert adopts a far more complex and powerful model they called 'Knowledge Transformation' [5, 16].

The difference between these models is fundamentally a question of feedback and iteration. Knowledge telling is a purely feed-forward process. Starting with particular content knowledge, which Bereiter and Scardamalia conceptualise as being manipulated mentally in 'Content Space', this style of writer simply sets the content all down on paper, working linearly from beginning to end. This is dominantly a writer-based telling that concentrates on the content to be communicated with little concern for how the reader will understand that content. The final written document has probably not been revised and is quite likely to be repetitive, inconsistent and incomplete.

Figure 1: Bereiter & Scardamalia's model of knowledge transformation

**Knowledge transformation**, however, makes use of an additional mental structure, 'Rhetorical Space', with bidirectional feedback paths to the content space (Figure 1). This kind of writer is well aware of the difficulty of communicating knowledge and the need for presenting a set of arguments logically and coherently, and so keeps the needs of the reader constantly in mind. The writer may start working with an item of knowledge in content space but will subsequently move to rhetorical space to consider how best to communicate it to the reader. For instance, does this knowledge that the writer wants to communicate need a context in order to be understood by the reader? If so, the writer reverts to content space to retrieve the additional knowledge before returning to rhetorical space to structure the argument. Maybe this is now sufficiently coherent and the writer can commit it to paper (or screen). But the newly-retrieved content may be inconsistent with what has already been written and then the writer will need to resolve the difficulties by continuing to iterate between content and rhetorical spaces. It is this conscious concern over the articulation of ideas and the backward and forward movement between these two spaces that makes this type of writing so difficult but also, through resolving inconsistencies, identifying connections, and so on, that leads to an active knowledge building process for the writer.

In summary, knowledge telling takes existing knowledge and simply presents it. This approach is typical of novice writers but is also totally appropriate where expert writers are presenting material that they already understand well and so is already structured and codified. However, this method fails in dealing with unstructured or unknown content and here the expert writer resorts to knowledge building, iterating between concerns over what to express
and how to express it, in the process structuring and extending his or her personal knowledge. Bereiter and Scardamalia's insight is that this represents a distinct difference between novice and expert. Novice and expert writers are not simply different points on the same continuum. To become an expert writer, a novice must learn an entirely new, additional set of skills, and not just extend an existing set. This may well apply to novice programmers too, in which case our teaching should help students to make this leap.

2.3 The Range of Expert Writing Styles

Models of expert behaviour can have a highly normative influence. Flower and Hayes's model can, and has repeatedly, been seen to suggest that there is one correct way of writing. But, as Galbraith [8] points out while summarising research into the writing process, this is not the case and many experts do not write in a highly structured, goal-oriented way. He suggests that there is a range of expert writing styles characterized by two groups, Classical and Romantic.

Flower and Hayes's model typifies the Classical writer. Sometimes, however, writers do not or possibly even cannot work like this either because they find it difficult to conceptualise their thinking in sufficient detail to structure a document in advance or because the subject area is unknown to them. These writers need to write physically to discover what they are thinking and then they adopt what Galbraith calls the Romantic style. They start by writing freely, allowing their thoughts to stream onto paper (or screen) without concern for overall structure or coherence. (This is much like knowledge telling, and novice writers stop at this point.) The expert Romantic writer however pauses to assess carefully what has been written by switching into a highly analytic phase. The goal now is to identify emerging themes, resolve inconsistencies and in general to clarify his or her disposition towards the topic. Once this phase has progressed far enough, the writer returns to physical writing. Should the topic be one the writer has not previously explored in any depth, the written document may be so rambling and confused that it becomes necessary to start from the beginning again, but this time with the insights culled from analysing the previous writing attempt. Alternatively, it may be possible simply to revise the existing work. Either way, a process of alternation between writing and analysis continues until a coherent and possibly highly original document results.

Galbraith explicitly links the Romantic approach to a semantic and not a storage (or episodic) model of memory. The main consequence is that knowledge is not seen as a collection of stored items in memory but as the result of the interaction between an individual's mental abilities and disposition and an external stimulus. This is similar to the concept of a neural network where the activity that occurs is a consequence of the interaction between an external stimulus and the network structure. In this view knowledge is dependent on the individual's encounters with the external environment and so building new knowledge arises out of doing and experiencing. Without the doing, no fresh knowledge is built.

Novices do not have a store of organised programming knowledge. By analogy with the Romantic model, a productive way of building up a store of systematized programming knowledge would be through encountering and resolving the problems of writing programs by alternating between implementation and analytic reflection. (Programming experts also often adopt this process of learning through doing, analysis and iteration in the prototyping approach to software development.) Thus in both Galbraith's Romantic model and Bereiter and Scardamalia's knowledge building model the doing is the key to the learning that occurs and any teaching model should structure suitable opportunities for this struggle of engagement.

Computing educators too are well aware of the importance of this struggle of engagement, as typified by Winslow's 'The key ... is practice, practice, practice...' [18] and is a central concept in the method presented below.

3 The Zone of Proximal Development

How does a student make the transition from novice to expert writer, whether working in human or computer languages? The work of the Russian psychologist Lev Vygotsky can provide interesting additional insights. Vygotsky [17] makes a strong distinction between a student's actual or completed development and his or her potential development, calling the difference between the two the 'zone of proximal development' (illustrated in figure 2).

The level of actual development is a retrospective view, representing what the student already knows and can do independently and so represents the mature end products of learning. The level of potential development is a prospective view, representing embryonic functions in the process of maturation. It represents skills that the student can perform under guidance or in collaboration with more capable peers. The difference between actual and potential development levels is what Vygotsky terms the 'zone of proximal development' and represents the student's imitative abilities. Since one can only imitate what is within one's potential it also corresponds to the next stage of the student's actual development.

From this basis Vygotsky argued that it is pointless directing any teaching effort within the level of actual development since the student can already undertake this unaided. Teaching should instead be aimed at the zone of proximal development with the intention of helping...
4 The Traditional Computing Perspective

While this paper focuses on lessons to be learned from the literature of teaching novices to write expertly in natural language, there is also a considerable literature on teaching novices to program expertly in computer language. How does this computing perspective compare with the writing perspective? In his overview of programming pedagogy, Winslow [18] considers novice-expert programmer differences and points out that many of the differences are independent of the domain of expertise. It is not surprising, then, that these differences, summarised in figure 3, are in keeping with the models presented by the writing theorists. For example, matching points 1, 2 and 3 in figure 3, Scardamalia & Bereiter [16] conclude that expert writers have and use a variety of interconnected representations (along with appropriate operations) while novices do not. These additional, higher level representations of a document help the expert to record intermediate steps and respond to different levels of text problems without losing perspective on the overall process.

Winslow goes on to say ‘There is a continuum from novice to expert ...’ [18] before presenting three different models of this assumed continuum. But is the assumption of a continuum justified? Scardamalia & Bereiter [16] think not. They argue convincingly that different levels of writing require distinctly different sets of cognitive structures, such as the knowledge telling and knowledge transformation structures presented above, rather than simple incremental expansion of a single set of structures. From this perspective, the novice lacks the high level representations and operations which an expert takes for granted. Thus a novice does not become an expert simply by developing greater skill in his or her current set of abilities, but needs to learn a set of additional skills. Moving from novice to expert is thus a ‘lumpy’ process, and not just a smooth, gradual progression that expands existing knowledge in gentle steps.

We return to the traditional computing perspective later.

5 Making the Transition From Novice to Expert

The view that a qualitative and not merely a quantitative difference exists between expert and novice represents a break with received programming pedagogical practices and introduces a subtle shift in perspective in teaching programming and in helping students to bridge this gap. To accommodate this shift, over the past few years at Unisa we have been exploring the coaching of complete newcomers to programming and, as the following pages describe, have built this, amongst other factors, into our teaching model.

How does a novice make the transition? Bereiter & Scardamalia suggest that novices are best taught through modelling of the missing expert level representations and procedures, through prompting to reflect on their own writing processes, and through peer interaction. A crucial part of this process is for the novice to engage repeatedly in the struggle of articulation encountered while balancing the conflicting demands of the content and rhetorical spaces.

Vygotsky also recommends that the adult (expert) provide extensive coaching to the child (novice). As possible teaching methods to achieve this, he mentions leading questions, problem completion, collaboration with peers, explanation and demonstration.

Thus, for learning to be effective, it is important that students a) can regularly make attempts at applying their knowledge to concrete situations that can prompt knowledge building, b) receive appropriate guidance for activities lying beyond their actual level of development and c) function within their zone of proximal development.

6 The Distance Coaching Model

For our (distance) students we have adopted a distance coaching model for the introductory programming modules in both information systems and computer science that is conducted through written study materials. Its most recent application is in a textbook for teaching introductory
<table>
<thead>
<tr>
<th>Novices</th>
<th>Experts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 lack adequate mental models</td>
<td>have many mental models, mixing and choosing them opportunistically</td>
</tr>
<tr>
<td>2 have only surface knowledge</td>
<td>have deep, hierarchical and layered knowledge with explicit maps between objects</td>
</tr>
<tr>
<td>3 may not use their knowledge as necessary and neglect strategies</td>
<td>apply everything they know</td>
</tr>
<tr>
<td>4 use general rather than problem dependent strategies</td>
<td>in known domains work logically from the givens, keeping general approaches for unknown areas; recognise problems that require known solutions</td>
</tr>
<tr>
<td>5 tend to approach programming through control structures; use a line-by-line, bottom up approach</td>
<td>tend to approach programming through data structures or objects</td>
</tr>
<tr>
<td>6</td>
<td>use algorithms rather than a specific syntax</td>
</tr>
<tr>
<td>7</td>
<td>are faster and more accurate</td>
</tr>
<tr>
<td>8</td>
<td>have better syntactic and semantic knowledge and better tactical and strategic skills</td>
</tr>
</tbody>
</table>

Figure 3: Novice-expert differences in programming

![Figure 4: The four phases of each study unit and their relationship to one other](image)

Delphi [3]. Each study unit (chapter) in the textbook proceeds in the four phases of Observation, Coaching, Abstraction and Application (figure 4).

6.1 The Observation Phase

The observation phase (figure 5) models expert skills and demonstrates new concepts which students can see in operation. Without giving extensive prior explanation or background, each example provides students with functional code that they enter into the computer and then run even though they may not yet understand it fully. In addition to exposing students to new concepts and thus, in Vygotsky’s terms, extending their level of potential development, this gives students the opportunity to read unfamiliar code, speculate on its effect, and then, through running the program, to see how well they had understood it. Thus the observation phase also teaches students to read programs, an important skill that must be taught separately to writing programs [18].

As a simple illustration, in Example 1.1 (the first of the of the textbook) students enter a single programming statement (shown in procedure TForm1.Button1Click in figure 6) to create an application where a button click changes a window’s background colour. Without being given any introductory information on programming and before running it, even a novice can see that the program involves a button click, the form’s colour, and ‘Purple’.

In some cases it is necessary to precede the demonstration by a brief introduction to set the context and, more often, to follow it by some discussion highlighting or clarifying important points. As the loop in figure 5 shows, at times more than one example may be necessary to illustrate the concepts adequately.

In summary, the initial thrust of our teaching occurs through worked examples which, as Maheshwari [13] points out, provide a major source of information to students and adds appropriate explanation, an approach that Linn & Clancy [11, 12] found highly beneficial.

SART / SACJ, No 24, 1999
unit Unit1;

interface

uses
  Windows, Messages, SysUtils,
  Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    {Private declarations}
  public
    {Public declarations}
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form1.Color := clPurple;
end;

Figure 6: Setting a window's background colour with a button click

6.2 The Coaching Phase

Much of the students' learning occurs from the coaching and the challenge of doing that characterises the next phase of this model (figure 7). It starts with a programming activity that encourages a student to independent problem-solving by using the material presented in the observation phase to construct further knowledge. For instance, example 1.1, presented briefly in the discussion of the observation phase above, leads on to activity 1.1 of the textbook. This asks students to place two buttons in a window, each setting the background to a different colour, to disable the active button after it is clicked and enable the other, and to allow activation through hot-keys. The first part of this activity closely mimics the preceding example but each successive part requires greater independence and exploration.

Once the student has solved as much of the activity as his or her current abilities allow, the next step of the coaching phase is self evaluation. This has two purposes. First it prompts students to the expert-type behaviour of reflecting on their approach and on their solution to the problem. Second it helps distinguish the threshold between actual and proximal development. If, on the basis of the self-evaluation, a student is confident that he or she can solve the problem independently, it is possible (as shown in figure 7) to jump ahead to the problem solution. If not, the student receives coaching by working through a set of subactivities that decompose the problem into a series of smaller steps and provide guidance to solving each of these smaller steps and so to building up to a solution of the overall problem. The self-evaluation for activity 1.1 mentioned earlier contains four tests with which the student's program should comply. Matching each of these tests is a subactivity to give optional coaching on whatever aspect the student could not solve.

In this way the student receives guidance in both the detailed skills needed at each step and in an expert approach (eg problem decomposition) to problem-solving. Depending on the context, at the completion of the activity the student may be taken through another example or activity demonstrating or testing additional concepts, or may
6.3 The Abstraction Phase

Having by now worked through a highly practical and concrete presentation of new theory, the student is prompted in the abstraction phase (figure 8) to achieve a theoretical integration of the concepts covered in the previous two phases and so to start developing higher level representations of the programming task. This phase comprises a conventional, though brief, presentation of the underlying theory to consolidate and organise the student’s experiential understanding.

6.4 The Application Phase

The final stage in a study unit (figure 9) is for the student to attempt to solve a set of problems independently. This is an opportunity to practise the skills covered in the study unit and to test the degree to which the study unit has prompted a transfer of skills from the zone of proximal development to the zone of actual development.

If the student has not assimilated the material in the study unit adequately, he or she can repeat it. Once again there will be the opportunity of self evaluation during the coaching phase. If significant development has occurred, the student can now solve the bulk of the activities independently and so take advantage of the flexible structuring of the study units by not repeating unneeded subactivities.

The precise nature of the application phase depends on the teaching context and generally includes appropriate material supplementary to that in the textbook. The second level degree module students are set three assignments each containing five or six programming questions and two theoretical questions. These assignments are marked and returned to students with a tutorial letter containing a commentary on the assignments. The workload in the course is too high to set a programming project as well, so this is kept over until the third level where 20 database programming project. When the course is taught on its own for non-degree purposes the application phase consists of set assignments and a small database project.

7 Evaluation

7.1 Evaluating the Delphi Module and Textbook

This distance coaching model originated in a module teaching Pascal to computer science students. At that stage it underwent a formative evaluation process based largely on focus group evaluation over two successive classes. The formative evaluation resulted in an extensive study guide to accompany a prescribed textbook which had been written for conventional face-to-face teaching situations. Several themes emerged during the focus group evaluations of the study guide:

- In isolation, students found the academic style of the prescribed Pascal textbook intimidating. However, after working through the coaching material in the study guide, and so gaining a practical understanding, they found the textbook accessible and beneficial.

- Students felt encouraged by the study guide to explore and learn for themselves and found this preferable to the more passive sense of studying from a conventional textbook.

- The supportive nature of the study guide addressed students’ fears and concerns about working with computers. Even though they were studying on their own, students did not feel left completely to their own resources. Instead, they received ongoing feedback about their programming abilities and progress.

- It is more demanding and time-consuming working through all the examples and activities that constitute this method but there is also significantly greater learning.

We were not able to find a suitable textbook to meet the need for a new programming module in Delphi scheduled to start in 1998 for information systems students. Encouraged by the students’ favourable reaction to the Pascal study guide, we decided to extend those concepts to the form described above. We wrote a book specifically for our students. This resulted in a second generation of the distance coaching model. The most noticeable extension was the addition of the observation phase as a way of introducing new material to take the role of the textbook in the Pascal module.

It is difficult to obtain an objective measure for the success of the method for a number of reasons. For several

\[\text{Our large student numbers (approx 1000 in 1999) combined with an in-house publishing facility helped make this approach economically viable}\]
years before the introduction of this module, programming had not been part of the information systems syllabus. This module was introduced at a time when the entire degree curriculum was being restructured to create an Informatics speciality and so there is no benchmark from previous years with which to compare it. It is also only one of three modules that constitute the Inf2 course. There is deliberate synergy between these three modules making it difficult to isolate the individual effects of each module, particularly in terms of higher levels of cognitive functioning. The widespread, sometimes remote localities of our distance students complicate both data collection and the standardisation of experimental conditions.

However, informal student response has again been positive. A number of students have volunteered that this has been one of the ‘best’ modules they have taken during their studies, that they wished much more study material in other modules was available in this format, that they had achieved a depth of understanding that carried over into other modules, and that this method facilitated the distance learning process. Our general impression as lecturers has also been that it works well with the pass rate at the end of the first year exceeding departmental expectations.

There has been a pilot study involving twelve voluntary respondents out of a class of forty one students doing a web-based version of the module for non-degree purposes. As part of the survey, the students were asked to rate each of the first eight lessons of the eighteen in the textbook on a four point scale from ‘very weak’ through ‘poor’ and ‘quite good’ to ‘excellent’. All students rated each lesson as either ‘quite good’ or ‘excellent’. Clearly, though, this is a crude assessment based on a small sample and one should be cautious about interpreting these results beyond noting that they tend to confirm the positive feedback received informally from students.

7.2 Theoretical Evaluation

An additional way of evaluating this model is by comparing it with other methods of teaching programming that have been considered to work well.

Working on the basis of Kolb’s Learning Cycle, Howard, Carter & Lane [10] construct each of their structured programming lessons in a ‘Teach around the Circle’ format to ensure that their presentation matches different students’ learning styles. Their circle moves around the poles ‘concrete experience’, ‘reflective observation’, ‘abstract conceptualisation’ and ‘active experimentation’ which, though differing in details of presentation, are remarkably similar in intent to our distance coaching model’s four phases. Allan & Kolesar based their method on the example, imitation and practice’ approach to problem solving as a precursor to learning to program [1], principles which are included within the first, second and fourth phases of our method.

In considering the issue of learning styles in teaching recursion to novice programmers, Wu, Dale & Bethel [19] found that concrete conceptual models are better than abstract conceptual models even for novices with abstract learning styles. This supports the distance coaching model’s progression, starting with the concrete and from there developing the abstract. In passing it is worth noting that our approach is inherently flexible and does not rigidly enforce a single learning style. Some students have reported that for each unit they first study the phase three material (the abstraction phase) to develop a theoretical foundation before working through the examples and activities of the first two phases.

Baldwin [2] points out that while discovery learning is powerful and potentially well-suited to computing, it has effectively been ignored by the computer community. Baldwin structures his discovery learning approach around a set of exercises that students must complete through finding and using their own resources, though they are free to discuss the exercises with the instructor. He reports accelerated learning and increased enthusiasm among students. Maheshwari [13] also comments on the benefits of self-learning mentioning that worked examples, for instance, allow the active learner in particular to construct self-explanations. Linn & Clancy [11, 12] found that the addition of a commentary enhances the value of worked examples significantly. The observation phase of the distance coaching method clearly provides a rich source of worked examples with commentaries. The activities of the coaching phase provide the challenge of discovery learning though, partly as a response to the distance learning context, with subactivities replacing Baldwin’s question and answer sessions. Should students choose to work through the subactivities, this phase also provides a further source of worked examples with commentaries.

The concept of ‘self-teaching’ raises the question of whether, in Mayer’s terms, and in an echo of Bereiter and Scardamalia’s terminology, learning is a matter of knowledge acquisition, with ‘the teacher ... a dispenser of information’, or of knowledge construction, where ‘the learner is an active processor of information who is trying to make sense of the presented information’ [14]. The distance coaching model falls within the ‘knowledge construction’ paradigm, giving students an active role in and responsibility for their own learning.

As a final comment, it is worth mentioning that while the distance coaching model provides a very clear and useful framework, it does not prescribe the content of the examples and activities. The instructor is therefore free to incorporate a variety of domain considerations ranging from overall factors such as determination and ordering of the curriculum through to detailed considerations that have been found helpful when dealing with programming novices, such as a ‘completion strategy’ rather than a ‘program generation’ strategy [6], the use of entire programs rather than mere fragments [13] and the eight guidelines for case studies (recycling, multiple representation, alternative paths, divide-and-conquer, fingerprint, persecution complex, literacy and reflection) suggested by Linn & Clancy [12, 13].
8 Conclusion

Starting from models of expert writing in conventional human languages and the concept of the zone of proximal development, this paper suggests that several of these insights and techniques are appropriate to writing computer programs as well. This leads to three important insights. First, novice and expert are not at different points on the same continuum but have different cognitive functioning. Second, coaching can play a significant role in helping a novice become expert. Third, the concept of a zone of proximal development can help clarify meaningful material with which to work.

These insights help the instructor to pay attention to knowledge that the learner can understand but not yet operationalise and provide a framework for the instructor to use in structuring learning experiences for students. The goal is the acquisition of deep knowledge through the 'difficult' psychological processes of knowledge building rather than the superficial knowledge acquired through the 'easy' process of knowledge telling and so lead to useful ways of thinking about teaching.

Unisa's experiment with this method has been adapted to the demands of isolated distance education. It is pragmatic though not unprincipled, and consists of four phases. First is the observation phase demonstrating a particular set of concepts. Second is the coaching phase which aims to prompt expert programming and problem solving, to evaluate the level of the student's actual development and to coach those aspects lying within the zone of proximal development. The third phase is abstraction prompting the student towards integration of the current set of concepts. Finally comes the application phase giving students the opportunity to exercise their independent programming abilities. Iteration is possible both within and between phases. While it is difficult to measure the success of this approach objectively, student feedback has been strongly positive.

Many of the separate aspects of this method correlate with other approaches to teaching programming to novices though none are known that provide the particular combination offered by the distance coaching model or that specifically address these issues in a distance learning environment.

Acknowledgements

Helene Gelderblom, Carina de Villiers and Ken Halland have all contributed significantly in the development and application of these ideas. The reviewers offered useful suggestions that significantly improved this paper. Melanie Malan helped considerably in finding literature references.

References

Research Article


not comprehensive. For example, what of mechanisms for visualising more than two concurrent events, and also concurrent events that do not have the same start and end times as each other. Extending the analysis would entail defining other ways in which the basic building blocks could be combined into structured eventualities; it would then be a matter of coding a PostScript routine for each additional structure. I argue that the modular approach taken would facilitate this kind of extension.

5.2 Related work

The work presented here is related to work on understanding instructions (e.g. [5],[4]); work on the object-eventuality analogy ([6],[9], [16]; work on visualisation of events and situations (e.g. [8], [10]). However, none of these deal with the issue of extracting and visualising information that is about repeated, extended or multiple eventualities.

There is an obvious connection with work in artificial creativity. We can see music and dance as artforms that exist in the eventuality domain and fine art existing in the object domain. It is clear that one aspect of aesthetic appeal in art comes from the recurrence of themes[7]; the repetition of musical phrases is a good example, while visually, say in a painting, patterns of repetition may be appealing. It would be useful to extend some of the models that exist of creativity[7] to include aspects of repetition and massiness, in both the object and eventuality domains.

5.3 Summary

This paper describes an illustrative segment15 of an approach to representing the understanding of language about event sub-structure. Given a semantic analysis of the repetition described by instructions, it is possible to produce a visual image of this. PostScript's functionality is exploited to provide a simple back-end to semantic analysis that is clean and modular. Additional event sub-structures can be included in a modular fashion.

Although the visualisation has some limitations, it is very useful in providing a means of representing semantic information in a medium that is distinct from the conventional ways of doing this, such as logical forms. Moving to a different medium has advantages that include being able to represent information that could not otherwise be easily represented, such as the passage of time. Also, applying the results of semantic processing to another application can be a good test of the validity of semantic output[8].

References


15Some detail has been omitted in order to simplify the exposition.
Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research notes. However, non-refereed review and exploratory articles of interest to the journal’s readers will be considered for publication under sections marked as Communications of Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript
Manuscripts for review should be prepared according to the following guidelines:

- Use wide margins and 1½ or double spacing.
- The first page should include:
  - the title (as brief as possible)
  - the author’s initials and surname
  - the author’s affiliation and address
  - an abstract of less than 200 words
  - an appropriate keyword list
  - a list of relevant Computing Review Categories
  - Tables and figures should be numbered and titled.
- References should be listed at the end of the text in alphabetic order of the (first) author’s surname, and should be cited in the text according to the Harvard method.

Manuscripts accepted for publication should comply with guidelines as set out on the SACJ web page, http://www.cs.up.ac.za/sacj which gives a number of examples.

SACJ is produced using the LATEX document preparation system, in particular LATEX 2ε. Previous versions were produced using a style file for a much older version of LATEX, which is no longer supported.

Please see the web site for further information on how to produce manuscripts which have been accepted for publication.

Authors of accepted publications will be required to sign a copyright transfer form.

Charges
Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect typesetting, reproduction and other costs. Currently, the minimum rate is R30.00 per final page for contributions which require no further attention. The maximum is R120.00, prices inclusive of VAT.

These charges may be waived upon request of the author and the discretion of the editor.

Proofs
Proofs of accepted papers may be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Letters and Communications
Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words. Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book Reviews
Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement
Placement of advertisements at R1000.00 per full page per issue and R500.00 per half page per issue will be considered. These charges exclude specialised production costs, which will be borne by the advertiser. Enquiries should be directed to the editor.