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

  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 Vaii 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
Melha 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

Experience Papers
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

PricewaterhouseCoopers

Standard Bank

NRF

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 e-mail 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 invite the president of the ACM, Barbara Simons as a 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
• 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

• 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 Deal
  – 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

SART/SACJ, No 24, 1999
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
Statistical Analysis of an Automated Computer Architecture Learning Environment

J. T. Waldron\textsuperscript{a}, J. Horgan\textsuperscript{b} and G. Keogh\textsuperscript{b}

\textsuperscript{a}University of Natal, Pietermaritzburg, South Africa, johnw@msmail.cs.unp.ac.za
\textsuperscript{b}School of Computer Applications, Dublin City University, Dublin 9, Ireland.

Abstract

The level of programming skill achieved by a method of learning introductory computer architecture with a strong emphasis on assembly language programming as part of a Graduate Diploma in Information Technology in the School of Computer Applications, Dublin City University is studied. The extent to which differences between students’ backgrounds and study patterns, including usage of various software tools, influence the learning of assembly language programming is investigated. Information on the practical and academic backgrounds of students was obtained from an application form and questionnaire completed at the end of the course. Students’ usage of the computer and software tools was monitored using log files and all this information was analysed to establish which factors impacted on the students’ performance in programming examinations. Neither previous degree type nor degree class had a significant impact on levels of assembly language programming skill achieved but whether or not the student had experience of one high level language before starting the course was significant. Surprisingly, it was found that those who did not prepare their programs in advance of going to the computer significantly outperformed those who did. The frequency with which students logged on to the system, the typical duration of their logon sessions and the frequency with which they used the XSPIM programming tool did not appear as significant, but the frequency of MIPSMARK test tool use and the order in which assignments were completed relative to other students were important.

Keywords: Assembly language programming, Computer-managed instruction
Computing Review Categories: D.3.2, K.3.1

1 Introduction

This paper analyses the level of programming skill resulting from a method of learning introductory computer architecture with a strong emphasis on assembly language programming as part of a graduate diploma in Information Technology in the School of Computer Applications, Dublin City University. The philosophy of the computer architecture course was that assessment defines the curriculum from the point of view of the student, and that the best way to understand computer architecture is by mastery learning of simple assembly language programming, using automatically corrected invigilated programming exams, moving on to processor design at a later stage.

A large number of automated systems for learning programming have been developed. HANDIN [6] developed at the University of Sheffield, concentrates solely on course management, while others, for example AUTOMARK [2] developed at McAlister University in Ontario, Canada, concentrates on the area of automatic marking. Other systems that have dealt with these problems are CENVIRON [7] and KELVIN [1] both of which were developed at Helsinki University of Technology. However, although there is much computer based learning software which involves some form of self-assessment, there is little work involving serious assessment of solutions to complex problems, where the assessments are significant enough to count towards the student’s grading.

Oliver [5] maintains one can divide the assessment of student programs into two parts: assessment of program correctness and assessment of program style, where style includes program structure, choice of identifier names, maintainability, efficiency, documentation etc. Marking by hand for style is less onerous than for correctness, and that therefore it is sensible to concentrate efforts on devising methods for automating marking for correctness. The automatic assessment of program correctness can be performed by subjecting the programs to sets of test data and then inspecting the output using programs written in some suitable language, whereas marking for correctness by hand is hard and tedious.

An automated system called Ceilidh [3] is used both to give feedback to students and for formal course assessment. The iterative use of auto-assessment was found to result in very high raw marks, 80\% and 90\% being common. Also, the marks tend to be far more tightly grouped (often the case in continuous assessment).

The MIPSMARK system tries to address the issues above by the use of invigilated programming exams. Programs were only checked for correctness, not style since aspects like the meaningfulness of identifier names or comments, the main aspect of assembly language style, cannot
be check automatically. Each program was assessed on a masterly learning basis, but a number of programs were given so that all of them did not have to work in order to obtain a pass mark. This provided a good measure of the differences in ability.

The course was based on the book *Introduction to RISC Assembly Language Programming* [10]. XSPIM, a virtual machine that runs programs for the MIPS R2000/R3000 computers was used since the architecture of the MIPS is an ideal example of a simple, clean RISC (Reduced Instruction Set Computer) machine, which makes it easy to learn and understand. The processor contains thirty two general-purpose registers and a well-designed instruction set. The existence of a simulator for the processor greatly simplifies the development and debugging of assembly language programs. For these reasons, MIPS may become the best choice for teaching computer architecture in the 2000s, just as the Motorola 68000 was during the 1980s.

The philosophy behind the course was to speed up the learning process relative to other MIPS computer architecture courses by enabling the student to start writing simple assembly programs early, without getting involved in laborious descriptions of the trade-offs involved in the design of the processor. It was assumed that the students had never studied computer programming before. The course was given at the same time as a programming course by enabling the student to start writing simple assembly language programs. For these reasons, MIPS may become the best choice for teaching computer architecture in the 2000s, just as the Motorola 68000 was during the 1980s.

The purpose of this study is to investigate to what extent differences between students' backgrounds and study patterns, including usage of various software tools, influence the learning of assembly language programming. Information on the practical and academic backgrounds of students was obtained from an application form filled out by each individual prior to the start of the course and from a questionnaire completed at the end of the course (Appendix A). Students' usage of the computer and software tools was monitored using log files. All this information was analysed to establish which factors impacted on the students' performance in the programming examinations.

In the next section, we describe the software used to during the students' work and in Section 3 the course structure and method of assessment are detailed. Section 4 investigates factors which influence programming performance and in Section 5 the regression model for the variables which are significant in describing the students' performance is presented. A discussion of the conclusions is given in the final section.

## 2 Software Tools Used

The course was associated with MIPS MARK, an automatic program testing system which allows a lecturer to set assembly language programming questions and collect and mark the assignments automatically. The MIPS MARK system, a collection of Unix C-shell scripts, is freely available from the web site:

http://www.compapp.dcu.ie/~jwaldron/itral/cahome.html

The user begins by writing a program into a special MIPS MARK question file, for example:

```
1 # Start of file loop4.a
2 #
3 # Question:
4 # Swap each pair of elements in
5 # the string 'chararray' and
6 # print the resulting string.
7 # There will always be an even number
8 # of characters in 'chararray'.
9 #
10 # Output format must be:
11 # 'badcfe'
12 #
13 #---------------------------------------------
14 # text segment
15 #
16 #---------------------------------------------
17 #
18 .text
19 .globl _start
20 _start: # execution starts here
21 # Any changes above this line will discarded by
22 # mipsmark. Put your answer between dashed lines.
23 #---------------------------------------------
24 # Any changes below this line will discarded by
25 # mipsmark. Put your answer between dashed lines.
26 #---------------------------------------------
27 #
28 #---------------------------------------------
29 #
30 #---------------------------------------------
31 #
32 #---------------------------------------------
33 #
34 #---------------------------------------------
35 # data segment
36 #
37 #
38 #---------------------------------------------
39 #
40 .data
41 chararray: .asciiz "abcdef"
42 end1: .asciiz "\n"
43 #
44 # End of file loop4.a
```

The solution must be typed between the dashed lines as indicated. The code can be tested by loading the file into XSPIM, a programming tool incorporating a source level debugger, and running it. Once the user is satisfied with the solution typing `mipsmark` in a terminal window runs MIPS MARK, which works by running a MIPS assembly language program and searching for a precise sequence of characters in the output. In general, MIPS MARK works by extracting the answer from between the dashed lines and trying it against several different test cases. If the program works for case zero and case one, but fails for case two, a copy of the source code for the third test case is obtained by typing `showcase loop4.a 2` which will give a file called `loop4.2.a`. The XSPIM programming tool can then be used to locate the bug in the code preventing this test case from
3 Course Structure and Assessment

The course lasted twelve weeks with two lectures, one tutorial and a two-hour laboratory session each week. Several simple programs had to be completed each week, and a student recorded success with each assignment using MIPS-MARK. Unsuccessful MIPS-MARK attempts were also logged. Programming ability was assessed by two openbook invigilated programming exams (each two hours in duration) at the end of the course in which plagiarism was impossible. These exams were based on mastery learning of simple programs — if a solution worked perfectly it got full marks, but if it did not assemble or failed some of the test cases no marks were given. Programming exam one (week ten) had three questions on arithmetic instructions, iteration involving character strings and array iteration. Programming exam two (week eleven) had three questions on logical, shift and rotate instructions, stack manipulation and MIPS function call mechanisms.

One of the advantages of automatic assessment is that the precise times taken to write a particular assembly program is available, so it is possible to take account of this when assessing programming competency. For the purposes of this study we created a variable (RANK6) by ranking the 87 students firstly by the number of questions solved and secondly by the speed taken to solve those questions. There is a very large difference between a programmer who writes a particular piece of code in an hour, and one who does the same job in ten minutes. Thirty years ago Sackman and his co-workers [8] showed time differences of up to 28 to 1 between pairs of programmers with equal experience. The biggest differences were found to be in debugging times.

There was also a written exam associated with the course comprising twenty multiple choice questions after the end of the course. The overall mark was the average of the written and the programming exams. For the purpose of assessing programming ability we confine ourselves to analysing the score obtained in the programming exams (RANK6). The score was regressed [4] on several sets of variables which, it was suspected, would be instrumental in predicting the students' competence.

4 Factors that Influenced Programming Performance

In this section, we investigate factors which may influence programming performance. In particular we look at the students' background and previous experience, and their approach to and pattern of usage of the programming tools, in order to assess to what extent those factors impacted on programming performance.

4.1 Background and Previous Experience

Our first analysis was to determine if the level of programming skill reached was affected by variations in academic background and previous practical experience: Firstly the primary degree class may be anything from a 2.2 to a 1st (the minimum entry requirement is a second class honours degree); secondly as this is a 'conversion course' in the sense that graduates from different academic backgrounds are eligible for the course, the academic history of students may vary; finally, although no previous computing is assumed in the administration of the course, participants' experience did in fact range from no knowledge whatsoever to experience of up to six high level languages.

We obtained information on students' backgrounds from the application form filled out when applying for a place on the course; this provided data on gender, age and the type and level of the primary degree. In addition, a questionnaire (Appendix A) was administered at the end of the course from which previous computing experience was extracted. This information enabled us to define the variables in Table 1.

RANK6, the performance score, was regressed on these variables to measure the impact of the students' previous experience on programming skill attained. Surprisingly, it was found that neither previous degree type or degree class had a significant impact on levels of assembly language programming skill achieved. The only significant variable to emerge was EXPLANG, whether or not the student had experience of one high level language before starting the course.

Table 2 gives the means and standard deviations of RANK6 for the two categories in EXPLANG; those who never programmed before starting the course and those who had programmed in at least one high level language. It also gives the results of the t-test of difference [9] between the means of the two groups. The null hypothesis is that the means of the two groups are the same, and the alternative hypothesis is that the mean of the group with with no previous language experience is less than the mean of the group who have experience of at least one language. The data were tested for normality using the Kolmogorov-Smirnov test; the group with no language experience had a p-value of 0.484 and the group with at least one language had a p-value of 0.607. This indicates that the data in each group are normal. As can be seen from Table 2, the p-value is 0.007 indicating that those with programming experience before embarking on the course significantly outperform those without.

4.2 Approach to Method

As this is a new method of learning, we were interested in investigating how the students go about writing the programs. Traditionally, students were encouraged to prepare their programs in advance of computer sessions, as this was thought to encourage the development of a thought out design before problems of syntax and the use of interfaces to various tools were addressed. The students were advised...
to adopt this approach on the Java high level language programming course which ran in parallel, but were neither encouraged nor discouraged to do so for this course. We obtained information on this by including the following questions in the questionnaire administered at the end of the course (see Appendix 1).

1. Did you prepare and write down the programs on paper before going to the computer (Yes/no)? (PREPARE)

2. If yes, about how many minutes did you typically spend on this for each program? (PREPARET)

Surprisingly, when RANK6 was regressed on these variables it was found that those who did not prepare their programs in advance significantly outperformed those who did. The average scores for the two groups and results of the t-test of difference are given in Table 3, where a p-value of .001 indicates that the difference between the two groups is highly significant. The null hypothesis is that the means of the two groups are the same, and the alternative hypothesis is that the mean of the group who do not prepare their programs in advance is greater than the mean of those who do. The data were tested for normality using the Kolmogorov-Smirnov test; the group who do not prepare in advance had a p-value of 0.561 and the group who prepare their programs in advance had a p-value of 0.837. This indicates that the data in each group are normal. As this conflicts with the view that preparing code before going to the computer is the best way to learn basic programming skills, we examined the results in more detail in Table 4 showing the average scores for the two groups broken down by EXPLANG.

From Table 4 we see that the group who have no previous programming experience and do not prepare in advance appear to do better than those who have programming experience but do prepare. Though not significant, the trend is clear and is counter to expectations.

### Table 1: Background Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGE</td>
<td>Age at entry to course</td>
<td>≥ 20</td>
</tr>
<tr>
<td>GENDER</td>
<td>Gender</td>
<td>F/M</td>
</tr>
<tr>
<td>EXPLANG</td>
<td>Experience in at least one high language</td>
<td>Y/N</td>
</tr>
<tr>
<td>DATA</td>
<td>Did you ever use a database?</td>
<td>Y/N</td>
</tr>
<tr>
<td>SPREAD</td>
<td>Did you ever use a spreadsheet</td>
<td>Y/N</td>
</tr>
<tr>
<td>WP</td>
<td>Did you ever use a word processor?</td>
<td>Y/N</td>
</tr>
<tr>
<td>CLASS</td>
<td>Class of Degree</td>
<td>1st, 2.1 and 2.2</td>
</tr>
<tr>
<td>TYPE</td>
<td>Degree type</td>
<td>Four categories</td>
</tr>
</tbody>
</table>

† Degree type was sorted into four categories; maths or engineering, general science including health or environmental science, business including law or politics, and degrees in general arts. This categorisation was introduced into the model by means of dummy variables.

### Table 2: Performance and Previous Computing Experience

<table>
<thead>
<tr>
<th>Mean</th>
<th>SD</th>
<th>No. of Cases</th>
<th>t-value</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No languages</td>
<td>38.06</td>
<td>24.33</td>
<td>57</td>
<td>2.75</td>
</tr>
<tr>
<td>At least one language</td>
<td>54.63</td>
<td>23.08</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4: Exam Score, Preparation and Programming Experience

<table>
<thead>
<tr>
<th>Programming Experience</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you prepare in advance?</td>
<td>45.47</td>
<td>68.15</td>
</tr>
<tr>
<td>Yes</td>
<td>34.04</td>
<td>43.38</td>
</tr>
</tbody>
</table>

### 4.3 Computer and Tool Usage

Our final analysis was to determine if the level of skill achieved in assembly programming was affected by usage patterns of the computer and software tools. It was possible to analyse from log files, the extent to which each student used the XSPIM programming and MIPS Marks test tools; this enabled us to define the variables in Table 5 as measures of system usage.

To examine the impact of usage patterns on performance, RANK6 was regressed on the variables listed in Table 5 and found that two significant variables emerged; T2SOLVE, time taken to solve class exercises relative to the first student who solved that exercise and XTEST, the frequency of use of the MIPSMARK test tool outside laboratory sessions. It was found that T2SOLVE impacts negatively on RANK6 since those who typically solved the class problems in shorter times tended to obtain higher exam scores. Counter intuitively, XTEST the total number of times the students used the MIPSMARK test tool, excluding use of MIPSMARK during laboratory sessions, which is not discretionary, was also found to have a negative effect on RANK6. One explanation for this could be that a student who copied an assignment would only use the MIPSMARK test tool once, whereas a student who did the program from scratch would need to run the test tool more often to produce a working solution. Those who solved problems later on may have relied too much of the
Table 3: Comparison of Performance in Prepare Groups

<table>
<thead>
<tr>
<th>No. of Students</th>
<th>Mean</th>
<th>Stan. Dev</th>
<th>t-value</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you prepare in advance?</td>
<td>No</td>
<td>27</td>
<td>53.87</td>
<td>25.23</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>51</td>
<td>36.24</td>
<td>22.79</td>
</tr>
</tbody>
</table>

Table 5: Software Usage Variables.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTEND</td>
<td>Number of laboratory sessions attended</td>
<td>≥ 0</td>
</tr>
<tr>
<td>AVE-TIME</td>
<td>Average login duration</td>
<td>MINS</td>
</tr>
<tr>
<td>NLOGIN</td>
<td>Number of logins during course</td>
<td>≥ 0</td>
</tr>
<tr>
<td>T2SOLVE</td>
<td>Average time to solve problems given out as class exercises relative to the earliest time any student in the class had solved each problem.</td>
<td>DAYS</td>
</tr>
<tr>
<td>TOTLOGIN</td>
<td>Total login time</td>
<td>DAYS</td>
</tr>
<tr>
<td>XPROG</td>
<td>Number of uses of XSPIM programming tool excluding use during laboratory session.</td>
<td>≥ 0</td>
</tr>
<tr>
<td>XTEST</td>
<td>Total number of times the students used the MIPSMARK test tool, excluding use during laboratory sessions.</td>
<td>DAYS</td>
</tr>
</tbody>
</table>

The frequency with which students logged on to the system (NLOGIN), the average duration of their logon sessions (AVE-TIME) and the frequency with which they used the XSPIM programming tool (XPROG) did not appear as significant determinants of programming competence. Also the number of laboratory session attended (ATTEND) did not significantly affect performance. The graphs shown in Figs. 1-5 show RANK6 plotted against the variables for each of the 87 students. From Fig. 1, we see that most of the students engaged in between twenty and forty login sessions. The amount of time varied greatly; in can be as small as 20 minutes and in some cases as high as 100 minutes (Fig. 2 or Fig. 3). There was also a wide spread in laboratory attendance (Fig. 4) but this did not influence performance in programming tests. Fig. 5 illustrates how despite the wide range of usage patterns of the programming tool, no pattern in performance emerged.

5 Regression Model

From the analyses in the previous section, four variables appeared to impact on performance EXPLANG, PREPARE, T2SOLVE and XTEST. These were regressed on RANK6 to establish their overall effect. The results are given in Table 6.
Table 6: RANK6 Regressed on Significant Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Co-efficient</th>
<th>T</th>
<th>VIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLANG</td>
<td>11.18</td>
<td>2.07</td>
<td>1.05</td>
</tr>
<tr>
<td>PREPARE</td>
<td>-10.88</td>
<td>-2.11</td>
<td>1.07</td>
</tr>
<tr>
<td>T2SOLVE</td>
<td>-3.80</td>
<td>-2.71</td>
<td>1.09</td>
</tr>
<tr>
<td>XTEST</td>
<td>0.40</td>
<td>2.57</td>
<td>1.07</td>
</tr>
</tbody>
</table>

Multiple R = .58  R-Squared = 0.34  F = 9.19  Significance F = .0000.

Figure 2: Performance score and average time logged on.

Figure 4: Performance score and laboratory attendance.

Figure 3: Performance score and total logon duration.

Figure 5: Performance score and programming tool use.
As can be seen from the R-squared value in Table 6, these variables explain over 30 per cent of the variation in RANK6. The coefficient of variable EXPLANG, 11.81, suggests that there was roughly a 12 mark advantage enjoyed by those with previous programming experience, and the coefficient of -10.88 for PREPARE indicates that there was approximately an 11 mark disadvantage suffered by those who prepared their programs in advance on paper. Since the T2SOLVE variable was measured in days, each extra day it took the student to solve the class problems on average, was associated with a drop of 4.24 exam marks. The coefficient of 0.46 for XTEST indicates that each use of the MIPSMArk test tool outside of scheduled laboratory sessions added a little less than one half mark to the students' measured performance. Variable Inflation Factors (VIFs) are also reported, and although some multi-collinearity was present amongst the independent variables, as it was not severe, it was not considered to be a problem.

6 Conclusions and Discussion

The philosophy of the computer architecture course analysed was that assessment defines the curriculum from the point of view of the student, and that the best way of understanding computer architecture is to begin by learning assembly language programming, moving on to processor design at a later stage. The emphasis was on mastery learning of assembly language assessed by automatically corrected invigilated programming exams. One of the advantages of automatic assessment is that precise times taken to write particular assembly programs are available, so it is possible to take account of this when assessing programming competency.

This was a graduate diploma and surprisingly, it was found that neither previous degree type nor degree class had a significant impact on levels of assembly language programming skill achieved. The only significant component of prior experience in determining assembly language programming skill was whether or not the student had experience of a programming language before starting the course.

Traditionally, students were encouraged to prepare their programs on paper in advance of computer sessions, as this was thought to encourage the development of a thought out design before problems of syntax and the use of interfaces to various tools were addressed. However it was found that those who did not prepare their programs in advance significantly outperformed those who did. The group who have no previous programming experience and do not prepare in advance appear to do better than those who have programming experience but do prepare their programs on paper.

Why is advanced preparation such a disadvantage? Perhaps the write, test, rewrite cycle of the automated marking software encourages hacking, or only students with less innate ability need to prepare for what are after all relatively simple programs. Each program typically only involved one control structure, for example a single loop, and one data structure such as an array. The most productive approach to initial phase of learning of a new programming language and associated software tools may be by "hacking" at the machine, rather than investing time writing out solutions with pen and paper. This conclusion is borne out by the fact that the more times a student used MIPSMArk during the semester, the higher the final exam mark obtained. Software engineering techniques involving specification and design, while appropriate for large or complicated programs, may not be helpful during the initial stages of acquiring a new language.

Interestingly, the frequency with which students logged on to the system, the typical duration of their logon sessions and the frequency with which they used the XSPIM programming tool did not appear as significant determinants of programming skill achieved. The average time measured in fractions of days students took to solve the 27 problems given out as class exercises relative to the earliest time any student in the class had solved each problem was found to be significant, as was the total number of times the students used the MIPSMArk test tool. One explanation for this could be that a student who copied an assignment would only use the MIPSMArk test tool once, whereas a student who did the program from scratch would need to run the test tool more often to produce a working solution. In addition, those who solved problems later on may have relied too much of the assistance of the early solvers.
## A Survey Form

**SURVEY OF GDF1 COMPUTER ARCHITECTURE**

This information given on this form will be used solely as part of a study into the factors determining success in computer architecture programming tests.

### Name:

### Usual address during term:

### Approximate travel time to college in minutes:

### Mode of conveyance to college? (Specify):

<table>
<thead>
<tr>
<th>Bus</th>
<th>Train</th>
<th>Bike</th>
<th>Car</th>
<th>Walk</th>
<th>Other (please specify):</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>q</td>
<td>q</td>
<td>q</td>
<td>q</td>
<td>q</td>
</tr>
</tbody>
</table>

### Did you use any of the following before starting the diploma?

<table>
<thead>
<tr>
<th>Spreadsheet (E.g. Excel)</th>
<th>Q</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word processor (E.g. Word)</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>Database package</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>Internet browser (E.g. Explorer)</td>
<td>Q</td>
<td>Q</td>
</tr>
</tbody>
</table>

### Did you program in any of the following before starting the diploma?

<table>
<thead>
<tr>
<th>C++</th>
<th>Java</th>
<th>Pascal</th>
<th>Assembly</th>
<th>Basic</th>
<th>Other (please specify):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td></td>
</tr>
</tbody>
</table>

### Did you prepare and write down the programs on paper before going to the computer? YES Q NO Q

If yes, about how many minutes did you typically spend on this for each program?

### Do you prefer the method of learning and assessment (i.e. automatic correction used on this course) compared with the other technical subjects in the diploma?

<table>
<thead>
<tr>
<th>Paper</th>
<th>The Same</th>
<th>Prefer</th>
<th>Dislike</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Q</td>
<td>Q</td>
<td>Q</td>
</tr>
</tbody>
</table>

### Any other comments or suggestions about the computer architecture course:

---

### References


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

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.