Computer Science
and
Information Systems

Rekenaarwetenskap
en
Inligtingstelsels
The South African Computer Journal
An official publication of the South African Computer Society and the South African Institute of Computer Scientists

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083

Production Editor
Prof G de V Smit
Department of Computer Science
University of Cape Town
Rondebosch 7700
Email: gds@cs.uct.ac.za

Editorial Board

Professor Gerhard Barth
Director: German AI Research Institute
Postfach 2080
D-6750 Kaiserslautern
West Germany

Professor Judy Bishop
Department of Computer Science
University of the Witwatersrand
Private Bag 3
WITS 2050

Professor Donald Cowan
Department of Computing and Communications
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada

Professor Jürg Gutknecht
Institut für Computersysteme
ETH
CH-8092 Zürich
Switzerland

Professor Pieter Kritzinger
Department of Computer Science
University of Cape Town
Rondebosch 7700

Professor F H Lochovsky
Computer Systems Research Institute
University of Toronto
Sanford Fleming Building
10 King’s College Road
Toronto, Ontario M5S 1A4
Canada

Professor Stephen R Schach
Computer Science Department
Vanderbilt University
Box 70, Station B
Nashville, Tennessee 37235
USA

Professor Basie von Solms
Departement Rekenaarwetenskap
Rand Afrikaanse Universiteit
P.O. Box 524
Auckland Park 0010

Subscriptions

Southern Africa:
Annual R32-00
Single copy R12-00
Elsewhere:
$32-00 $12-00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685
Regardless of the size of your computer system it is a strategic tool. We are committed to helping you use it most effectively. All it needs is a little imagination.

UNIDATA
Telephone (012) 45-1152

THERE'S A GIANT IN EVERY ONE OF US
The Department of Computer Science at Pretoria University is inviting applications for two annually awarded bursaries for MSc study. These bursaries are aimed at outstanding students who have a Computer Science Honours degree (or equivalent qualification). As far as possible, one such bursary will be awarded to a South African student, while the other will be awarded to a student from another African state. The hope is to thereby promote interaction between South African computer scientists and computer scientists elsewhere in Africa.

The bursaries are funded by the Department's contract research work and will amount to a minimum of R20000 per year. They are intended to cover costs for tuition, housing, transport, books, and an adequate allowance for other living expenses. Successful candidates will be required to work for 20 hours per week as part of the Computer Science Department's contract research team. The work assigned to each candidate will be aimed at providing him with practical experience in his field of interest.

Study fields for which bursaries will be considered include:
- Computer Networking
- Distributed Systems
- Software Engineering
- Artificial Intelligence
- Intelligent Computer Aided Instruction
- Machine Aided Translation

Students should aim to complete the dissertation that is required for an MSc within two years of study (i.e. one year of full time study). The dissertation should preferably be written in English but it may also be written in Afrikaans.

The continuation of the bursary and adjustments to the allowance will be decided upon annually and will depend on progress. It is a condition of the bursary that, after completing his/her studies, the student should return to his/her country of origin for the same number of years for which the bursary was held.

Applicants may request application forms from:

MSc. Bursaries
The Computer Science Department
University of Pretoria
Hillcrest
Pretoria
0083
South Africa
Editor's Notes

It is with sincere gratitude that SACJ takes leave of Dr Peter Lay who, until recently, was the assistant editor dealing with Information Systems. He has left academia for what sounds like a more gentle lifestyle. (He has gone farming!) Under Peter's stewardship the number of high-quality IS papers in SACJ grew steadily. In general, IS papers tend to be accessible and relevant to a wide spectrum of computer professionals, and the quality of IS papers that have been appearing in SACJ has significantly contributed to the increased interest being shown in the journal by the local computer industry. If this growth in interest is to be sustained, it is urgent and important to find a suitable replacement assistant editor. The ideal candidate should not only be respected as an academic by his peers, but should also be disposed to enthusiastically promote SACJ in the private sector. Since a shortlist of candidates is currently being compiled, I would like issue a general appeal for names that might be included on it. Please contact me urgently if you would like to be considered for the job, or if you would like to nominate someone that you consider to be particularly suitable.

My three year term of office as editor expires in October. I have always considered it a great privilege to hold this position, and as a result, I felt honoured when the SAICS executive committee requested that I stay on for a further term. Nevertheless, I initially declined the request on the grounds that the time-demands of the job were significantly eroding my ability to fulfil other duties. Particularly demanding has been the task of seeing to the typesetting of the various contributions - either by doing it myself, or by ensuring that it is adequately done by someone else. Recently, however, Prof G de V Smit (Riel Smit) at UCT has offered to assume the role of production editor. This generous offer so much changes the complexion of what is being asked of me that I am now both willing and honoured to continue as editor for another term. I am very grateful to Riel for his offer and I look forward to working with him. In future, authors whose papers have been accepted for publication will be asked to liaise directly with him regarding the precise form in which the final contribution should be submitted.

The next issue of SACJ will consist largely of a selection of papers that were presented at the 6th South African Computer symposium. The selection will be based on comments from the referees who, at the time, were asked to adjudicate the papers in terms of their appropriateness for both the conference as well as for SACJ publication. Papers which, in the opinion of one or more referees, required major revision will have to be resubmitted to SACJ for refereeing purposes. Authors will soon be contact in this regard.

At the time of writing, the updated list of "approved" publications for the first half of 1991 had not yet been released by the relevant authorities. For the sake of past, present and future contributors I sincerely hope that SACJ will be on the list when it eventually comes out. However, I have become increasingly aware that there is a real danger of laying too much store on papers published in so-called approved journals as a basis for evaluating and rewarding research. I hope to expand more fully on this theme in a future edition of SACJ. Keep watching this space!

Derrick Kourie
Editor

This SACJ issue is sponsored by a generous donation from UNIDATA
A Method of Controlling Quality of Application Software

T D Crossman
Department of Accounting & Finance, University of Natal, King George Vth Avenue, Durban, 4001

Abstract

This paper describes the inspection team process and explores its value to the application developer. The conclusion reached is that the technique does incur added development costs, but the whole process brings with it direct cost savings, and (as can be illustrated by data from two South African companies) the confidence that systems which are subsequently implemented are of appropriate quality.

Keywords: Inspection teams, software quality, development costs, errors, egoless programming
Computing review categories: D.2.9, K.6.1, K.6.4

1. Introduction

To be competitive within their market sector, organisations need to exploit advances in computer-based technology [13,1]. If quality software is implemented it helps to make this exploitation possible [14]. Unfortunately there is no guarantee that quality software will be implemented by adhering to commonly used development methodologies [11]. There are several techniques available to application developers which help to ensure that quality software will be built. One such technique is the use of inspection teams.

This paper focuses on the inspection team process, not just as a method of controlling the quality of application software, but also as a way of managing the development process and containing its costs. Besides providing a detailed description of the technique, data is cited from two South African companies to illustrate its practical value. One company is a nation-wide retail chain (Checkers) and the other is a large insurance company (Liberty Life).

2. The inspection team process

Since a description of the technique and its value in the software development environment was first documented [5], details of the use and adaptations of its procedures have been published in the literature (e.g. [8, 2, 4, 9]).

Basic philosophy

The inspection technique is a formally defined process for the verification of the software product throughout its development [4]. All software materials are examined at defined phases to assess the current status and appropriateness (quality) of the product. Consequently one of the most important checks made by the inspection team is to decide whether the work-product is eligible to proceed to the next development stage.

Inspection co-ordinator

The inspection process is usually driven by a person who has the responsibility of an inspection co-ordinator. It is this person who ensures that inspection staff are appropriately trained, that inspection teams are appointed, that statistics are gathered on each inspection and that management is kept informed on the effectiveness of the process.

Objective of the inspection team

To help focus the energy of the inspection team on a single goal, the objective of the review is just to assess the appropriateness of a software product. This decision is made after attempting to identify the errors in each deliverable from the development process [3]. It is noted that the value of the technique is diminished if it is used to evaluate the developers, or to recommend to them methods of correcting the errors found [3].

Although this appears straightforward, in practice (e.g. at both Liberty Life and Checkers) it has been necessary for the reviewers to be given a clear definition of ‘error’. This definition has three dimensions. An error is regarded as any condition which:

• will result in a malfunction of the system;
• deviates from the standards of the development department;
• deviates from system specifications.

It is this definition of ‘error’ which ensures that the inspection process is a quality control technique.

Membership of the inspection team

It is suggested that the inspection team be limited to about four members [9]. Large teams tend to expend too much potential productivity, and small teams do not generate the kind of human interaction that is effective in detecting errors [10]. All the members are technical personnel whose primary responsibility is software development. One of these four, who is not directly associated with project, is appointed as a moderator of
the inspection team. This moderator is given the responsibility of ensuring that a good inspection is held (or explaining to management why a good inspection was not held). To help prevent the team becoming too large, but to ensure that the product is exposed to a broad base of expertise, only one person directly involved in the development of the product is included in the inspection team. Typically, this person (sometimes referred to as the systems designer) is responsible for the technical excellence of that system. The other members of the inspection team are, like the moderator, personnel who are not directly involved in the development of the system being inspected.

Timing of the inspections
Obviously when an inspection is held - in relation to the development stages within the project - depends on what type of inspection is being conducted [9]. Although any deliverable from the software development process can be inspected [9], the main thrust of this article is to provide evidence of the value of using the technique to inspect programs. These reviews are held (usually at least one for each program in the system) immediately after each program is syntactically error free, but before any form of unit testing is allowed on that program.

The inspection procedure
Because the inspection team process is well documented [5, 9, 2] only a summary of the procedure is given here.

Initiating the process: The responsibility for requesting the inspection rests with the developers. The appointing of the actual team members is a staff responsibility, usually performed by the inspection coordinator. There are four distinct stages in the total inspection procedure.

Overview meeting: This meeting provides the opportunity for the members of the inspection team who are not directly involved in the project to understand the objectives of the deliverable, and the context in which it was developed (e.g. the position of a program in the overall systems design, and its functional and performance requirements).

Preparation: The individual inspection team members then spend time (usually some 2 - 3 hours over the next few days) preparing for the actual inspection. The objective of this preparation stage is to ensure that the inspectors are familiar with the product to be inspected (e.g. to ensure that they understand each part of a program). It is essential for effective error-seeking that each member be prepared thoroughly for the actual inspection.

The inspection meeting: The following procedural standards are enforced to assist in the effective operation of the inspection team:
- the inspection should be an uninterrupted session lasting approximately 2 hours;
- every piece of logic in the documentation should be covered;
- each team member should participate in the error-seeking exercise.

After recording each error found, the inspection team has to decide whether the product should be regarded as acceptable in its current form. Should this decision be taken, in effect the inspectors are accepting co-responsibility for the product. If they feel uncomfortable with taking such a stand, the other options open to them are to ask for revisions to be done - with the possibility that the work be re-inspected if the revisions are significant or, should the situation demand it, to indicate that the product be reworked completely.

(It must be noted that the decision of the inspection team is taken on the quality of the work alone, and does not take cognizance of other factors like the estimated completion dates of the project. Those are management issues which must be resolved after noting the inspection team's decision).

Rework: Because the objective of the inspection team is to find errors in the product being reviewed, it is not their responsibility to become involved in suggestions on how the errors should be corrected. Any rework is done by the members of the project team under the guidance of the systems designer.

3. Evaluation of the inspection team process
The inspection team has been found to be a powerful tool in controlling the quality of software [3,4,5,9]. Like many instruments of power, there are both advantages and disadvantages associated with its use.

Disadvantages of the inspection team process
Overhead and disruption: Sometimes a concern is expressed that inspections slow down the development process. Perhaps this problem is exaggerated by what can become a fanatical desire to meet project deadlines even if this results in the installation of a product of inferior quality. It is, however, a sufficiently serious problem to demand attention. From data collected by the inspection co-ordinators, it is suggested that an overhead (of about 10%) be built into each project estimate to allow for these disruptions [9].

Ego problems: It is claimed that systems developers find it difficult not to regard the deliverables which they have built as an extension of their own personalities [17]. If this happens, the developers may feel that errors identified at an inspection are error being found, not in their work, but in themselves. This can lead to people becoming highly protective of their work, and immediately the effectiveness of the error-seeking activity is reduced. To prevent this requires a conscious effort on the part of managers and participants to remember that the goal of inspections is to help build a product which is error free and, as such, adds to the assets of the organisation.
Management commitment: From the experiences both locally and in the USA [3, 12] there is evidence that the inspection process will not be effective without visible management support. If management hints that this quality control process is a luxury which can be afforded only under ideal circumstances, the authority and potential value of the inspection team will be eroded.

Technology: Compared with program development techniques of the 1960's, interactive programming makes it easy to combine the compilation (or interpreting) and testing stages of program development. Strict procedures are required to ensure that programmers decouple these two stages of program development, and not proceed to testing until the code has been inspected. This has been shown to result in the highest quality product at the lowest cost (see 'The cost savings of inspections' below).

Advantages of the inspection team process
In spite of the problems associated with using the technique, it is possible to identify substantive issues which reinforce the value of this particular quality control procedure. Some statistics used to support these arguments are published in the literature [3, 9]. The particular data used here were extracted from information provided by the inspection co-ordinators at two South African companies where the procedures are currently being used. Among their dissimilarities, the two companies have significantly different experience in the use of inspections. Liberty Life made the decision to use the technique late in 1989, while Checkers have been inspecting programs since 1981. Both companies, however, confirm the procedure's value.

Egoless programming: It was noted that the inspection team is a possible forum for developers to become emotionally protective of the work produced (see 'Ego problems' above). It has been found, however, that inspections can help towards cultivating an environment in which programming cannot be regarded as a private art. Programs which have been subjected to the rigours of the inspection technique cannot be regarded as the secret belongings of an individual. Too many people have contributed significantly to its development for the program to be perceived as an extension of the author's personality. It is more natural in this environment for program development to be acknowledged as a contribution to the assets of the company.

In addition to this, it has been found that contrary to certain expectations, the inspection technique does not detract from the sense of satisfaction from producing quality work (see [12]). Both junior and senior members of staff in the companies using inspections have confirmed that they experience a sense of satisfaction from producing a product with the level of quality which is appreciated by their colleagues. (Of course the converse is also true. There is a sense of failure if work has to be redone because colleagues have found its quality is unsatisfactory).

Project control: One of the responsibilities of the project manager is to establish the current state of systems development [7]. This information enables projects to be controlled [9]. In the inspection team environment it is not necessary for programmers to try to convince their managers that 'programming is now 60% (or 80% or 90%) complete'. Programs being developed can be placed into one of three categories: development not yet started, being developed, or passed inspection.

Passing the inspection is a significant phase of program development. It is an objective method of ascertaining the current status of systems development. As a direct consequence of this, the inspection process assists in the controlling (and consequently the managing) of projects. A second project control benefit has been identified. The technique provides a perfect forum for work to be checked for its adherence to standards. The inspection team specifically looks for violations of those rules and conventions on which a development department decides to standardise (See definition of 'Error' above).

Quality assurance: This is the area where the benefits of inspection teams are automatically expected. The following reasons for these benefits have been identified [4]:

i) Applications are exposed to a broader base of expertise during the earlier stages of development.

ii) Errors are identified and corrected early in the development life cycle.

iii) There is evidence that not only do programs which have been passed by the inspection team require less maintenance, but that any maintenance that is required is less complex and therefore less expensive.

Reports from Checkers and Liberty Life support these claims. Both companies keep detailed records associated with the process. Some of these data are regarded by the companies as confidential, but permission has been given to provide some information in Table 1. The value of exposing development work to a broad base of expertise early in the development process has been noted in both environments. If the maintenance benefits identified by Ebenau are valid, the inspection process will continue to have a direct influence on the costs of maintenance for as long as the inspected systems are operational (see 'The cost saving of inspections' below).
Table 1: A comparison of the use of inspections at two South African companies during the first quarter 1990.

<table>
<thead>
<tr>
<th>Year inspections first used</th>
<th>Checkers</th>
<th>Liberty Life</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1981</td>
<td>1989</td>
</tr>
<tr>
<td>Programming language</td>
<td>COBOL</td>
<td>Natural 2</td>
</tr>
<tr>
<td>Number of inspections in 3 months</td>
<td>43</td>
<td>17</td>
</tr>
<tr>
<td>Number of errors identified</td>
<td>174</td>
<td>222</td>
</tr>
<tr>
<td>Total hours on inspections</td>
<td>146</td>
<td>160</td>
</tr>
<tr>
<td>Average source statements/program</td>
<td>1277</td>
<td>628</td>
</tr>
</tbody>
</table>

Classification of errors: Because each error found during the inspection process is formally identified, it is possible for I.S. departments to know the types of mistakes their staff are making. This information has value in two areas. Firstly, it provides a clear indication of potentially sensitive areas to both the members of the inspection teams and the maintenance staff. Secondly, it provides trainers with specific objectives to ensure that staff are equipped to develop quality software. A list of the top ten errors identified by the inspection teams during the first quarter of 1990 at Checkers and Liberty Life have been supplied by their inspection coordinators, and are listed in Table 2. Although there are obvious differences in the development environments and experience in the use of inspections at the two companies, there are some striking similarities in the types of errors being made. Perhaps the three most important factors are the large number of logic errors identified, the number of times development staff left unnecessary code in programs and the high incidence of missing code in programs which are ready to test.

The number of errors in live systems: Some information on the number of errors found in live systems which were subjected to the inspection team process during their development, have been published elsewhere [3, 12, 9]. In each case it is confirmed that the use of inspections makes it possible to instal systems with a small percentage of residual errors. Consequently the amount of effort spent on finding and correcting errors in live systems is significantly reduced. This reduction may be as high as one twentieth of the effort normally associated with this aspect of maintenance [3].

The cost saving of inspections: While a value can be calculated for some of the benefits of inspections, it is unfortunate that perhaps the most significant benefits are those to which it is difficult to attach a money value (see the ‘Conclusion’ below). From the data which has been collected by companies using the technique, it is possible to calculate direct savings in the following areas:

(i) Debugging
Pele and Crossman have both shown that it is less costly to find and correct errors using inspection teams than by purely unit testing the programs [12,3]. The effects of inflation and the difficulty of transferring costs directly from one country to another precludes the possibility of applying these figures directly to the South African development environment of the 1990s. However if the ratios still apply, the cost of finding and correcting errors during development will be approximately two and a half times less expensive if inspection teams are used. This leads to a significant direct cost saving when, (as in the case of Checkers), about 1 000 errors are being identified annually by their inspection teams.

(ii) Maintenance
The effort required to maintain systems which have been inspected during their development (and, therefore, the cost of maintenance) is reduced [10, 3]. This is also confirmed by the data from Checkers, where approximately 40% of the I.S. staff are involved in the maintenance and enhancement of systems (with only 10% - 20% responsible for the correcting of errors in live systems). This figure compares favourably with the approximately 50% of an organisation's total programming effort being devoted to maintenance which was identified in the Lientz and Swanson survey (reported in [15]).

As a direct result of this reduced cost in both debugging time and the maintenance of developed systems, Gilb wrote:

'in spite of the fact that most errors are found during inspections, they only account for a third of the total repair cost' [9].

4. Conclusion
The whole focus of the inspection team process is to ensure that appropriate application software is implemented. Introducing another step in the development life cycle will inevitably incur further development costs. There is, however, evidence of direct cost savings associated with:

• ensuring that all software is exposed to a broad base of expertise throughout its development phases;
• removing errors from each phase of development before they become expensive to identify and repair;
Table 2: The top ten errors found at two South African Companies during the first quarter 1990.

<table>
<thead>
<tr>
<th>Checkers</th>
<th>%</th>
<th>Liberty Life</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing code</td>
<td>24.7</td>
<td>Superfluous</td>
<td>20.2</td>
</tr>
<tr>
<td>Logic errors</td>
<td>19.0</td>
<td>Logic errors</td>
<td>17.1</td>
</tr>
<tr>
<td>Superfluous code</td>
<td>17.2</td>
<td>Standards</td>
<td>16.2</td>
</tr>
<tr>
<td>Incorrect comments</td>
<td>9.2</td>
<td>Incorrect</td>
<td>9.9</td>
</tr>
<tr>
<td>Standards violations</td>
<td>9.2</td>
<td>Not to</td>
<td>9.0</td>
</tr>
<tr>
<td>Full stop incorrect</td>
<td>8.0</td>
<td>Missing code</td>
<td>8.1</td>
</tr>
<tr>
<td>Incorrect initialisation</td>
<td>4.6</td>
<td>Performance</td>
<td>6.3</td>
</tr>
<tr>
<td>Data definition</td>
<td>4.0</td>
<td>Data</td>
<td>2.7</td>
</tr>
<tr>
<td>Incorrect name usage</td>
<td>1.7</td>
<td>Incorrect</td>
<td>0.9</td>
</tr>
<tr>
<td>Incorrect 1/0 instructions</td>
<td>1.1</td>
<td>Incorrect</td>
<td>0.9</td>
</tr>
<tr>
<td>Other</td>
<td>1.3</td>
<td>Other</td>
<td>8.7</td>
</tr>
</tbody>
</table>

n = 174  n = 222

- maintaining an environment where people gain satisfaction from knowing that building applications is building company assets;
- enforcing departmental standards on all development work;
- basing training programmes on identified shortcomings in the skills of the development staff;

The real value of the inspection process, though, is the sense of confidence which management, systems developers and users have from the knowledge that the systems which are implemented are of appropriate quality.

Bibliography

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 papers. However, non-refereed review and exploratory articles of interest to the journal’s readers will be considered for publication under sections marked as Communications or 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 double-space typing on one side only of A4 paper, and provide wide margins.
• The first page should include:
  - title (as brief as possible);
  - author’s initials and surname;
  - 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 on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
• Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.
• 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 in square brackets. References should thus take the following form:
Manuscripts accepted for publication should comply with the above guidelines, and may be provided in one of the following formats:
• in a typed form (i.e. suitable for scanning);
• as an ASCII file on diskette; or
• as a WordPerfect, \text{T\LaTeX} or \LaTeX\ or file; or
• in camera-ready format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies. A styles file is available from the editor for Wordperfect, \text{T\LaTeX} or \LaTeX\ documents.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R20-00 per final page for camera-ready contributions and the maximum is R100-00 per page for contributions in typed format.

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

Proofs

Proofs of accepted papers will 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.

Note that, in the case of camera-ready submissions, it is the author’s responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

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 specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.
Contents

GUEST CONTRIBUTION

Why all the Fuss About Neural Networks?
G Barth ........................................ 4

RESEARCH ARTICLES

The Placement of Subprograms by an Automatic Programming System
J P du Plessis and H J Messerschmidt ......................... 9

An Investigation into the Separation of the Application from its User Interface
J H Greyling and P R Warren .................................. 16

The Physical Correlates of Local Minima
L F A Wessels, E Barnard and E van Rooyen .............. 22

An Efficient Primal Simplex Implementation for the Continuous 2-Matching Problem
T H C Smith, T W S Meyer and L Leenen ................... 28

Concept Network Framework for a Multi-paradigm Knowledge Base
J Kambanis ........................................ 32

PEST - A Microcomputer Pascal Based Expert System Shell
A G Sartori-Angus and R Neville ............................. 39

A Linda Solution to the Evolving Philosophers Problem
S E Hazelhurst ........................................ 44

TECHNICAL NOTE

Knowledge Representation using Formal Grammars
S H von Solms, E M Ehlers and D J Enslin .................. 54

COMMUNICATIONS AND REPORTS

Book Review & Books Received ................................ 57
Cost-effective Visual Simulation Based on Graphics Workstations
A Caduri and D G Kourie ................................... 58
A Method of Controlling Quality of Application Software
T D Crossman ........................................ 70
An Update on UNINET-ZA: The Southern African Academic and Research Network
V Shaw ........................................ 75