CONTENTS

INVITED PAPER
1-6 Software and Hardware Technology for the ICL Distributed Array Processor
   R.W. GOSTICK

ADVANCED TUTORIALS
7-12 On Understanding Binary Search
   B.P. KIDMAN
13-23 Some Trends in System Design Methodologies
   I.T. HAWRYSZKIEWYCZ

SHORT COMMUNICATIONS
24-25 NEBALL and FINGRP: New Programs for Multiple Nearest-Neighbour Analysis
   D.J. ABEL and W.T. WILLIAMS
26 Program INVER Revisited
   D.J. ABEL and W.T. WILLIAMS
27-28 A Comparison between PASCAL, FORTRAN and PL/1
   D.J. KEWLEY

SPECIAL FEATURES
29 Book Reviews
30-31 Letters to the Editor
32 Call for Papers
"This wouldn't have happened, Fenwick, with a Tandem NonStop™ System."

When your computer's down, are you out of business?

You can bank on it. Timing couldn't be worse.
Your busiest season. Customers pounding for service.
All it takes is one small failure somewhere in the system. One
disc or disc controller. One input/output channel.
For want of an alternative, the system is down and business is
lost. Sometimes forever.
Unless you have a Tandem dispersed processing system.

NonStop™ Operation to be sure.
This is the only system you can buy which keeps
on running, right through a failure that would
shut down any other system on the market
today. Including a failure in a disc, a disc con­
troller, an input/output channel or one of the
processors.

Tandem takes even a critical failure
in stride.
No transaction-in-process is lost
or duplicated.
No downtime.
No lost records.
No lost customers.

No question about it.
When you're on a Tandem NonStop™ System, your information
files and your processes are protected from contamination in a
way no other system can match. We've built in safeguards other
suppliers can only dream about. You've heard the war stories about
restart and restore data base on other systems. They're
not exaggerations, but they can be a thing of the past. With a
Tandem NonStop™ System.

Grow without penalty.
Start with only the computer system you need
right now. You can grow it the way a
local system to a global network. And never lose
one cent on your original investment.

Same hardware. Same software. With low cost
additions, not replacements.
It's worth your interest.
Software and Hardware Technology for the ICL Distributed Array Processor

R.W. Gostick*

The ICL Distributed Array Processor (DAP) is a radical departure from conventional computer architectures. Advances in both hardware technology and software technology have been combined to produce a new type of high speed computer.

This paper concentrates on the relationship between the hardware and software, and shows how the various components have been integrated within the 2900 system.

INTRODUCTION

Many of the changes in computer technology during the past few decades have been considered simply as a means of achieving improvements to the classic computer architecture. Changes such as solid state components replacing valves and core have reduced cost and improved performance but the basic distinction between processing and store has remained. Although functional distribution of 'intelligence' has increased, with separate processors for input/output and peripheral control, the heart of all modern mainframes is still the central processor (CPU) with its associated store.

For many applications, typically the demanding scientific applications such as meteorology, it has become apparent that the physical separation of store and processing limits the overall speed of the 'Von Neumann' serial architecture. The DAP design recognises that current hardware uses the same physical devices, semiconductor chips, for both store and processors, and that the physical and logical division is no longer mandatory. By associating, i.e. distributing processing power within the store the restrictions imposed by the store highway speeds may be overcome. (See Flanders et al 1977.)

To turn this simple idea into a productive computing system requires development of both hardware and software within the new architecture. This paper shows how the DAP has combined modern hardware technology with modern software technology through effective integration within the 2900 system.

2. HARDWARE STRUCTURE

2.1 Overall

The term 'array processor' has been used for several different computer designs, ranging from conventional computers with added vector instructions, to attached pipelined units through to the large supercomputers such as the CRAY-1 and CDC Cyber 205. Unlike the vector processors, which use pipelining and slaving techniques to improve the throughput of a single processor, the DAP is an actual array of processors.

In taking the concept of distribution of processing within the store, or active store, it is easily shown that the most cost effective and flexible system arises from a large number of very simple processors. The DAP takes a set of 4096 identical processors, connected as a 64 x 64 array, each with a standard 4K bit store element to make a 2Mbyte store. The processors are unlike the nodes in a distributed processing system in that the processors all perform the same operations at the same time. These activities are controlled by a central unit, the Master Control Unit, (MCU) which handles all aspects of instruction decoding and address generation.

The overall structure, with one control unit and multiple co-ordinated processing units, is generically known as a Single Instruction Multiple Data stream (SIMD) architecture.

2.2 The Processing Element

The philosophy of the DAP, of having a large number of very simple processors, leads to the design of the processing element (PE) shown in Figure 1. As with all computers there are the two components of storage and processing within the new architecture. This paper shows how the DAP has combined modern hardware technology with modern software technology through effective integration within the 2900 system.

<table>
<thead>
<tr>
<th>Table 1. DAP-FORTRAN PRECISIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
</tr>
<tr>
<td>INTEGER</td>
</tr>
<tr>
<td>CHARACTER</td>
</tr>
<tr>
<td>LOGICAL</td>
</tr>
</tbody>
</table>

*Invited Paper
For real applications, however, many operations are required to be performed on subsets of the field or arrays of data. On the DAP these conditional operations are controlled by the activity register. The register acts as a simple switch, effectively inhibiting selected PEs from obeying a given sequence of instructions.

As shown in Figure 2, each processing element has six major connections, four nearest neighbour connections (N,S,E,W) and two highway connections (row and column).

The two distinct types of interconnection have important roles in the solution of problems. Neighbour connections are fundamental to the solution of field equations as found in applications such as meteorology. For such equations the value at one point in the field depends on values at surrounding points. The highway connections are used to communicate between the array and the MCU, and also to provide the valuable 'global testing' facilities shown below.

### 2.3 The Master Control Unit

The master control unit, Figure 2, as its name implies, controls the processing functions of the array. In hardware terms many of the facilities of the MCU are identical to those of any standard computer, such as instruction fetching, decoding, address generation. Most of the instructions, however, are not obeyed by the MCU itself, but are broadcast to the array. Since these bit level instructions are performed repetitively during bit serial processing, the MCU has a special hardware instruction buffer, with automatic incrementing of bit level addresses under control of a hardware DO instruction.

The general purpose 64-bit registers within the MCU perform several important functions. One general class of operations includes those between an array of values and a single (scalar) item. For these cases the scalar is held within the MCU and is broadcast across the highways to the array as required.

The global testing facility provided by the MCU is used by both low level subroutines (e.g. for overflow detection) and by high level statements to control the overall flow of a program. The MCU can test whether a condition is true in all or any processing elements in two cycles. The first cycle uses the highway logic to perform a logical AND or OR across the rows of the array. This results in a single row (64 bits) which is then held in one of the MCU

---

**Figure 2. Master Control Unit**

3. SYSTEM INTEGRATION

One of the major differences between the DAP and previous SIMD machines, e.g. ILLIAC IV (Slotnick et al 1972), is the way in which the DAP has been integrated into a modern serial computer. Traditionally back-end array processors have been connected to their host computers, which provide the user interface, by means of a medium-high speed channel. Since the DAP may be considered as an active storage module, it may be attached as store rather than as a peripheral.

Figure 3 shows how the DAP attaches as a 2Mbyte store module to an ICL 2900 series system.

This method of attachment means that the DAP system automatically benefits from all the facilities available within the 2900 VME/B operating system, with advantages in both system and application development (see below).

In hardware terms the total 2900/DAP system may be considered as a hybrid dual combination, with the DAP store acting as a common memory for the two types of processor. The 2900 system is responsible for all data management, input/output and loading, and in this respect the DAP store acts purely as normal memory. Any data within the DAP store may be accessed by the 2900 Order Code Processor and Store Access Controller. When a DAP program is activated the 2900 suspends serial processing for that process, and activates the DAP at the appropriate entry point. The system will have previously ensured that all code and data segments required for DAP processing have been loaded into DAP store rather than the other store modules. On completion of DAP processing the DAP returns control in the calling process.

Using this common memory system there are no specific overheads associated with loading the DAP, as there would be using peripheral connection.

3.1 Software Architecture

Any user of a 2900 system interacts with the VME/B operating system using the System Control Language (SCL). Within the system full mixed language programming is possible, and the DAP software has been designed such that the DAP-Fortran system appears simply as another programming language. By designing the system in this manner the user sees the 2900/DAP as a mixed language system and is not concerned with the physical relationship between the two types of processor.

Any application which is to use the DAP may be considered as having serial and parallel components, which require 2900 and DAP facilities respectively. Figure 4 shows how such an application is seen within the DAP system. This is, of necessity, a general view, and the

![Figure 3. 2900/DAP Configuration](image1)

![Figure 4. DAP Application Structure](image2)

![Figure 5. DAP Run-time Structure](image3)
ratios of both size of code and amount of processing represented in the various modules will vary between applications. In particular any application with no DAP-Fortran sections will run unchanged on the 2900, using standard 2900 facilities, and retaining the normal user interface. By rewriting modules in DAP-Fortran such an application may be converted gradually to take advantage of the DAP power.

Communication between the serial and parallel components of the program is via standard Fortran/DAP-Fortran COMMON blocks for data, with a simple CALL and RETURN interface. This ensures maximum efficiency by eliminating physical data transfers.

3.2 DAP Software Interface

The integration of the software required for the DAP with the VME/B system required very few modifications to VME/B, since the majority of the DAP specific facilities already existed within the architecture of VME/B. New facilities were added in the form of the VME/B DAP Manager.

The first requirement is to load the DAP segments into a contiguous area in the DAP store. The DAP Manager arranges for non-DAP segments within the store to be relocated until sufficient contiguous store is available. The DAP program is then loaded and locked into the store.

The second requirement is to manage the CALL-RETURN interface between the serial (2900) and parallel (DAP) modules. For each entry point within the DAP program a link is made to the DAP Manager (Figure 5) so that a standard CALL from a high level 2900 procedure is diverted into the DAP Manager. The DAP Manager activates the DAP by passing the details of the target entry point into a special set of registers within the DAP store access controller (DAC). These registers form part of the standard 2900 'image store' mechanism. The Manager calls the 2900 event system to prepare for the end of DAP processing and suspends the host program.

On completion of DAP processing, either normally or abnormally, the DAP sets the image store registers corresponding to the type of completion, and interrupts the 2900. The interrupt is passed via the event system either to the calling program, which resumes at the CALL point, or else to the diagnostic system (see Section 4). On final completion of the host program the DAP segments are unlocked, and the store may be reused for further DAP or 2900 use.

3.3 DAP Simulator

The feasibility of the DAP was established using a pilot 32 x 32 PE array attached to a host emulator. During the building of the production DAP hardware and software system it was advisable to have a system to test the software independent of the actual hardware. Such a system would have the dual advantages of being able to test the large software suite and act as a reference for the performance of the actual hardware.

A software simulator for the DAP had been written as an earlier tool. The system simulated a general N x N DAP at the bit level, and was used as the basis for the production version of the simulator. The basic action of the simulator is to take the individual instructions from the DAP program, decode, and 'obey' them using standard 2900 instructions. Array instructions are typically performed using 4096 bit (512 byte) string operations.

The simulator is designed to use the full DAP compiling system, and programs are prepared in the standard manner. At execution time, the user indicates that he wishes to use the simulator rather than the actual DAP hardware. Loading takes place as for a normal DAP program, but the final stage of moving segments into the DAP does not take place. The simulator takes the place of the DAP Manager in Figure 5, and when a CALL is made to a DAP entry point the simulator is activated. By building the simulator into the existing system, all the facilities of the actual DAP software are available, including the full diagnostic system (see Section 5). The speed of the simulator is, of course, slow compared with the DAP, by a factor of $10^3 - 10^4$, and hence it is regarded as an aid to testing rather than a substitute for the hardware.

Further facilities have been built in to the simulator to examine the low level behaviour of the DAP. Tracing is performed automatically at the instruction level, and at the end of each run histograms may be produced showing the number of times each group of one or more instructions has been obeyed. The simulator also gives an estimate of the execution time expected on the hardware.

The simulator is currently being used both internally as a design aid for future DAP designs and as a program development tool for 2900 sites without DAPs.

4. DAP-FORTRAN SYSTEM

The language and use of DAP-Fortran have been described in several earlier papers (Gostick 1979a, Flanders et al 1977). This paper concentrates on its implementation, and in particular the way the system takes advantage of and integrates with the VME/B system.

4.1 Compiling Strategy

The DAP hardware, with its bit level processing, provides great flexibility at the software level. To take advan-
tage of this flexibility it was decided to use a two stage compiling system. At the top level the DAP-Fortran compiler is responsible for the syntax and semantics of the language, while at the bottom level a large set of assembler written subroutines take care of the actual implementation of the arithmetic and other facilities. Figure 6 shows the actual stages used in the creation of a DAP program (N.B. the SCL system hides the actual stages from the novice user).

The output from the compiler, instead of code, is a set of macro calls to the assembler. These macros, called the assembler macro format (AMF), correspond loosely to the instructions on a conventional multiple address machine. A typical call is of the form

\[
\text{ADD MM } X = M1 \quad Y = M2 \quad R = M3 \quad A = 1 \quad L = 32 \quad T = E
\]

which is the call to add two matrices of 32 bit (L=32) floating point (T=E) words, pointed at by registers M1 and M2, and store the result in the address pointed to by M3 under control of the activity register (A=1). On a serial 3 address machine this corresponds to a register-to-register add, but on the DAP complete matrices are being added.

The AMF is interpreted by the assembler, which uses the set of built-in macros to generate either direct code, for simple cases such as assignment, or calls to specific subroutines for complex operations. The library of subroutines caters for all the combinations of operations, numeric formats and word lengths, and hence represents a large body of code.

Splitting the overall compilation system into these differing stages gives several major advantages during the early life of a product such as the DAP. For the initial development the overall task could be broken down into several distinct products, with clearly defined interfaces, which simplified the management of the project. Similarly, significant enhancements such as the multiple data lengths could also be treated as separate developments in the compiler and subroutines.

Investigations into new compiler techniques or new optimisations could be carried out by replacing one or more of the system supplied AMF macros during the assembly stage, and only if the tests were successful would the compiler or production AMF macros be changed.

The DAP assembler language, APAL, is used to write the low level subroutines and to process the compiled AMF. It is also available to users, although the facilities in DAP-Fortran necessitate descending to APAL only on rare occasions, usually when writing small utility functions.

### 4.2 Implementation and Diagnostics

To aid the user working with several languages on the 2900, many of the facilities provided by the various compilers are common. To support this at the implementation level, VME/B provides a common compiler environment (CCE), which handles many of the aspects of the user interface. Facilities such as the SCL interface, reading files, generating listings and error messages all use the CCE, which eases the task of the developer as well as the final user. Since all compilation for the DAP is performed by the 2900, the CCE can be used for the DAP compiling system.

A further facility common to high level languages on the 2900 is the diagnostic system, OPEH (Object Program Error Handler). The objective of the system is that when a program fails the same quality of diagnostics should be provided, regardless of the language or languages used for the program. A typical high level report is shown in Figure 7.

The OPEH system is simple in concept, and powerful in its realisation. The use of the system depends on information provided by the various compilers. By default, each compiler produces along with the loadable code (Object Module Format) a diagnostic module for each program or procedure. The diagnostic module contains information which relates the code to the original source of the program. When a program is running normally these diagnostic modules remain on disc with no consequent overhead on the system. If a program aborts, or if the program itself generates a request for diagnostics, two components of OPEH are activated. The first, language independent, component analyses the cause of failure and examines the state of the program stack. The current instruction address indicates the active module and hence the relevant diagnostic module. This module in turn indicates the language used to generate the code module. Specific to each language in the system are language dependent modules within OPEH. These modules are used to analyse the state of each active subroutine in terms of the original language, with the appropriate data types and relevant formats for diagnostic printing.

At each stage of the process the user has total control
of the system through options which can be set at compile time, execution time or even dynamically during the run. Options range from inhibiting all printout, through to printing the values of every variable in every active routine in the program. A further option allows the user to handle his own error recovery for some or all classes of errors (the Errortrap facility).

The task of interfacing the DAP compiling system to the OPEH system is similar to that for any other language. The DAP-Fortran compiler and assembler generate diagnostic records along with the loadable OMF.

When the DAP program meets an error condition the interrupt generated in the 2900 contains a simple error code. Dependant on the type of error and the options currently in use the DAP OPEH modules are loaded and the software stack within the DAP is examined before the 2900 stack. By making use of the OPEH system in this way, the DAP immediately has the benefits of modern diagnostics, and the DAP system appears as another component of the mixed language system, as required in Section 3.1.

The DAP language dependant modules within OPEH and the DAP-Fortran and APAL languages contain specific facilities for array processing and bit level processing. Details of these facilities are given in (Gostick 1979b, ICL 1979).

SUMMARY

The DAP has used the concept of SIMD architecture to provide processing power within the store of a conventional computer. The implementation has taken full advantage of the advanced software facilities available within the host computer to provide a full high level system. The close connection between the serial and parallel hardware and software systems optimise throughput of the total system while retaining the familiar user interface.

REFERENCES

5. DAP Fortran Language ICL TP6918 (1979).

BIOGRAPHICAL NOTE

R.W. Gostick graduated from Imperial College in 1971 with a B.Sc in Physics and subsequently gained a Ph.D in Applied Physics. In 1975 he joined ICL as a Consultant on scientific computing. Since 1977 he has worked as a Consultant with the Distributed Array Processor Marketing Unit, concerned with applications of the DAP to scientific computing.
On Understanding Binary Research

B.P. Kidman*

A formal approach to specifications and verification leads to the identification of underlying basic search schemes and hence to precise understanding of different versions of binary search including uniform binary search. Programs based on these schemes are simple, general and adaptable.

Keywords and phrases: Binary search, program understanding, program verification.

1. INTRODUCTION

Binary search is a well known technique and many different versions of the algorithm have been published and used (Knuth, 1973). The underlying principle is quite simple, but that the precise details can be troublesome is demonstrated in inadequacies or errors in published programs, further discussed subsequently. Moreover, the adaptability, the assumptions and even the correctness of any given program are not obvious from visual inspection.

The value of a more formal approach to specifications and correctness is examined in this light. Precise formulation of the loop invariant is itself useful documentation particularly as its form is closely related to those aspects of the program which seem most difficult to get right. Identification of basic search schemes is based, in part, on the underlying invariant assertions. Correct programs can be derived easily from the basic schemes, and if necessary, tailored to specific requirements and tuned for efficiency while maintaining correctness. Verification is straightforward.

This study has also been extended to a less well known form of the algorithm, uniform binary search (Knuth, 1973), whose verification has not been previously published.

The paper assumes that the reader understands the basic terminology and practice of program verification by the classical method of inductive assertions. Interested readers without this background could first read one or more of Alagic and Arbib (1978), Hantler and King (1976), or Kidman (1978), for example.

We express algorithms in Pascal, and assertions in the notation of predicate calculus; for reasons of clarity examples are presented as general Pascal programs rather than more general schemes. In logical assertions we abbreviate the Pascal $div$ by $\div$, and the fact that table $R[b:t]$ has elements ordered in non-descending order is expressed by the conventional predicate, ordered $(R,b,t)$. The symbol $\&$ is used for logical conjunction and $\Rightarrow$ for implication.


2. BASIC SEARCH SCHEMES

Identification

Stated loosely, the given problem is to search table $R$ for an element (with key) equal to the search key $x$. More precisely and generally, the ordered table or table segment, $R[b:t]$, may be empty and may or may not contain more than one element with a particular key; the search for an element (with key) equal to $x$ should locate the index of the element if found, and also the position where $x$ would be inserted into the ordered table.

Bottenbruch (1962) first published a program along the lines of these algorithms, which work by keeping upper and lower table indices, $i$ and $j$, say, to define the segment of the table remaining to be searched. At each step, $m$ is selected in the middle of $i,j$, $x$ is compared with $R[m]$ and

```
scheme(1)
----------
i:=b; j:=t;
while i<j+1 do
    (assertion I1)
    begin m:=(i+j)div2;
    \{assertion I1\}
    \{i<m\&<j\}
    if x<R[m] then
        j:=m-1
    else
        i:=m+1
    end;
    if i>b then
        found:=x=R[i-1]
    else found:=false;

scheme(2)
----------
i:=b-1; j:=t+1;
while i<j do
    (assertion I2)
    begin m:=(i+j)div2;
    \{assertion I2\}
    \{i<m\&<j\}
    if x<R[m] then
        j:=m
    else
        i:=m
    end;
    if i>b then
        found:=x=R[i-1]
    else found:=false;

scheme(3)
----------
(b<>0) i:=b; j:=t+1;
while i<j do
    (assertion I3)
    begin m:=(i+j)/2; \{i<m\&<j\}
    \{b<>0\}
    \{assertion I3\}
    if x<R[m] then
        j:=m
    else
        i:=m+1
    end;
    if i>b then
        found:=x=R[i-1]
    else found:=false;
```

Figure 1. Basic scheme programs

"Copyright © 1981, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted; provided that ACJ's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society."

"The author is with the Department of Computing Science, University of Adelaide, Adelaide, SA 5001. Manuscript received 20 June 1980 and revised 18 August 1980."

i or j adjusted accordingly. The search loop terminates when the unsearched segment is completely reduced to at most one element, whose equality with x (tested after the loop) defines the ultimate outcome of the search.

With m defined as \( L \left( \frac{i+j}{2} \right) \), we identify three basic schemes (see Figure 1) differing in the relations between i, j and m; the Pascal programs in Figure 1 are based on the general schemes but use the specific relational operator \(<\) in the comparison of x with \( R[m] \). The operator will be \( >\) for a table with reversed ordering; using \(<\) causes the element with the highest index to be selected from a group with equal keys, using \( <\) with that with the lowest index.

In all schemes, \( j—i \) is reduced at each step of the iteration ensuring loop termination. Any integer division operator may be used for computing m in schemes (1) and (2), but scheme (3), which appears, in one sense, to be a combination of (1) and (2), is associated with the calculation of m by the operation \( L \left( \frac{i+j}{2} \right) \). We note that in many programming languages (including some popular implementations of Pascal), the integer division operator truncates the quotient towards zero, leading to a different result quotient when \( i+j \) is negative. For this reason, in the scheme (3) Pascal program in Figure 1, the table indices are restricted to non-negative values by the initial constraint \( b \geq 0 \). It should be emphasised that the other symmetric "combination" of schemes (1) and (2) is only a valid terminating program if m is calculated as \( \left( \frac{i+j}{2} \right) \).

Loop invariants and verification

Loop invariants for these programs are given below. In each assertion the first two terms closely parallel the conditional resetting of i and j in the program loop, and in expressing the current state of the search they can be viewed as the principle components of the invariant. With reference to scheme (2), in an intermediate state of the search we have \( R[i] \leq x < R[j] \), on the assumption that both branches of the conditional have been exercised.

11: \[(i<j \Rightarrow x<R[j+1]) \& (j>b \Rightarrow x>R[i-1]) \& j \leq t \& i \geq b \] & \( b < t \Rightarrow j = i + 1 \)

12: \[(i<j \Rightarrow x<R[i]) \& (j>b-1 \Rightarrow x>R[i]) \& j \leq t+1 \& i \geq b-1 \& b \leq t \Rightarrow j = i - 1 \]

13: \[(j<i \Rightarrow x<R[i]) \& (j>b \Rightarrow x>R[i-1]) \& j \leq t+1 \& i \geq b \& b \leq t \Rightarrow j = i \]

The conjunction of loop invariant and termination condition gives us the output assertion. For example for (2), on loop termination, \( i = j-1 \), and from the last term of 12, \( i \leq j-1 \), hence \( i=j-1 \), and hence we have

\[ j < t+1 \Rightarrow x < R[j+1] \] & \( j > b-1 \Rightarrow x > R[i] \].

This, with the fact that \( R \) is ordered, is sufficient for us to determine whether \( x \) is in the table in one test against \( R[i] \); we note an error in this regard in a tested program (McKeeman, 1974).

The validity of these assertions is simple enough to check by visual inspection; for illustrative purposes we give, for (2), one of the formal verification conditions (for the path inside the loop corresponding to \( x < R[m] \)) and justification for its validity.

```
begin ra := (i+j)div 2;
if x < R[m] then
  m := (i+j)div 2;
else
  if x > R[m] then
    l := m
  else
    found := true
end;
```

```
Figure 2. Adaptations (a) and (b) of scheme (2).
```

The axioms for integer division are assumed in the proofs

scheme (1): \( i < j+1 \Rightarrow i \leq (i+j)/2 \leq j \)
scheme (2): \( i < j => i < (i+j)/2 < j \)
scheme (3): \( i < j \Rightarrow i < (i+j)/2 < j \). The axioms for schemes (1) and (2) will hold for any integer division operator, that for scheme (3) holds for \( (i+j)/2 \).

Also for all three schemes, from this same basis it follows that \( j—i \) is reduced over each loop iteration, and hence we can establish formally that loop execution terminates. For example, over the path whose verification condition was given above, the resetting of \( j \) to \( \left( \frac{i+j}{2} \right) \) always results in reducing the search interval (see the second axiom).

3. MODIFICATIONS TO BASIC SCHEMES

Any of the basic schemes (Figure 1) can be transformed into various program schemes in which loop iteration is somehow terminated when the table element tested happens to equal \( x \), before the binary search is completed; many published algorithms are of this kind. Below we categorise these and in Figures 2 and 3 give program examples based on scheme (2) of Figure 1. The programs could, of course, have been based on schemes (1) or (3).

(a) Loop termination is determined by an additional condition kept in a boolean flag (Wirth, 1973).

(b) Loop termination is controlled by an explicit test for equality in the loop control condition (Wirth, 1976).

(c) The loop is controlled by the simple comparison of \( i \) and \( j \) as in the basic schemes, but should \( x = R[m] \) a special value of \( j—i \) is set which will terminate the loop (Wirth, 1976).

(d) Various non-structured forms of binary search with the underlying search controlled as in the basic schemes described above (Knuth, 1973).

The loop invariants for these programs are also given below.

(a) \( b = c+1 \); \( j = t+1 \); \( found = false \);

\( b < c \)

while not found and \( \{ i < j—1 \} \) do

(repeat IA2)

begin m := (i+j)div 2;
if x < R[m] then
  j := m
else if x > R[m] then
  l := m
else
  found := true
end;

(b) \( b = c \)

\( i = b—1 \); \( j = t+1 \);

if not found and \( \{ i < j—1 \} \) do

repeat

begin m := (i+j)div 2;
if x < R[m] then
  j := m
else if x > R[m] then
  l := m
else
  found := true
end;

Binary Search

(c)

\[ i := b - 1; j := t + 1; \]
while \( i < j - 1 \) do
\{assertion IC2\}
begin \( m := (i + j) \div 2 \);
if \( x < R[m] \) then
\( j := m \); else begin
\( i := m; \) if \( x = R[m] \) then \( j := m \); end;
end;
found := \( i = j \);
end;

(d)

\[ i := b - 1; j := t + 1; \]
while \( i < j - 1 \) do
\{assertion ID2\}
begin \( m := (i + j) \div 2 \);
if \( x < R[m] \) then
\( j := m \); else
if \( x = R[m] \) then \( j := m \);
end;
end;
found := \( i = j \);

Figure 3. Adaptations (c) and (d) of scheme (2).

We note again their close relationship to the program loop statement. IB2 is identical to 12 of the basic scheme which has the same loop contents, but a repeat loop is required in (b) to ensure that \( m \) is defined for the control test; thus, unlike the while programs, (b) will not correctly handle an empty table. The invariants for (a), (c) and (d) now include the term \( i > b - 1 \Rightarrow x > R[m] \) rather than \( i > b - 1 \Rightarrow x > R[m] \) of the basic scheme (2).

IA2 : \( j < t + 1 \Rightarrow x < R[j] \) & \( i > b - 1 \Rightarrow x > R[i] \) & found \( \Rightarrow x = R[m] \)

IB2 :  like IB2

IC2 : \( i \leq j - 1 \Rightarrow (j < t + 1 \Rightarrow x < R[j]) \) & \( i > b - 1 \Rightarrow x > R[i] \)

ID2 : \( j < t + 1 \Rightarrow x < R[j] \) & \( i > b - 1 \Rightarrow x > R[i] \)

4. UNIFORM BINARY SEARCH

In uniform binary search (Knuth, 1973), the search interval is defined, not by its upper and lower index bounds, but rather in terms of one index \( m \), marking the middle of the unsearched table segment, and the half width \( f \), of that segment. While the basic principle of this is clear, making the necessary adjustments to \( m \) and \( f \) requires extreme care. At the start of the \( k \)-th step, roughly \( f \) is \( n/2^k \), and the search interval the index range \( (m-f, m+f) \). At each step \( m \) is adjusted to roughly \( m-f/2 \) or \( m+f/2 \) and \( f \) is approximately halved. Clearly on the last step \( m \) is adjusted to \( m-1 \) or \( m+1 \) so that the search interval is finally reduced to a width of 2 rather than 1 as in the "standard" form of the algorithm. Knuth’s original algorithm uses a multiple exit loop with the main loop controlled by the key comparison; an added complication is a possible out of bounds table reference requiring the setting of a sentinel in table \( R \) when the number of elements is even.

For ease of understanding and verification we have transformed Knuth’s algorithm into a structured Pascal program (Figure 4), which in fact amounts to unwinding part of the last loop iteration. As the out of bounds index reference will occur only in this last iteration it can be avoided by guarding the table access which comes after the loop. Whereas Knuth’s algorithm computes both \( f/2 \) and \( f \div 2 \) in Pascal the latter value is obtained from the former by an adjustment in the variable “one” (of value 1 or 0). The initial assumption \( b \leq t \), though not essential, has enabled some simplification of the assertions.

It is also possible to express uniform binary search in a form analogous to the basic schemes by excluding the separate equality test from the loop, as in Figure 5.

Verification of uniform binary search

The loop invariants, IU1 and IU2, for the programs in Figures 4 and 5 follow. All of the conjuncts or terms are necessary for the correctness proofs and again it is useful to regard the second and third as the principle terms.

IU1 : ordered(\( R, b, t \)) & \( (m+f+1 < t \Rightarrow x < R[m+f+1]) \) & \( (m-f-1 > b \Rightarrow x > R[m-f-1]) \) & \( (f > 0) \) & \( (\text{found} \Rightarrow x = R[m]) \) & \( \sim \text{odd}(m+t+f) \)

IU2 : ordered(\( R, b, t \)) & \( (m+f+1 < t \Rightarrow x < R[m+f+1]) \) & \( (m-f-1 > b \Rightarrow x > R[m-f-1]) \) & \( (m+f \leq t) \) & \( (m-f \geq b-1 + \text{odd} n) \) & \( (f > 0) \) & \( \sim \text{odd}(m+t+f) \)

With reference to the program in Figure 4, on termination \( f = 0 \) or found, and, in general, excepting boundary cases, with not found we have

\[ R[m-1] < x < R[m+1] \]

Thus, as \( R \) is ordered, the search will be completed after one further comparison of \( x \) with \( R[m] \), as expected. Note that the second bounds condition in the invariant, \( m-\geq b-1+\text{odd} n \), includes specific reference to whether or not the number of table elements is odd; in fact, on loop exit, \( f = 0 \) and \( m \geq b-1+\text{odd} n \), so that with an even number of table elements (\( \text{odd} n = 0 \)), \( m \) may go out of table index bounds. The last term in the invariant expresses a further subtle characteristic of the final value of \( m \) (with \( f = 0 \)), namely, that \( m \) is one of \( t, t-2, \ldots \).

For the program in Figure 5, the condition on loop exit is

\[ R[m-1] < x < R[m+1] \]

\{b\leq t\}
\( f := t-\text{b}+1; \text{odd} n := \text{ord(odd(f))}; \)
\( f := f \) div 2; \( m := b + f + \text{odd} n - 1; \)
found := false;
while not found and (f \( > 0 \)) do
\{assertion IU1\}
begin
if \( x = R[m] \) then found := true
else begin
one := ord(odd(f)); \( f := f \) div 2;
if \( x < R[m] \) then
\( m := m-f+\text{one} \)
else
\( m := m+f+\text{one} \)
end;
end;
if not found and (m \( = b \)) then found := \( x = R[m] \);

Figure 4. Uniform binary search with equality test.
and hence the two appropriately guarded key comparisons after loop exit. In verifying these programs we assume the following axioms for integer \( N \)

\[
N > 0 \Rightarrow (N/2 + N/2 + \text{ord}(\text{odd}(N)) = N) \\
N > 0 \Rightarrow [(\text{ord}(\text{odd}(N)) = 1) \lor (\text{ord}(\text{odd}(N)) = 0)] \\
N > 0 \Rightarrow \text{ord}(M+N) = \text{ord}(N) = \text{ord}(M+\text{ord}(\text{odd}(N))).
\]

On this basis and with the initial assertion, ordered\((R,b,t)\), it is trivial to establish IU1 prior to loop entry. Below we give the verification conditions to be proven in order to establish the invariance of IU1 over the loop in Figure 4. Justification for accepting each term in the conclusion is given in braces on the right; in the logical expressions, \( \ldots \) stands for an antecedent in an implication, the antecedent being irrelevant because the consequence is shown to hold directly; \( \text{odd} \) stands for \( \text{ord}(\text{odd}(f)) \). As \( f \) is reduced over an execution of the loop, it is obvious that the loop must terminate. Verification of the program in Figure 5 is similar.

for path \( \text{"x<R[m]"} \)

assumptions: IU1 \& \( f > 0 \) \& ~found & \( x < R[m] \)

conclusions: ordered\((R,b,t)\) \{ IU1 \}

\[
& \ldots \Rightarrow x < R[m-f/2-\text{odd}(f)/2+1] \{ IU1, x < R[m], \text{axiom} \} \\
& \ldots \Rightarrow x > R[m-f/2-\text{odd}(f)/2-1] \{ IU1, \text{axiom} \} \\
& m-f/2-\text{odd}(f)/2 < t \{ IU1, f > 0, \text{axiom} \} \\
& m-f/2-\text{odd}(f)/2 > b-1 + \text{odd}(t) \{ \text{axiom} \} \\
& (b < t) \Rightarrow f/2 > 0 \{ \text{axiom} \} \\
& \text{found} => x = R[m] \{ \text{found} \} \\
& \sim \text{odd}(m-f/2-\text{odd}(f)+t+f/2) \{ IU1, f > 0, \text{axiom} \}
\]

for path \( \text{"x>R[m]"} \)

assumptions: IU1 \& \( f > 0 \) \& ~found & \( x > R[m] \)

conclusions: ordered\((R,b,t)\) \{ IU1 \}

\[
& \ldots \Rightarrow x < R[m+f/2+\text{odd}(f)/2+1] \{ IU1, \text{axiom} \} \\
& \ldots \Rightarrow x > R[m+f/2+\text{odd}(f)/2-1] \{ IU1, x > R[m], \text{axiom} \} \\
& m+f/2+\text{odd}(f)/2 < t \{ IU1, \text{axiom} \} \\
& m+f/2+\text{odd}(f)/2 > b-1 + \text{odd}(t) \{ \text{axiom} \} \\
& (b < t) \Rightarrow f/2 > 0 \{ \text{axiom} \} \\
& \text{found} => x = R[m] \{ \text{found} \} \\
& \sim \text{odd}(m+f/2+\text{odd}(f)+t+f/2) \{ IU1, f > 0, \text{axiom} \}
\]

for path \( \text{"x=R[m]"} \)

assumptions: IU1 \& \( f > 0 \) \& ~found & \( x = R[m] \)

conclusions: IU1 \{ \text{from \( x = R[m] \)} \}

\[
\text{replacing found}
\]

5. DISCUSSION
Comparison of programs and proofs

The need for care over detail is characteristic of programming and the binary search algorithms are no exception in this regard. The details of uniform binary search, despite the simple underlying principle, are exceptionally hard to understand and verify by visual inspection and tracing. This difficulty is not paralleled in the correctness proof presented in outline above, which, though relatively longer than for more standard versions, is nevertheless, straightforward. The loop invariant is similar in form but contains an extra term, and there are two differences in the proof of the invariance of these assertions for uniform binary search; firstly variables in all terms of the invariant are subjected to change along both flow paths in the loop, and secondly the table ordering is required at this stage of the proof.

In standard binary search the two principal components of the invariant reflect in an obvious way, what the algorithm or program does. On the other hand, in uniform binary search we have found that the principle components of the loop invariant (for example \( \text{"x < R[m+f/2+1]"} \) do not precisely correspond in form to the program (for the examples above, \( \text{"x < R[m+f/2+1]"} \) in which \( "\text{one}" \) may be 0 or 1). It is suggested that this slight mismatch may be related to difficulties in intuitive understanding of details of the program.

Overview of binary search

The principle components of the loop invariants of binary search loops can be regarded as reflecting the state of the search both for intermediate and final states. For standard forms of the algorithm we can think in terms of a simplified invariant, such as the following which is applicable to scheme (2)

\[
R[i] \leq x < R[j]
\]

with \( i = j-1 \) on loop termination; for uniform binary search (Figure 5) we have

\[
R[m-f-1] \leq x < R[m+f+1]
\]

with \( f=0 \) on termination. Hence in the former case one extra table access after loop exit finalises the search, and in the latter case two. When the search loop includes a test for equality, the informal invariants have \( \leq \) replaced by \( < \) and there are consequent changes to search finalisation.

While this informal invariant gives us an overview of what the programs do, to understand fully the detailed form and behaviour of a particular program we need to look at the formally stated and rigorously established logical assertions given earlier. In these assertions the relations between \( x \) and the table elements are embedded in logical implications which reflect whether or not the branches of the conditional statement inside the loop have been executed; the informal invariant, in fact, only applies after both branches have been executed.

Program details in basic schemes

One of the particular troublespots of binary search programs lies in the initialisation of the boundary indices \(i\) and \(j\) in scheme (2) and scheme (3) programs. The initialisation in our basic schemes is arranged so that the search will cover the whole table without making any assumptions about the search key with respect to the table. The value of looking at the formal statement of the loop invariants is that the preconditions of the implications in these assertions involve the initial values specifically. Often published programs based on scheme (2) do not conform in this regard. For example, Dijkstra (1978) demonstrates how to develop a scheme (2) program from a simplified invariant when \(i\) and \(j\) are initialised to \(b\) and \(t\), not \(b-1\) and \(t+1\) as in Figure 1. This gives a correct program because of an initial assumption, \(R[b] \leq x < R[t]\), which would require two table accesses or sentinel setting to establish. Failure to understand the significance of the initial setting of \(i\) and \(j\) is suggested by scheme (2) programs given in Alagic and Arbib (1978) and Kernighan and Plauger (1978); the latter present a modified scheme (2) program (comparable with (c) in Figure 3) which uses four table accesses which would be unnecessary if \(i\) and \(j\) were initialised as in Figure 1. There is no indication of similar misunderstandings with respect to scheme (1) programs, in which evidently the correct initial setting of \(i\) and \(j\) is in accord with intuitive thinking.

If the insertion position of \(x\) in the table is not required, in fact, there is some scope for varying the initial setting of either \(i\) or \(j\), but not both, in a particular program, depending on the relational operator in the key comparison of the search loop. For example, for program (2) in Figure 1, \(i\) could be preset to \(b\), and the last state­ment simplified to

\[
\text{found} := x = R[m] 
\]

in these circumstances. Looking at the associated changes in the invariant for (2) we have

\[
j < t+1 \Rightarrow x < R[j] \quad \text{&} \quad j > b \Rightarrow x > R[i]
\]

On loop termination, if \(i = b\) then \(x < R[b+1]\), which allows search finalisation as above. A similar situation arises if \(R[b]\) is a sentinel. Note that the upper bound initialisation cannot be adapted in this way for this particular example.

Thus with less general input/output specifications, programs may work with detail different from that in our basic schemes. However, such programs are special cases, which are not adaptable in the way of programs strictly conforming to the basic search schemes, for example, the algorithm in (Wegbreit, 1977) will not give correct results if \(<\) is changed to \(\leq\) in the main comparison.

Relative efficiency

The table elements probed in a binary search can be regarded as forming a decision tree (Knuth, 1973), which contains no gaps if \(N\), the number of table elements, is of the form \(2^L - 1\), where \(L\) is the number of levels in the tree. All of the given algorithms work for any \(N\); let \(L = \log_2 N + J + 1\) and then \(M\), the number of elements on the last level of the decision tree, is \(N - 2^L - 1\). For the basic schemes, the maximum number of table accesses is \(L + 1\) and the average \(L + 2M/N\) for both successful and unsuccessful search. For programs with specific equality tests within the loop statement (Figures 2 and 3), on search failure, loop execution proceeds as in the basic schemes resulting in a total of \(L\) table accesses (\(L + 1\) for (b)), but on success, there is approximately one less table access on average for moderately large \(N\) (Flores and Madpis, 1971). Note that this is counting the two array references in the programs of Figures 2 and 3 as one table access. However, the loop statements and the loop control tests in these programs involve more processing than the simpler basic schemes of Figure 1, so that in many situations the basic scheme programs will be more efficient than the programs which involve one less table access.

In uniform binary search the bottom level of the decision tree is probed (outside of the loop in our programs) on all unsuccessful paths irrespective of whether the node in the last level is occupied. In the form without equality test in the loop, \(L + 1\) table accesses are involved for \(2^L - 1 < N < 2^{L+1} - 1\). However, Knuth has shown that uniform binary search can be coded very efficiently in MIX when the adjustments to \(m\) are stored in a table.

Identification of basic schemes

A formal, analytic approach to specifications and verification of binary search in its many and varied forms has led to the identification of several underlying basic search schemes. The programs in Figure 1 are schemes with respect to the relational operator used to compare \(x\) with a table entry. A particular scheme is characterised by the interdependent termination condition and adjustments to search space boundary indices, \(i\) and \(j\); the exact nature of this interdependence becomes obvious in the process of formally verifying the programs. The initial values of the indices determine the table segment covered by the search.

The basic schemes do not depend on any input constraints other than the appropriate table ordering (and the non-zero lower bound for scheme (3) programs when \(m\) is computed as the integer quotient truncated towards zero). The position of \(x\), if it were to be inserted in the table, is pinpointed. The algorithms handle empty tables, and they handle tables containing elements with equal keys in a manner (see above) consistent with the relational operator used in the comparison of \(x\) with the table entry.

The programs of section 3 retain many of the features of the basic schemes, but it should be noted that, in contrast to the basic schemes, they do not select from a group of equal table elements in a consistent way.

Basic scheme programs, such as those given in Figure 1, are simple, efficient, general and adaptable with respect to ordering relations; moreover, for searching without insertion, they can be simplified still further as indicated above. Though readily modified along the lines of Figures 2 and 3, the additional complication needs to be justified and supported by experimental timing of execution. Knowledge of these schemes and their formal specifications enables understanding of all details of binary search programs.

ACKNOWLEDGEMENT

The author would like to thank the referees for their helpful comments on an earlier version of this paper.

REFERENCES

Binary Search


BIOGRAFICAL NOTE

Barbara Kidman was awarded a B.Sc. degree with First Class Honours in Physics from the University of Adelaide in 1948. Over the period 1949-57 she was engaged in research in biophysics, first at the University of Oxford (1949-51) and then at the University of Adelaide. She obtained a Ph.D. degree from the University of Adelaide in 1955. From 1966 to 1970 she worked as a computer programmer at the University of Adelaide. She was appointed lecturer in Computing Science in 1971 and later became senior lecturer. Her research interests are the theory and practice of programming, programming languages and Computer Science education.
Some trends in System Design Methodologies

I.T. Hawryszkiewycz*

Criteria for effective design methodologies are defined and some design methodologies compared against these criteria. The criteria call both for a disciplined set of integrated design stages as well as for techniques that link program and data base design in a semantically consistent way thus reducing program complexity.

INTRODUCTION

There has been considerable effort in the last few years to improve the system development process and ensure that new systems meet user requirements within agreed time and cost estimates. Such efforts are expected to continue given greater investment in new and increasingly more complex systems together with the increasing ratio of software development to hardware cost. General consensus is that many problems in systems development are caused by:

- the difficulty of correctly capturing all user requirements in clear and unambiguous ways,
- the lack of standards in reducing user requirements to a system design, and
- unclear project control and administration.

Consequently improvements in system development usually concentrate on these three areas. This paper deals with the second of these areas; it considers the role of design methodologies in standardising design processes. To do this, design methodologies provide a set of design steps through which a designer can proceed. Standard rules are made available for conversion from one step to the next to eliminate problems that often arise with ad-hoc conversions. The state of development of design methodologies is still in its infancy with a number of methodologies proposed but none yet proven complete or superior in any way. Perhaps one reason for this is that the principles of the design process and information systems structure are not yet developed; hence theory to evaluate methodologies does not yet exist. It is then not generally clear what methodology is best suited for a given problem; for example, using an Entity-Relationship based methodology with a given problem may yield a different design than say a reduction methodology with the same problem.

To evaluate methodologies in this context requires an understanding of the relationship between semantic structures both at the user and at the computer system level and the effect of imposing the design methodology semantics between these two levels. Little theory or few design guidelines exist in this area. Some problems that can result from such semantic relationships are outlined in this paper and shortcomings of existing methodologies discussed.

Such shortcomings indicate possible future trends. One possibility of avoiding semantic problems is to develop computer systems that are adaptable to a variety of user semantics; the use of conceptual schema in three level data base architectures is one example of this trend. By adapting the semantics at the computer system level to that of the user level eliminates the need for any intermediate methodology semantics.

Apart from such fundamental semantic issues, design methodologies must also fit within the general system development environment. To do this they should be divisible into well defined steps that form a controlled development cycle. Design methodologies must then satisfy criteria determined by fundamental semantic considerations as well as those imposed by the environment in which design takes place. Let us commence with a general framework that describes the latter.

THE GENERAL FRAMEWORK

A design methodology is here considered to be one of three components that make up system development; these components are:

- the system development cycle, which defines the reporting stages through which a project proceeds. The tasks at each stage together with their inputs and outputs are defined and the documentation at each stage is specified; this documentation is subsequently used in reviews that precede management approval to commence following stages.
- a project management system to monitor the progress of a project and take corrective action whenever problems arise. The project management system is closely integrated with the system development cycle as it uses the reports produced at each stage of system development.

"Copyright © 1981, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted; provided that ACJ's copyright notice is given and that reference is made to the publication, its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society."

*The author is with the School of Information Sciences, Canberra CAE, Belconnen, ACT 2616. Manuscript received 31 October 1980 and revised 23 December 1980.

Design Methodologies

Table 1. Criteria to evaluate Design Methodologies

| 1. Specifications at all levels should be complete both in |
| - the level of detail, and |
| - the semantic power necessary to express the design. |
| 2. A formal notation should be available at each level to |
| clearly and unambiguously specify the design at the level. |
| 3. The formal notation at the level should be clearly |
| understood by the designers and/or users involved at that level. |
| 4. There should be continuity between levels providing |
| the designer the ability to formally convert one level |
| to the next. |
| 5. The design should use principles based on sound |
| theory to provide clear and acceptable criteria to make |
| design choices. |
| 6. Both events and data structure must appear in the |
| specification. |
| 7. A linkage must be maintained between the events and |
| data structure throughout the design process. |

- a design methodology that encompasses the techniques used to design the system. The goal of such methodologies is to integrate the design techniques into a rigorous software engineering process that reduces the user specification to a computer-based information system through a number of design levels.

The general trend is to integrate these three components both vertically and horizontally; vertically by integrating all the tasks within each component and horizontally by integrating tasks between components.

One way to achieve such integration is to commence with the design methodology by grouping the design techniques into levels that mark off some well-defined design objective. These levels are then adapted to the stages of the system development cycle and subsequently to a project management system.

CRITERIA FOR EVALUATING DESIGN METHODOLOGIES

The effectiveness of a design methodology in serving this role can be judged in terms of criteria consistent with the general framework.

In the first instance, a design process must possess the necessary administrative support functions to link it to the SDLC or project management function that make up the environment. This will to some extent depend on the organisation that is using the design methodology. Such organisations usually have a set of established procedures that must be followed in a development process and the design methodology must conform with them.

Design methodologies must also, however, satisfy a second set of criteria — those that measure their cohesiveness in integrating a set of techniques into an ordered and effective design process; criteria for this purpose are suggested in Table 1.

The first four of these criteria are general and applicable to any design process, computer or otherwise. So is the fifth, namely an underlying theory. In terms of theory, computer system design is relatively immature when compared to engineering design processes, which are founded on well understood principles such as those aerodynamics, mechanics or circuit theory; these provide well-founded criteria on which alternative design choices can be made.

The last two criteria in Table 1 are particularly relevant to computing systems design, where it is necessary to represent both the events and the structure of a given information system. Apart from modelling both the events and the data structure the seventh criteria calls for the design methodology to create data structures which allow events to be modelled by programs that
- are not unnecessarily complex, and
- satisfy processing time and response requirements during execution.

DESIGN METHODOLOGIES

Three terms are used here to describe a design methodology — process, technique and specification level. The design methodology is seen as a process made up of any number of specification levels. Each specification level uses some technique to elaborate the design, and some language is used at each level to formally describe this design; the language is part of the techniques used at that level.

The concept of specification level and language is applicable to any design process, formal or informal. Most early computer system design processes consisted of only two levels, namely — the user level where the user problem is described in terms natural to the user; and — the computer level, which requires the user problem to be expressed, using the language of some particular system.

Experience has found design processes composed of these two levels to be wanting; translation from one level to the other is complex with consequent introduction of errors. Hence, as in engineering design, the trend in computer system design has been to introduce additional levels of specification to convert user specifications to a system design in incremental and more controllable steps.

It is, however, not clear what the optimum set of levels is to be nor what techniques are to be used at each specification level. On the one hand the goal must be to reduce the number of specification levels as each level requires both documentation and learning the technique and associated language thus adding to the workload; on the other hand a sufficient number of levels must be provided to ensure an ordered reduction from user requirements to the system. As a result there is a large variety of design stages. Atkinson, for example, reports a survey that shows ranges of 4 to 11 for reporting stages and 40 to 216 for activities in the system development process. Perhaps one reason for such variation is the dependence of the design process on the chosen techniques as some techniques require more steps than others to produce a design. Under these circumstances there seems to be little point in proposing an ideal set of levels and an empirical approach appears to be attractive when choosing a design methodology. One should not, however, leave the reader with the impression that any empirical set of levels are the ideal. They may suit today's state of knowledge and technology but an alternate set of levels may prove attractive given alternate technologies or design techniques; some such approaches are discussed later.

THE EMPIRICAL APPROACH

In the empirical approach, system design methodologies can in the first instance be subdivided into three categories — problem specific methodologies — those concerned with some particular problem kind

Design Methodologies

Figure 1. Entity-Relationship based methods

- computer system methodologies — those oriented towards a particular computer system
- general methodologies — these are independent of a problem kind or computer system.

Problem specific methodologies deal with well-defined problems; some examples of these are discussed by Leavenworth and Sammet\(^1\). Computer system oriented methodologies on the other hand deal with particular systems; Raver and Hubbard\(^1\) describe a method for analysis and design of IMS databases.

This paper is primarily concerned with methodologies that are independent of both the problem kind and a particular computer system. Most writers describe such methodologies in terms of four empirical design levels, namely

- data flow analysis to determine the events and the data flows associated with them
- information (or data) analysis to provide a formal model of the organisation's data structure
- logical design to develop an initial data structure and
- physical design to choose the physical access that in some way satisfy performance requirements.

Given this empirical framework and the set of criteria in Table 1, some general trends can be detected in design methodologies. For example, most methodologies now formalise data flow analysis by using graphical techniques (criteria 2), which on the most part are meaningful to both designer and user (criteria 3). Examples include data flow diagrams (De Marco\(^4\), Gane and Sarson\(^5\)) or the SADT diagrams described by Ross\(^15\). Most of these methods use top-down elaboration to capture the level of detail as required by criteria 1. Data flow analysis techniques are primarily concerned with event analysis; hence criteria 6 and 7 usually get secondary consideration; data structures in data flows, are tabulated in data dictionaries; criteria 7 is met by maintaining cross-references between events at data structures through "WHERE-USED" reports.

Reduction to the logical data level on the other hand shows more variation. It is at these steps that users require-
Design Methodologies

LOCATE-ACCOUNT
FIND ACCOUNT RECORD
DO SECTION-SEARCH

SECTION-SEARCH
DO UNTIL END-OF-SET4 = TRUE
BEGIN
FIND NEXT RECORD IN SET4.
IF NOT END-OF-SET4 THEN
BEGIN
FIND OWNER RECORD IN SET3 SET
DO SECT-TO-DIVISION-MOVE
END

SECT-TO-DIVISION-MOVE
FIND OWNER RECORD OF SET2 SET
FIND OWNER RECORD OF SET1 SET
DO PRINT

Figure 2. Finding Responsible Divisions

ments are first expressed by the semantics of the particular analysis technique; they are then converted to a logical structure with further constraints imposed by the semantics of the data model of an available dbm system. Furthermore any mismatches between the semantics of the user structure and the semantics of subsequent levels usually manifest themselves in complex programs that express event semantics in program logic, which is consistent with the data semantics imposed by the methodology and later the data model. This will be discussed later.

It is this interrelationship between user semantics, the semantics of information analysis and those of the data model and its subsequent effect on program structures that makes comparison between methodologies difficult. This is particularly so as there is both a lack of clear semantic theory of user information structures or the equivalence between these structures and various semantic and data models. Hence care must be exercised when evaluating success claims of various methodologies to ensure that such success was achieved in a general sphere rather than being a fortuitous match between the methodology semantics and the semantics of a particular problem.

Within this environment two broad analysis techniques can be identified, namely

• E-R (Entity-Relationships) based methodologies, and
• data reduction methodologies either based on
  — event reduction, or
  — data reductions.

E-R BASED METHODOLOGIES

As shown in Figure 1, a design based on E-R techniques commences by describing data by entities and relationships together with their attributes. Figure 1 shows four entity kinds — DIVISION, DEPARTMENT, SECTION and ACCOUNT. There are three relationships that show that

• one DIVISION can supervise N DEPARTMENTS
• one DEPARTMENT can supervise N SECTIONS
• ACCOUNTs are associated with SECTIONs; each ACCOUNT can be associated with many sections and each section can be associated with any number of accounts. Thus, say, a CAPITAL account can be associated with all sections as can a LABOR account. The relationships between sections and accounts contain data on actual expenditures by the sections.

This kind of information structure is typical of many flow structures through organisations as for example, job flow through a job shop, where expenditure incurred at the end of each number of different processing stages is recorded.

The conversion from the information structure to the data structure can be formalised although the conversion rules obviously depend on, and are constrained by, the database management system used. Figure 1 illustrates conversion rules to network structures; here

• each entity becomes a record type
• a1:N relationship becomes a set that contains the two entity records, and
• a M:N relationship becomes an association structure composed of a LINK record that maintains the intersection data in the M:N relationship.

In most methodologies, events at the logical level are linked to the logical access structure by access paths; however there are usually no formal rules to convert event descriptions to the access path. The formal syntax for the access path is also generally under development. A cost function is then applied to estimate performance and this is used to iteratively optimise the design. Figure 1 includes an access path for adding intersection data for a given ACCOUNT and SECTION. Two initial steps S1 and S2 find the appropriate SECTION and ACCOUNT record and then step S3 is used to STORE the intersection data. The cost function depends on the dbm system and some agreed method for estimating storage accesses for each kind of step are used in the analysis. Teorey and Fry18 and Davenport3 describe some documentation procedures that can be used to arrive at such estimates.

Viewed against the criteria of Table 1 such methodologies

• have a formal and relatively well understood formalism for representing data structure both at the information analysis and logical levels
• use linkages between the data and event structures at the logical level to develop design data that is used to optimise performance requirements rather than to develop simple program structures

Figure 3. Expanded Requirements

Design Methodologies

Figure 4. Reducing Figure 3 to a Network Structure

- lack formal conversion rules from event description at the data flow level to program code at the logical level, and
- have a set of semantics that requires users to express their requirements by entity-relationship concepts.

It is perhaps interesting to consider this last comment in light of the interaction between the semantics used in analysis, the conversion rules to logical structures and the final program produced. As an example consider the program structure to find those DIVISIONs whose SECTIONs incurred expenditures on a given ACCOUNT. The corresponding program structure is shown in skeleton form in Figure 2. Here:

- the account is located
- all SECTIONs associated with an account are then found through their association with the ACCOUNT record through sets SET-4, and
- the responsible DIVISION is then found by finding owners in sets SET-2 and SET-1.

This code looks relatively straightforward. However, consider the kind of complications that result with extension to the user structure shown in Figure 3. Now there is a more realistic picture where some SECTIONs may report directly to DIVISIONs without an intermediate DEPARTMENT. Expenditure may now also be directly incurred by DIVISIONs and DEPARTMENTs as well as SECTIONs.

Applying the earlier reductions to the E-R diagram in Figure 3 yields the logical data structure shown in Figure 4. There are now different relationship records, LINK, LINK1 and LINK2 one for each of the M:N relationship between an ACCOUNT and type organisational unit. There is also an additional set, SET-6, to represent direct reporting links from SECTIONs to DIVISIONs.

Such additional structures obviously effect programs that correspond to events. To find the DIVISIONs whose subordinates incurred expenditures on an ACCOUNT the program in Figure 2 must now be converted to that shown in Figure 5. It includes a different paragraph for each of the different relationships between ACCOUNT and a kind of organisational unit. In addition different hierarchical searches to a division must be made; these depend on whether a SECTION or DEPARTMENT is the unit that incurred an expense.

The program is now much more complex. It may then be argued that this program complexity results because of a mismatch of data structure and event semantics. The event semantics define expenses incurred by units, irrespective of type, and hierarchical reporting structures. The data structure semantics have relationships that distinguish between organisational units. Such differences require unnecessary code complexity to express event semantics in terms of data semantics. It is possible to reduce such complexity by using data semantics that more naturally express data structures. As an example, Figure 6 models the same user requirements using functional dependence (F-D) diagrams together with the role concept. For convenience only the key attributes together with functional relationships are modelled. The role concept is used to express:

- the hierarchical relationship between units; a MANAGED-UNIT and MANAGER-UNIT roles are introduced (their domain is the same as ORG-UNIT); these represent ORG-UNIT roles of being managed and managing. A functional dependency from MANAGED-UNIT to MANAGER-UNIT models the unit to which MANAGED-UNIT reports.
- the association between accounts and units is modelled by the roles USED-BY-UNIT (whose domain is the same as ACCOUNT) and ACCOUNT-USER (whose domain is the same as ORG-UNIT); a functional dependency from each combination of ACCOUNT-USER and USED-BY-UNIT models the expenditure incurred by the unit or an ACCOUNT.

In Figure 6 DIV-NO, DEPT-NO, and SECT-NO are the identifiers of the different types of organisational unit.

For reader’s convenience a tabular representation of the structure is given in Figure 7. The logical network structure, which is derived from this information model is shown in Figure 8. Here entities and relationships are again converted to record types; the richer model semantics now, however, require additional conversion rules. The types of unit are converted to record types, (DIVISION, DEPARTMENT, SECTION) as are roles (MANAGER and MANAGED). The relationship between role and role

| LOCATE-ACCOUNT. |
| FIND ACCOUNT RECORD. |
| DO SECTION-SEARCH. |
| DO DEPARTMENT-SEARCH. |
| DO DIVISION-SEARCH. |

SECTION-SEARCH.
DO UNTIL END-OF-SET-4 = TRUE
BEGIN
    FIND NEXT RECORD IN SET-4.
    IF NOT END-OF-SET-4 THEN
        BEGIN
            FIND OWNER RECORD IN SET-3 SET.
            IF SUPERIOR = "DEPT" DO SEC-TO-DEPT-MOVE.
            IF SUPERIOR = "DIV" DO SEC-TO-DIV-MOVE.
        END
    END
SEC-TO-DEPT-MOVE.
FIND OWNER RECORD OF SET-2 SET.
DO DEPT-TO-DIVISION-MOVE.
DEPT-TO-DIVISION-MOVE.
FIND OWNER RECORD OF SET-1 SET.
DO PRINT.
DEPARTMENT-SEARCH.
• similar to SECTION-SEARCH
DIVISION-SEARCH

Figure 5. Finding Responsible Division from Figure 3

Figure 6. Using Functional Dependence

DIVISIONS

<table>
<thead>
<tr>
<th>NAME</th>
<th>BUDGET</th>
<th>UNIT-NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV1</td>
<td>150</td>
<td>1</td>
</tr>
</tbody>
</table>

DEPARTMENTS

<table>
<thead>
<tr>
<th>NAME</th>
<th>PERSONNEL</th>
<th>UNIT-NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPT1</td>
<td>62</td>
<td>2</td>
</tr>
<tr>
<td>DEPT2</td>
<td>55</td>
<td>3</td>
</tr>
</tbody>
</table>

SECTIONS

<table>
<thead>
<tr>
<th>NAME</th>
<th>FUNCTION</th>
<th>TARGET</th>
<th>UNIT-NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC1</td>
<td>MACHINE</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>SEC2</td>
<td>ASSEMBLE</td>
<td>105</td>
<td>5</td>
</tr>
<tr>
<td>SEC3</td>
<td>PAINT</td>
<td>32</td>
<td>6</td>
</tr>
<tr>
<td>SEC4</td>
<td>PLAN</td>
<td>17</td>
<td>7</td>
</tr>
</tbody>
</table>

ACCOUNTS

<table>
<thead>
<tr>
<th>ACCOUNT-NAME</th>
<th>ACCOUNT-USER</th>
<th>ACCOUNT-USED-BY-UNIT</th>
<th>AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LABOR</td>
<td>2</td>
<td>LABOR</td>
<td>123</td>
</tr>
<tr>
<td>MATERIAL</td>
<td>1</td>
<td>MATERIAL</td>
<td>75</td>
</tr>
<tr>
<td>LABOR</td>
<td>1</td>
<td>LABOR</td>
<td>80</td>
</tr>
<tr>
<td>MATERIAL</td>
<td>6</td>
<td>MATERIAL</td>
<td>315</td>
</tr>
</tbody>
</table>

Figure 7. A Tabular Representation of Figure 6

player is expressed by set membership of role within role player (sets M-R and M-D). Entity types are related to their generic parent through multi-member sets (TYPE in Figure 8). Readers will note that ACCOUNT only plays one kind of role; hence no distinction is made between the entity and the role records in the logical structure. The new program, with the same function of that in Figure 5, is now shown in Figure 9. The code in paragraph FIND-EXPENDITURES expresses the semantics of association of accounts with organisational units without distinguishing between type of unit. Thus the need to search three relationships between units and accounts is replaced by one search of relationship between ACCOUNT and ORG-UNIT. Similarly different hierarchial searches are replaced by one recursive procedure FIND-DIVISION. The program remains unchanged if additional units are introduced into the organisation. In fact the program in Figure 9 applies both to the information structure shown in Figure 1 and Figure 3. Hence there is more data independence. What we have done so far is illustrate the strong relationship that exists between the semantics used in information analysis and the programs that model of events. In general, most methodologies do not consider this relationship and concentrate on data structure design given the information analysis semantics; any mismatches between data and event semantics must be subsequently implemented using program code. This is also true of reduction methods.

REDUCTION METHODS

The most common approach is to reduce data structures to 3NF relations. The same alternatives as those used in E-R reductions are also possible here; for example Figure 7 and Figure 10 illustrate two alternatives 3NF representation for the requirements in Figure 3. Again it is up to the program to convert the event semantics into semantics that match the chosen structure.

There is also a class of reduction method that commence with event descriptions and reduce these descriptions to data structures. Perhaps the most popular of these methodologies in this country is that developed by DeMarco. Here events are described by a data dictionary formalism, which is reduced to a data structure using a prescribed set of steps. However, the data dictionary formalism requires events to be expressed in terms of the data dictionary semantics, which are repeating groups. Programs must then also express the events in terms of such semantics again requiring complex translation given mismatches between event and data semantics.

For example, Figure 11 describes the information structure of Figure 3 using DeMarco's method. The inputs to the system create accounts, divisions, departments and sections and also define hierarchial relationships between them and the expenditures. There is only one output — the expenditure to an account by all units associated with a division. The specification structure is influenced by the repeating group structure semantics of most data dictionaries. Figure 11 also illustrates the results after DeMarco's reductions are applied to the output requirement. The result is four files; a program similar to that in Figure 5 would need to search these files to answer the query. However, as each file now contains the responsible division, the hierarchial search of Figure 5 is no longer necessary. This is symptomatic of most event reduction processes which tend to orient files to reports by including the majority of items required in one report in the minimum files.

Methodologies that produce one file for each report appear attractive especially where requirements are for a small number of reports as in batch systems. Their usefulness in producing suitable designs where the report requirements are large and ad-hoc is however questionable.
### Design Methodologies

<table>
<thead>
<tr>
<th>DIV-DEP</th>
<th>ACCOUNT-DEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVISION</td>
<td>DEPARTMENT</td>
</tr>
<tr>
<td>DIV1</td>
<td>DEPT1</td>
</tr>
<tr>
<td>DIV2</td>
<td>DEPT2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIV-EXPENDITURE</th>
<th>DEPT-EXPENDITURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV + ACCOUNT + DIV-EXPENDITURE</td>
<td>DEPT + ACCOUNT + DEPT-EXPENDITURE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIV-DEP</th>
<th>ACCOUNT-DEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVISION</td>
<td>DEPARTMENT</td>
</tr>
<tr>
<td>DIV1</td>
<td>DEPT1</td>
</tr>
<tr>
<td>DIV2</td>
<td>DEPT2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIV-SEC</th>
<th>ACCOUNT-SEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVISION</td>
<td>SECTION</td>
</tr>
<tr>
<td>DIV1</td>
<td>SEC1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIV-EXPENDITURE</th>
<th>SECTION-EXPENDITURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV + ACCOUNT + DIV-EXPENDITURE</td>
<td>SECTION + SECT-EXPENDITURE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEP-SEC</th>
<th>ACCOUNT-SEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPARTMENT</td>
<td>SECTION</td>
</tr>
<tr>
<td>DEPT1</td>
<td>SEC2</td>
</tr>
<tr>
<td>DEPT1</td>
<td>SEC3</td>
</tr>
<tr>
<td>DEPT2</td>
<td>SEC4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT-EXPENDITURE</th>
<th>SECTION-EXPENDITURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPT + ACCOUNT + DEPT-EXPENDITURE</td>
<td>SECTION + SECT-EXPENDITURE</td>
</tr>
</tbody>
</table>

(replaces ACCOUNT-USE in Figure 7)

Figure 10. An Alternate 3NF Reduction

### Summary of Current Methodologies

In summary design methodologies have introduced discipline in the design process in the main satisfying criteria 1 to 4 in Table 1. At the same time it may be noted that additional development is necessary to overcome some of their shortfall especially those identified by criteria 5 to 7. Such shortfalls have been here identified as:

- insufficient attention to semantic relationships between data structure and programs with resulting program complexity; this is often caused by an inflexible set of semantics used by each methodology and
- the workload involved in the conversion between levels.

### PROPOSALS FOR NEW METHODOLOGIES

Given this, it can be expected that future developments will be oriented towards:

- filling the gaps in existing methodologies
- automating some levels of design, and
- introducing new techniques to overcome the semantic constraints imposed by particular methodologies.

### AUTOMATED AIDS

Automated aids appear as an attractive alternative in the first instance. However, aids are usually extensions of manual design processes, and as such exhibit the same semantic problems.

Most aids are limited in their applicability and are restricted to some particular problem, or a particular computer system or only a subset of design level. Most available aids are related to one design level or at most have a limited scope of conversion between two adjacent specification levels. The PSL/PSA system for example concentrates mainly on user requirements; its goal is to validate requirements specifications. There are also aids that concentrate on levels other than the requirements level. Many of these are used to optimise database design given an initial structure; examples include Alter or Mitoma and Irani.

Formal techniques to convert specifications from one level to the next also exist; most generate programs for one particular processing method. Some are restricted to particular data structures. Jackson's method is one of the better known methods; it generates programs that produce reports from certain classes of data structures. Other reported examples of this class of generator includes an interactive program generator for IDMS files described by

Figure 11. Reduction Process

The conversions illustrated in Figure 12 are attractive as they lead to the possibility of automatic conversion to code. One such possibility is illustrated in Figure 13. Here a sample set of enterprise level commands is given together with standard code, using the CODASYL network proposal, to which each such command is translated. The first command is to find an entity with a given value of KEY-ITEM-1. Here standard representations are used for each enterprise concept; the standard representation for an entity kind is the record type. The FIND ENTITY command is converted to a FIND RECORD command addressed to the record type together with tests for the existence of such records. An option is allowed to find those entities that have a given value for the properties; these also convert to standard code. The second command is to find an entity related to a second entity with a given relationship. The two entity kinds are represented by record type R1 and R2; the standard representation for the relationship is two sets, S1 and S2, and the association record, LINK. The corresponding standard code traverses from record R1 (which must have earlier been found through the code that corresponds to the FIND ENTITY command) through sets S1 and S2 to R2 records. Further details of such commands are given in Reference 6. Mackenzie[12] also describes a similar system, which translates programs using a relational model to programs using a CODASYL-like network model.

These developments however do not solve the problems caused by the semantic mismatches, which were described earlier. To overcome such problems requires a
general semantic method with applicability to a wide set of user problems. Data models are not usually sufficient for this purpose; most debates on data models conclude that any single data model lacks the richness to adequately model large classes of user requirements. Perhaps rather than relying on one data model the goal should be to develop design methodologies that permit semantic flexibility at the user level. These methodologies in turn require computing systems that can be adapted to the semantics chosen by the user.

**ADAPTABLE SYSTEMS**

The general structure of an adaptable system is shown in Figure 14. Here the user chooses the semantic terms to express the user requirements. The database administrator then defines an interface that allows user requirements to be expressed in the same semantic terms. Thus, suppose E-R concepts are chosen to express the problem; the dba then develops an interface that includes commands that allow entities and relationships to be defined as the user data model and operations expressed using syntax consistent with E-R concepts.

To do this a high level definition facilities must be provided to the dba; these include a
- a language to define a semantic or data model, and
- a language to define basic operations on this model.

An interpreter uses these definitions to transform an internal structure consistently with the basic operations initiated by users. Readers may note the resemblance between this approach and the ANSI/SPARC architecture; the semantic level in Figure 14 being similar to the conceptual level in this architecture.

Given these definition facilities the dba would define operations such as

**OPERATION: DEFINE ENTITY KIND** (definition in terms of definition language specifying that the user nominate a parameter ENTITY NAME and any PROPERTIES names).

**OPERATION: ADD ENTITY** (definition in terms of the definition language specifying that user nominate actual property value).

The user would then execute these operations as follows: **DEFINE ENTITY KIND** (ENTITY-NAME: DIVISION, PROPERTIES; (DIV-NO, BUDGET)); **ADD ENTITY** (ENTITY-NAME: DIVISION, PROPERTIES: (DIV-NO: DIV1, BUDGET: 150)); The feasibility of the adaptable system rests on the availability of a set of concepts that can be used to describe arbitrary semantic or data models. The internal level in Figure 14 is a representation of these concepts; the data model definition and operation definition are then defined as transformations in terms of these concepts. This feasibility has been proven by a system that uses two concepts. A structure abstraction is used to represent data model structures such as RELATION, RECORD and so on and occurrence abstractions to represent occurrences of these structures. The internal level implements these abstractions by trees with specialised nodes. Data model operations are then defined in terms of tree transformations using a special language; details are given in Reference 7.

**CONCLUSION**

A set of criteria for evaluating design methodologies was defined and some shortfalls of existing methodologies when compared to these criteria were identified. In particular the interaction of user methodologies and data model semantics can sometimes have undesirable effects on the final designs especially program structures. It was proposed that an adaptable system, which provides an interface in terms of user semantics, be developed to minimise such effects.

**REFERENCES**

15. ROSS, D.T., SCHOMAN, K.E. Jr., "Structural Analysis for Requirements Definition", in *Software Design Techniques*


BIOGRAPHICAL NOTE
Igor Hawryszkiewycz received his B.E. and M.E. degrees in Electrical Engineering from the University of Adelaide and his Ph.D. in Computer Science from the Massachusetts Institute of Technology in 1973. He first worked at the Research Laboratories and subsequently the Data Processing Branch, of the PMG Department (now Telecom), researching computer networks and database systems. Currently he is Principal Lecturer for Information Systems at the Canberra College of Advanced Education. His interests are in systems design, database systems and distributed systems.
Short Communications

NEBALL and FINGRP: NEW PROGRAMS FOR MULTIPLE NEAREST-NEIGHBOUR ANALYSES

D.J. Abel and W.T. Williams*

*D.J. Abel is with CSIRO Division of Computing Research, Townsville, Qld 4814. W.T. Williams is with the Australian Institute of Marine Science, Townsville, Qld 4810. Manuscript received 8 August 1980 and revised 23 September 1980.

NEBALL assumes the prior existence, for n elements (n < 150), of an inter-element distance matrix of order n. Each element is then associated with the complete ordered set of its (n - 1) nearest neighbours. For every pair of elements the number of matches in successive subsets of r (r = 1 to n) is compared with random expectation, and a measure of discrepancy, $\Delta_r$, is written away. It is shown that $\Delta_r$ is constrained within the limits ± 0.25. Program FINGRP accepts the matrix of $\Delta_r$ values and submits it to a non-hierarchical classificatory procedure at externally-specified levels of similarity. The result is a non-exclusive set of homogeneous groups of elements.

Keywords: Multiple nearest neighbours, minimum spanning tree, classification, ordination.

CR categories: 3.12, 5.19, 5.32, 5.4.

INTRODUCTION

It is well known that classical ordination techniques, such as principal component analysis and its analogues, encounter difficulties if the data-set under study contains maked discontinuities or non-linear relations. As a result biologists have increasingly turned to graph-theoretic approaches for the study of variation in complex systems. The earliest of these was the minimum spanning tree (Prim 1957). This uses only the first nearest neighbour of each element; if the system contains an intricate network of cross-relations it is apt to produce a richly-branched tree, with many peripheral elements, that is difficult to interpret. Improved results have recently been obtained by computing a network based on the first two nearest neighbours (Williams, Burt, Pengelly and Robinson 1980; Williams 1980), but this too may prove unsatisfactory for very complex systems. In one application of this technique (Clifford and Williams 1980) a puzzling situation was resolved by appeal to a list of the first seven neighbours. This suggests an obvious extension: to devise algorithms which use, for each of n elements, the complete ordered set of (n - 1) neighbours; each element would then be examined in the context of every other element in the system. In this communication we first describe a new program NEBALL which implements such a procedure.

THEORETICAL

The ordered list of neighbours for each element will be a permutation of the integers 1 to n (each element being regarded as its own first nearest neighbour). Consider now two such lists and let us for the moment denote by $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$. For each integer r in $1 \leq r \leq n$, we count the number of elements that the two sublists $(a_1, \ldots, a_r)$ and $(b_1, \ldots, b_r)$ have in common and we divide this by n to obtain a quantity we shall denote by $\gamma_r$. For two identical elements we shall have, for each value of r after the first, $\gamma_r = r/n$. For comparison we now need to know the random expectation $\gamma_r$, i.e., the value of $\alpha_r$ to be expected from two randomly permuted sets.

We have adopted the formulation

$$\alpha_r = \frac{1}{n} \sum_{s=1}^{r} \frac{s}{n} \left( \frac{n-r}{s} \right) \left( \frac{r-s}{r} \right) \ldots (1)$$

It should be noted that this $\alpha_r$ is not precisely equal to the average of $\gamma_r$ for repeated sampling. (Clearly our method is comparing two lists which have differing first elements.) We have, however, determined experimentally that the difference in these two quantities is of the order $10^{-4}$ for even small n (n = 25).

This expression has a singularity at $r = n/2$. It will be obvious that computation of expression (1) involves, inter alia, the computation of $1/n(n-2r+s)$. However, if $r > n/2$ then, for small values of s, the expression inside the brackets may be negative; this is simply because, once the halfway mark is reached, there must be some matches. The appropriate procedure is to use the quantity $(n-2r+s)$ as a flag: if for any combination of r and s it becomes negative, no contribution is made to the summation.

There will be available then a vector of $\gamma_r$ values dependent only on n and, for each pair of elements, a vector of $\alpha_r$ values. We denote the quantity $(\alpha_r - \gamma_r)$ by $\Delta_r$. For comparison we have elected to use the Kolmogorov-Smirnov procedure (see, e.g., Siegel 1956, p. 128), and to write away the signed value of $\Delta_r$ for which $|\Delta_r|$ is maximum. We shall now show that the value of $\Delta_r$ is constrained between the limits ± 0.5.

For two identical elements, we have empirically determined that the maximum value of $\Delta_r$ will occur at $r = n/2$ (if n is odd it will occur at $(n-1)/2$, but this need not concern us). The numerator of expression (1) at this value then becomes:

$$\sum_{s=1}^{r} \frac{s}{n} \left( \frac{r}{s} \right)^2$$

but this

$$= r \sum_{s=1}^{r} \frac{(r-1)}{s-1} \frac{r}{s}$$

which, by appeal to the coefficient of $x^r$ in the expression of $(1 + x)/(1 + x)^{r-1}$,

$$= r \left( \frac{2r-1}{r} \right)$$

(We are indebted to Mr Russel John, of the CSIRO Division of Mathematics and Statistics, for this proof.) If we now put $n = 2r$, and substitute expression (2) in expression (1), we have $\gamma_r = 1/4$. Since at this point $\alpha_r = 1/2$, we have $\Delta_r = 1/4$. For completely unlike elements whose ordered rows are mirror-images of each other, there will be no matches up to $r = n/2$, so that at this point $\alpha_r = 0$ and $\Delta_r = -1/4$. It follows that, although $\gamma_r$ could plausibly be regarded as a cumulative probability distribution, use of the Kolmogorov-Smirnov statistic in a test of statistical significance would
be unwise, since the distribution of $\Delta_r$ is truncated above and below.

**USE OF THE STORED VALUES**

When the above procedure has been completed for all pairs of elements, the upper triangle of an $n \times n$ matrix of $\Delta_r$ values will have been stored. Two procedures are suggested for the use of this matrix. First, if all values are subtracted from 0.25, the matrix resembles a distance matrix (though there is no reason to suppose that the resulting quantities will be metrics), and a new minimum spanning tree can be constructed. In every case we have so far examined, the resulting tree, compared with its counterpart from the original distance matrix, is much less richly branched, a very large proportion of the elements being strung out along a single main axis. Intuitively, we assume this is because every element has been examined in the context of every other, so that underlying continuous variation can be more effectively expressed. However, it will usually also be desirable to establish optimum cutting-points, and for this it is necessary to submit the $\Delta_r$ matrix to a non-hierarchical classificatory procedure. For this purpose a companion program for the extraction of homogeneous groups is necessary.

**THE GROUPING PROGRAM**

We define a homogeneous group as one all of whose members are within a specified distance (i.e., with $\Delta_r$ not less than a specified 'grouping criterion') of all other members. The resulting classification will be non-exclusive, in that an element may be a member of more than one group. The program FINGRP develops such groups by exhaustive enumeration. The procedure is sequenced so that all groups including element 1 are identified first, then all groups including element 2 but not 1, and so on. For a given element, initially a set of two-element groups is formed, from the element and each of its neighbours that satisfy the grouping criterion. Larger groups are then formed, from the element and each of its neighbours that have not been combined into a single program. It is possible that further experience will allow grouping criteria to be set automatically.

**SPECIFICATIONS**

NEBALL expects the upper triangle of an $n \times n$ distance matrix on a file TAPE15 in conventional TAXON FORMAT (7(F9.5,1X)). It requires only a single control card, giving the value of $n$ in FORMAT (14); as currently dimensioned, $n$ must not exceed 150. The set of $\Delta_r$ values with their element-tags are printed out as an inter-element upper triangle, six items to a row. The distance matrix ($0.25 - \Delta_r$) is written away as a file TAPE14, for later use by MISPAN or FINGRP. The program is in FORTRAN, except for a single subroutine (SIDADD) in the Cyber assembly language COMPASS; a 60-bit word is assumed. Computing time is approximately dependent on $n^3$; on the Cyber 76 computer the time taken was 0.6 seconds for $n = 37$, 39.6 seconds for $n = 150$.

FINGRP requires two control cards: the first gives a header for the analysis, the second gives $n$, the number of grouping criteria required, and the grouping criteria themselves. The cards are formatted as (20A4) and (I3,I2,7F10.0) respectively. The maximum value of $n$ is again 150; 36000 words are allocated for the group-members lists. Computation time for the 150-element set, for four grouping criteria, was 9.1 seconds.

Print-outs of the programs can be obtained on application to the first author.

**REFERENCES**


PROGRAM INVER REVISITED

D.J. Abel and W.T. Williams*

*D.J. Abel is with CSIRO Division of Computing Research, Townsville, Qld 4814. W.T. Williams is with the Australian Institute of Marine Science, Townsville, Qld 4810. Manuscript received 8 October 1980.

The original program INVER computed a distance-matrix between attributes of mixed types. Though generally effective, it occasionally encountered difficulties, largely as a result of unusual data-configurations which had not been foreseen. These have been identified, and a new version written which appears to overcome the problems.

Keywords: classification, attributes, mixed-data.
CR categories: 5.4.

The problem of defining a distance-matrix between attributes of mixed types long remained intractable; the first published solution appears to be that of Lance and Williams (1979) as the program INVER. This program has been successfully applied to a number of problems, notably for genetic resources data (see, e.g., Williams, Burt and Lance 1980, Robinson, Burt and Williams 1980). However, with increasing use the original program was found to possess certain weaknesses:

(i) it did not cater for all attribute-types required by users,
(ii) the safeguards against indeterminacy were insufficiently rigorous, and could fail with some unusual data-configurations, and
(iii) the algorithm for transforming a product-moment correlation coefficient into a distance needed re-examination.

The senior author has now written a new version of INVER which incorporates the following changes:

(i) Provision has now been made for "non-exclusive nominal multistates", which were not handled by the original program. The facility for asymmetric binaries has, however, been removed; it has not been a requirement of any user we have encountered, and it complicates the control cards.

(ii) The sequence of computation has been altered to check more thoroughly for indeterminacy. Range-chopping of numeric attributes, when necessary, is now carried out immediately before each relevant comparison, instead of immediately after input and masking. As before, for each pair of attributes, only individuals with both attributes non-missing are considered; the new version facilitates recognition of the possibility that, of the included individuals, one or both has all values identical or missing. This possibility was not envisaged in the original program, but is has occurred in practice; when it does occur the distance-measure is set to 1.0 (i.e., attributes totally unrelated). Strictly, that distance is a "missing value"; but since graph-theory programs using distance-matrices do not normally cater for missing values, we have thought this arbitrary decision unavoidable. Difficulties which can arise from contingency tables with all-zero rows or columns are now overcome by collapsing such tables before computation.

(iii) For transformation of a correlation coefficient $r$ into a distance measure $d$ the original program examined three possibilities:

$$d_1 = \frac{1}{2}(1 - r); \quad d_2 = 1 - |r|; \quad d_3 = 1 - r^2$$

Practical tests suggested that $d_2$ was unacceptable, and the authors decided to use $d_1$. This evoked some correspondence between Drs A.J. Swain and W.T. Williams in the columns of this Journal (Australian Computer Journal, Vol. 11, No. 3, pp. 113-4) concerning the validity of this decision; and the problem evidently required reconsideration. It has now been clarified by our observation that there are two types of user, with different requirements. User A wishes to use the results predictively, and from his point of view a perfect negative correlation is as effective as a perfect positive one; he expects two attributes which are perfectly negatively correlated to be coincident on a minimum spanning tree computed from the distance matrix. For this user, maximum distance is represented by zero correlation. User B regards negatively-correlated attributes as being as dissimilar as possible, and expects them to be widely separated on the tree. For user A, $d_2$ is the appropriate choice; for user B it is $d_1$. It is for the user to decide which attribute he wishes to take; the program provides both transformations as options.

The external features of the program are substantially unchanged. Values for multistate and binary attributes are checked on input for validity; when all data have been read, attributes which are missing for all individuals, or which are identical in non-missing values, are masked out. Apart from deletion of references to asymmetric binaries, the conventions for control and data cards are unchanged. The program has been extensively tested against hand-computation, and is believed to be more reliable than its original counterpart. It is proposed to offer the program for inclusion in the TAXON library of the CSIRO Cyber 76 computer; meanwhile, a copy can be obtained from the senior author.

REFERENCES


A COMPARISON BETWEEN PASCAL, FORTRAN AND PL/1

D.J. Kewley*

*The author is with the Defence Research Centre, Salisbury, SA. Manuscript received 7 October 1980.

The PASCAL language implementation on the Defence Research Centre Salisbury IBM3033 is tested for program CPU execution times and compared with similar programs in FORTRAN and PL/1. The results show that the relative execution times are similar to those measured on CDC and UNIVAC machines and that the PASCAL language programs were usually slower than optimised FORTRAN while being faster than PL/1 for some of the tests.

Keywords and phrases: PASCAL, execution times, compiler evaluation, scientific performance.

CR categories: 3.1, 3.2, 4.6.

INTRODUCTION

This paper reports on a comparative study of the execution times of computer programs written in PASCAL, FORTRAN IV and PL/1 on an IBM3033 computer. The work was carried out in order to assess the quantitative cost of scientific computations when using a modern high level language such as PASCAL, which has easily understood structured programming controls and extensive data structures, and to determine any significant time-expensive features. The comparison is limited to allow obvious and easy translation of algorithms between the languages and therefore does not fully use the power of PASCAL. Three of the five test programs have been used in similar comparisons on a CDC 6400 (Wirth, 1971) and a UNIVAC 1100 (Ball, 1978), while two programs are added to be representative of scientific computations.

TEST PROGRAMS

The test programs consist of three examples from Wirth's (1971) original comparison, the GAMM measure of scientific computer performance (Wichman and DuCroz, 1979) and a Fast Fourier Transform (FFT) algorithm.

In detail, these are:

1. MATMUL, matrix multiplication B := A*A, no handling. This tests real multiplication and 2D array handling for a 100 x 100 matrix.

2. SORT, sorting an array of 1000 numbers, no output. This tests integer arithmetic, logical testing and 1D array handling.

3. PART and PARTNP. Finding all possible additive partitions of integers 1-30. This tests recursion, while using a 'hand-coded' stack in FORTRAN, and formatted outputs. PARTNP has no output.

4. GAMM, the measure program uses 13 loops made up of 2 additions, 1 multiplication, 3 polynomials, 3 maxima and 4 square-root loops. This tests the performance of the language for scientific computations.

5. FFT, the simple program given in Rabiner and Gold (1975) has been converted to real arithmetic and gives virtually identical program statements for each language. This tests the use of standard functions and is another test of the scientific performance. Note that the algorithm is not the fastest possible.

Listings of each of these programs are available from the author.

RESULTS

The test programs were all executed in batch mode to ensure system overheads were separated from the actual CPU execution times. The compilation of the programs were made using PASCAL (Version 1.2) (Cox and Tobias, 1978), with and without run time checks, FORTRAN G (Version 2.0), FORTRAN H (Version 2.3.0) opt(2) and PL/1 optimising compiler (Version 1.3.0) compilers. The compilation times for FORTRAN G and PASCAL were similar while the times for the FORTRAN H and the PL/1 compilers were two and four times longer, respectively. The quickest compilation was 0.12 secs.

The relative execution times are presented in Table 1. For comparison purposes the FORTRAN G compiler times were taken as a base. This enables the results of Wirth (1971) and Ball (1978) also to be tabulated. The actual times are given for the unoptimised FORTRAN IV compilers.

Inspection of the table shows that the three machines' compilers give similar performance although the languages on any particular machine show considerable variation. The PASCAL compiler run time checks can in some cases take 30% of the time. On average the FORTRAN H compiler gives the fastest code except for PART when there is substantial output required. In this case PASCAL is seen to be faster than FORTRAN and PL/1. The two scientific measure programs give similar results which show PASCAL execution times nearly twice as long as the optimised FORTRAN times.

The poorest comparison example for PASCAL is the matrix multiplication program. There were a number of possibilities for this poor performance and so variations on the PASCAL program were tested. These are summarised in Table 2. It is clear from the table that the major reason for the slower times was due to the calculation of the variable array indices.

The FFT and GAMM tests both use on-dimensional

Table 1. Comparison of CPU Execution Times of Test Programs on Three Machines

<table>
<thead>
<tr>
<th></th>
<th>MATMUL</th>
<th>SORT</th>
<th>PART</th>
<th>PARTNP</th>
<th>FFT</th>
<th>GAMM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CDC</td>
<td>U</td>
<td>IBM</td>
<td>CDC</td>
<td>U</td>
<td>IBM</td>
</tr>
<tr>
<td>PASCAL</td>
<td>2.5</td>
<td>1.8</td>
<td>2.9</td>
<td>1.2</td>
<td>1.4</td>
<td>1.5</td>
</tr>
<tr>
<td>No checks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORTRAN IV</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Time (sec)</td>
<td>19.8</td>
<td>10.3</td>
<td>1.85</td>
<td>17.1</td>
<td>18.0</td>
<td>.85</td>
</tr>
<tr>
<td>FORTRAN IV</td>
<td></td>
<td>.4</td>
<td>.7</td>
<td>.6</td>
<td>.4</td>
<td></td>
</tr>
<tr>
<td>Opt(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL/1 (opt)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

arrays and consequently have a relative time of 1.3 which is close to 1.2 given in Table 2. Wirth (1971) had recognised this problem and his compiler used optimised index computations where possible, e.g. for MATMUL with n=96 the relative time was reduced from 2.5 at n=100 to 1.6.

It would appear that the UNIVAC implementation has a better index calculation method than the current IBM3033 PASCAL compiler. A time-saving method often used on older FORTRAN compilers is to reduce the 2D arrays to 1D and explicitly calculate the index in the program. This did not lead to any time reduction when applied to the PASCAL program.

CONCLUSIONS

A comparison of the PASCAL, FORTRAN and PL/1 programming languages on the DRCS IBM3033 showed that, on average, code produced by the FORTRAN H compiler produced faster execution times than PASCAL and PL/1. For most tests the PASCAL times were similar to those of the FORTRAN G compiler. Half the tests showed PASCAL was equal to or faster than PL/1.

The speed of PASCAL programs for scientific computations is found to be very sensitive to array index calculations and for one-dimensional arrays 1.8 times slower than optimised FORTRAN. This suggests that microprocessor implementations (Crespi-Reghizzi, Corti and Dapra', 1980, and Kewley and Marwood, 1980) of PASCAL or a derivative should ensure that these calculations are optimised.

If the user wishes to use the self-documenting, data structure and structured programming facilities of PASCAL while maintaining high execution speeds then using FORTRAN H compiled subroutines in critical areas is an option available on the present IBM3033 system.

| Table 2. Comparison of Relative CPU Execution Times for Variations of the MATMUL PASCAL Program |
|----------------------------------------------------------|-------|
| FOR, REAL, *, 2D arrays | 2.1   |
| FOR, REAL, *, 1D arrays | 1.2   |
| REPEAT, REAL, *, 2D arrays | 2.0 |
| REPEAT, INTEGER, *, 2D arrays | 1.7 |
| REPEAT, REAL, +, 2D arrays | 1.8   |
| FOR, REAL, *, 3D arrays | 3.0   |
| FOR, REAL, *, No arrays | 1.0   |
| FOR = FOR loop used |
| REPEAT = FOR loop replaced by REPEAT. UNTIL |
| REAL = REAL data |
| INTEGER = INTEGER data |
| * = multiply used |
| + = multiply replaced by addition |
| 3D = three-dimensional arrays with variable indices used |

REFERENCES


Book Reviews


This is an excellent reference manual for a large variety of practical network optimisation problems. It would be most useful as a first source of ideas for a consultant engineer or OR practitioner. The book gives many different algorithms and techniques for a number of problems in the fields of water and sewage flow, garbage collection, bus routing, traffic flow, depot location, etc. Each algorithm is well illustrated by diagrams and a simple example. A number have step by step output from a computer program written by the author. (While there is a reference to what appears to be a most useful program, no details of availability, language, etc are given.)

In my experience, few problems would exactly fit the scenario and hence the algorithm given. In most cases, I would see the methods being used as a first step in the problem solution and/or as forming the basis for a heuristic solution. In fact, the book itself frequently suggests this, and many of the techniques given are themselves heuristic.

For each type of problem and technique, the book gives a number of detailed theoretical references, both articles and surveys. These are up to date and to my mind are one of the best features of the book, although not all references appeared in the reference list. Each chapter is terminated by a number of useful exercises. Unfortunately, no answers are given.

I found the book fairly heavy going as, despite the simple examples, it is basically just one algorithm after another. In some cases the terminology is very confusing as it differs from algorithm to algorithm throughout a chapter, and in many cases, references are made to terms which are defined pages earlier. It is definitely a reference book on techniques, not a general text on networks (such as Christofides (1975) to which he refers frequently) unless used in conjunction with another text. The language is fairly colloquial (American) and hence the book is full of irritating minor grammatical errors. However, as a source of techniques, ideas and references, it would be a most useful addition to the bookshelf of anyone interested in practical network problems.


Jenny Edwards
New South Wales Institute of Technology


With rare exceptions this book ignores examining the scope and limits of CAL and suggests little useful progress has been made. The papers are, on the whole, catalogues of terminals, languages (Pascal is praised but no-one says why), biographies and acronyms devoid of critical analysis and leaving the reader wondering if CAL is really that bad.

Odd useful ideas creep in, and the papers by Howe et al, Cooper and Lower are stimulating. These authors, at least, have tried to come to grips with some of the interesting questions which CAL raises.

All in all it is a good book to use for a list of names and addresses on contacts and then to store in the closed stack of your local library.

R.P. Watkins
University of Tasmania


This book describes the use of a suite of interactive computer programmes for eliciting and manipulating the user's "repertory grid" for any area in which he may wish to study his own conceptual structure. Generally these programmes control an interaction in which the user nominates elements within the conceptual area (e.g. programming languages), the user is then asked to name bipolar constructs (conceptual dimensions such as "block structured vs. not block structured" or "I like vs. I don't like") which separate these elements and to rate each element on each construct. Finally cluster analyses are performed for the constructs and for the elements, suggesting groupings by similarity of ratings.

This structure of course represents the choices of the user and may well be largely specific to the individual. The process is based on the Personal Construct Theory of George Kelly who extended the view that scientists are less concerned with discovering reality than with inventing ways to construct it, to the view that in his psychological development each individual must behave as a personal scientist and construct his own view of reality.

As a metatheory this is not unreasonable. The problem is whether it is a productive way to construe reality, that is, whether personal constructs can be handled on a more general basis than a purely descriptive one.

There are additional problems with the repertory grid process both in the reliability and validity of the ratings and in the effect of element selection on the set of constructs elicited, and the problem of the stability and meaningfulness of the cluster analysis. The process suggested for relating one personal construct system to others whether from other individuals, or from the same user at different times or in different roles ('as I am', "as I would like to be" etc.) is to cluster analyse the combined constructs over a fixed set of elements. A process is also suggested for calculating a similarity measure between grids, and thus constructing sociograms, showing links between, and suggesting clusters of, individuals.

One obvious question is why use the repertory grid procedure, let alone a computer to administer it. Most of us could informally list and structure our constructs in an area using a pen and paper. One obvious application of the less direct, but automatic, procedure is in clinical psychology, where a person's pattern of choices may reflect attitudes (e.g. towards members of their family) which they could not state explicitly. However the strongest argument advanced for this computer administered process (other than adherence to Personal Construct theory) is that people find it meaningful and useful.

As Shaw notes (p. 148) such "... content free conversational algorithms have the capacity to encourage and control conversation ... and (p. 151) that such conversation with oneself via a computer terminal is becoming a viable alternative to confiding in the vicar.

Other applications discussed include the views of management and inspection staff of faults in garment manufacturing, the management appraisal of subordinates.

About half the book is occupied by printouts of interactions with the programmes. Programme listings are not included but assistance is offered by the author (Mildred Shaw, Middlesex Polytechnic, Herts, England). While the book raises some general issues such as the use of an interactive computer as the tool of the user rather than a machine, its value would seem to be largely confined to those with a professional interest in Personal Construct Theory.

J.R.M. Alexander
Department of Psychology
University of Tasmania

COMMENT ON JOURNAL CONTENTS

For many years I have received a great deal of pleasure from reading the titles of the papers presented in the Australian Computer Journal. It's common knowledge that their relevance to the vast bulk of data processing personnel is extremely minute. However, I could not let the title of an article in the August issue go without comment. The title of the article is as follows, “Computer Elucidation of the Occurrence of Higher Odd Sub Harmonic Motion and Other Sub Harmonic Phenomena”. This surely must rank as one of the most ridiculous paper ever to be presented by the Australian Computer Society in a magazine which over the years has certainly rated many ridiculous papers relative to the real use of data processing.

While I understand it is the role of academics to publish or perish it is unfortunate the vast majority of data processing personnel have to fund the cost of that exercise. I would suggest that an opportunity be given to computer members when paying their subscriptions as to whether or not they choose to participate in receiving the Australian Computer Journal or not. Those who decide they would be able to operate effectively without receiving the Journal should be given a rebate of say $5 to $10. Those persons choosing to receive the Journal should therefore split the cost of printing these rather abstract articles.

I find it continually distressing that an organisation that constantly calls for the need of greater awareness and support from the computer industry can continue to produce a Journal with articles of this type. They only serve to make the worthwhile Australian computer society a laughing stock of traditional hardworking, fully employed data processors.

G.E. McGuiness
Arthur Andersen & Co.,
330 Collins Street, Melbourne

EDITOR'S COMMENTS

The above letter was received on 13 November 1980. It so happens that my Editorial in the November issue had just addressed itself to the points raised by Mr McGuiness. We are certainly well aware that these sentiments he expresses are widely shared, and this is why I will use this opportunity to go over the same ground yet another time.

If Mr McGuiness believes that ceasing to support the Journal will save ACS members five or ten dollars per head, he is in for a severe disappointment. Currently it costs about $30,000 a year to publish the Journal, and against this we can offset about $10,000 of outside subscription income (over 600 at $15 each), giving a net cost of about $20,000, or $2 per member. If any member believes that the amount is important enough, perhaps he would like to raise the issue with his Branch Executive or the National Council. None of these bodies are currently controlled by academics, and the practitioner can expect his views to receive a fair hearing. I might add that, in view of the seriousness of Mr McGuiness' criticisms, it is disappointing that he did not use the opportunity to first check the publication cost figures, which are included in the annual budget statements printed in the Bulletin.

Mr McGuiness did not explain what in the paper referred to he found so objectionable, other than mentioning the title. It seems a little unfair to the authors that anyone should dismiss their work without first making some effort to know more about its contents, regardless of whether the paper is relevant to one's particular interests. No doubt academic research is not everyone's cup of tea. However, perhaps Mr McGuiness overlooked the fact that the Constitution of ACS prominently lists the advancement of computing knowledge as one of its main objectives. If Mr McGuiness believes that publication of research papers of all kinds in computing is not an activity worthy of the Society's resources, he should consider, again, raising the matter with the decision-making bodies.

I am as anxious as Mr McGuiness that the Journal should publish more material of direct interest to the great majority of its readers. If he would look over the authors' biographies in the August issue, he should note that several of them, though currently employed in academic institutions, have considerable industrial experience. (Perhaps it says something about the industry's working environment that these people chose to move away from it.) If Mr McGuiness knows of other authors willing and able to contribute technical material (not publicity or customer-services material) to the Journal without pay, I would be very pleased to hear about them.

FOLLOW-UP LETTER

Thank you very much for your prompt reply to my letter and I have noted the contents of your proposed reply with great interest.

It is unfortunate that you feel the need to defend a previous publication with such enthusiasm regardless of the merit of any criticism. Due to the emotional nature of your editorial reply, I must insist that if your proposed editorial comment is printed in its present form then this reply must also be considered for publication at the same time.

Firstly you have missed the major intent of my original letter and have chosen to comment on points which are immaterial to the major thesis. For example, your comments relative to the cost per member for the journal were interesting however the major point is not whether it is two dollars or five dollars, but whether the editor considers the benefit or substance of all research material relative to the general bulk of the computing profession before printing the articles. The fact that it is a cheap magazine does not add substance to an argument to continue printing research material of limited value to members. My comments are not directed specifically to the two authors of this paper as many other papers are equally inappropriate. Their article may well be a notable achievement. None of these bodies are currently controlled by academics, and the practitioner can expect his views to receive a fair hearing. I might add that, in view of the seriousness of Mr McGuiness' criticisms, it is disappointing that he did not use the opportunity to first check the publication cost figures, which are included in the annual budget statements printed in the Bulletin.

Mr McGuiness did not explain what in the paper referred to he found so objectionable, other than mentioning the title. It seems a little unfair to the authors that anyone should dismiss their work without first making some effort to know more about its contents, regardless of whether the paper is relevant to one's particular interests. No doubt academic research is not everyone's cup of tea. However, perhaps Mr McGuiness overlooked the fact that the Constitution of ACS prominently lists the advancement of computing knowledge as one of its main objectives. If Mr McGuiness believes that publication of research papers of all kinds in computing is not an activity worthy of the Society's resources, he should consider, again, raising the matter with the decision-making bodies.

I am as anxious as Mr McGuiness that the Journal should publish more material of direct interest to the great majority of its readers. If he would look over the authors' biographies in the August issue, he should note that several of them, though currently employed in academic institutions, have considerable industrial experience. (Perhaps it says something about the industry's working environment that these people chose to move away from it.) If Mr McGuiness knows of other authors willing and able to contribute technical material (not publicity or customer-services material) to the Journal without pay, I would be very pleased to hear about them.

FOLLOW-UP LETTER

Thank you very much for your prompt reply to my letter and I have noted the contents of your proposed reply with great interest.

It is unfortunate that you feel the need to defend a previous publication with such enthusiasm regardless of the merit of any criticism. Due to the emotional nature of your editorial reply, I must insist that if your proposed editorial comment is printed in its present form then this reply must also be considered for publication at the same time.

Firstly you have missed the major intent of my original letter and have chosen to comment on points which are immaterial to the major thesis. For example, your comments relative to the cost per member for the journal were interesting however the major point is not whether it is two dollars or five dollars, but whether the editor considers the benefit or substance of all research material relative to the general bulk of the computing profession before printing the articles. The fact that it is a cheap magazine does not add substance to an argument to continue printing research material of limited value to members. My comments are not directed specifically to the two authors of this paper as many other papers are equally inappropriate. Their article may well be a notable achievement. Unfortunately the vast majority of members would not be in a position to assess this.

I also noted the rather unfortunate comment by yourself which inferred that I did not support the publication of research papers "of all kinds". This is another illustration of the unfortunate attitude adopted by you in your rather childish reply to my letter. There is no basis in my original letter for this comment.

I also noted another comment by you that could only be regarded as an emotional and immature outburst, quote "It says something about the industry's working environment that these people chose to move away from it". If you believe that this type of comment constitutes a reasonable editorial reply to another ACS member's viewpoint
Letters to the Editor

G. E. McGuiness

EDITOR'S FOLLOW-UP COMMENTS

I am very pleased to see that Mr McGuiness actually shares my view that publication of research papers of "all kinds" is a task worthy of the Society's resources. Unfortunately, he says at the same time that he is against "esoteric" research papers. One cannot have it both ways.

Since I have already discussed Mr McGuiness' main point, the balance of material in the Journal, in my November Editorial, there is no need to repeat the remarks here. I will just make the point that the Journal's contents reflect the supply of material. If practitioners would not write papers and submit them to the Journal for consideration, there would not be any published. Readers might like to refer to some earlier discussions on the matter, on p. 113 of the August 1979 issue.

TWO COMMENTS ON PAPER BY JULIFF

I refer to the essay "Program Control by State Transition Tables" by Juliff, in which the author states that he could find no piece of work which covers this topic.

The book "Compiler Design Theory" written by P.M. Lewis et al, and published in 1976 by Addison-Wesley is one such work which gives an excellent presentation of this and similar program construction models such as finite state recognisers and pushdown stack machines.

Juliff, in his examples, defines program state explicitly with an identifier CURRENT-STATE which is used to index a 2-dimensional transition table. An equally valid model is to define the states implicitly i.e. for different locations in the program, test the input symbol and go directly to a program location (without accessing any table) that will process the symbol.

This of course negates the ease of maintenance that a transition table offers, but offers advantages in speed of execution and memory economy, where a high number of states exist and a high percentage of state symbol combinations result in a transition to the same error state.

At Wackers, we have been using state transition tables to control commercial on-line interactive screen programs. The success of software implemented in this way is due in part to the high reliability and modifiability that this methodology offers. Individual client's requirements differ to varying degrees but maintenance to standard programs is a safe and easy task.

However, the concept of state-control is more important as a design tool than as a maintenance aid. 'Michael-Jackson-type' structured techniques are of little use in multi-path interactive programming environments.

John Southgate,
Industrial Year Student, NSWIT
Wacker Partners Pty Ltd

It is indeed encouraging to find concepts previously considered the proper domain of computer science being applied to the world of commercial data processing as described in a recent ACJ article by Juliff. There is no doubt that this technique when properly used enhances both program development, reliability and maintainability. It is, however somewhat disconcerting to note the complete lack of references in that article. I have been teaching the file updating technique described in Example 3 since 1976 to classes at the University of New South Wales and within the ACS Professional Development Seminars in Basic Systems Analysis. A preliminary version of the algorithm was published in February 1978 (Grouse 1978) and subsequently included in "Basic Information Systems Analysis" by Brookes, Grouse, Lawrence and Jeffery, published by the University of New South Wales. The latter text has been used as notes for the above-mentioned Professional Development Seminars for two years.

References


P.J. Grouse
Associate Professor,
Department of Information Systems,
The University of New South Wales

AUTHOR'S REPLY

I am indebted to both Prof. Grouse and Mr Southgate for providing readers with references to additional material on this topic. I have been aware of Prof. Grouse's treatise on flowbacks for some time and should have acknowledged it in my paper. Like Mr Southgate, I have also used this technique extensively in on-line interactive software and agree with his observations as to its value as a design tool and aid to maintenance.

I also acknowledge private correspondence from Mr A. Henzell which provided the further references:

DAY, A.C., The Use of Symbol-State Tables

P. Juliff
Prahran CAE

CALL FOR PAPERS

SPECIAL ISSUE ON DATABASE MANAGEMENT

The Australian Computer Journal will publish a special issue on "Database Management" in November 1981. The issue will be under the Guest Editorship of Professor A.Y. Montgomery, Department of Computer Science, Monash University.

Research papers, tutorial articles and industry case studies on all aspects of computer databases and information systems are welcome, either as full papers or as short communications. Prospective authors should write to:

Professor A. Y. Montgomery, Guest Editor
ACJ Special Issue on Database Management
Dept. of Computer Science,
Monash University,
Clayton, Victoria 3168
Australia

to notify him of their intention to submit material for the issue and provide a brief summary of their intended contribution. To allow adequate time for processing, full papers must be received by 1 August 1981, and short communications by 15 August 1981. Material received after the deadlines may be considered for publication in later issues of the Journal.

Papers should be prepared in accordance with the guidelines published in the May 1980 issue of the Journal.

CALL FOR PAPERS

SPECIAL ISSUE ON SOFTWARE ENGINEERING

The Australian Computer Journal will publish a special issue on "Software Engineering" in early 1982. Research papers, tutorial articles and industry case studies on all aspects of the subject will be welcome, and both full papers and short communications will be considered.

Prospective authors should write to:

Professor P.C. Poole, Guest Editor,
ACJ Special Issue on Software Engineering,
Department of Computer Science,
University of Melbourne,
Parkville, Victoria, 3052,
Australia.

to notify him of their intention to submit material for the issue and provide a brief summary of their intended contribution.
This is the 200m², Snaplock access floor installed in June 1980 at the Perth offices of Woodside Petroleum Limited. It supports computers used to process data for the North-West Shelf gas field development.

The floor uses Series 800 metal panels on 300mm pedestals and has a Heritage II, monolithic, anti-static carpet finish.

Partitioning was also installed at the same time as the floor.

Cemac Tate's all-metal access floor system offers four panel sizes, with or without stringers and the widest range of finishes – high pressure laminate, wood parquet, cork and vinyl asbestos – as well as anti-static carpet.

Cemac also has a patented, adjustable pedestal base which ensures a level floor on a sloping slab. So, for access flooring, or complete office interior layout (floors, systems furniture, ceilings, partitions and task or ambient lighting), call Cemac Interiors.

Brochures and details from Cemac Interiors:

Sydney 290 3788
Melbourne 419 8233
Brisbane 221 5099
Perth 434 7888

Licensees:
Adelaide 45 3656
Hobart 29 5444
Perth 434 7888
The Australian Computer Journal is an official publication of the Australian Computer Society Incorporated.


EDITORIAL COMMITTEE: Editor: C.K. Yuen, CSIRO Division of Computing Research, PO Box 1800, Canberra, ACT, 2601. Associate Editors: J.M. Bennett, T. Pearcey, P.C. Poole, A.V. Montgomery, J. Lions.

SUBSCRIPTIONS: The annual subscription is $18.00. All subscriptions to the Journal are payable in advance and should be sent (in Australian currency) to the Australian Computer Society Inc., PO Box N26, Grosvenor Street, Sydney, 2000. A subscription form may be found below.

PRICE TO NON-Members: There are now four issues per annum. The price of individual copies of back issues still available is $2.00. Some already out of print. Issues for the current year are available at $5.00 per copy. All of these may be obtained from the National Secretariat, PO Box N26, Grosvenor Street, Sydney, 2000. No trade discounts are given, and agents should recover their own handling charges. Special rates apply to members of other Computer Societies and applications should be made to the Society concerned.

MEMBERS: The current issue of the Journal is supplied to personal members and to Corresponding Institutions. A member joining part-way through a calendar year is entitled to receive one copy of each issue of the Journal published earlier in that calendar year. Back numbers are supplied to members while supplies last, for a charge of $2.00 per copy. To ensure receipt of all issues, members should advise the Branch Honorary Secretary concerned, or the National Secretariat, promptly of any change of address.

MEMBERSHIP: Membership of the Society is via a Branch. Branches are autonomous in local matters, and may charge different membership subscriptions. Information may be obtained from the following Branch Honorary Secretaries. Canberra: PO Box 446, Canberra City, ACT, 2601. NSW: Science House, 35-43 Clarence St, Sydney, NSW, 2000. Qld: Box 1484, GPO, Brisbane, Qld, 4001. SA: Box 2423, GPO, Adelaide, SA, 5001. WA: Box F320, GPO, Perth, WA, 6001. Vic: PO Box 98, East Melbourne, Vic, 3002. Tas: PO Box 216, Sandy Bay, Tas, 7005.

CONTRIBUTIONS: All material for publication should be sent to the Editor for processing. Prospective authors may wish to consult manuscript preparation guidelines published in the May 1980 issue. The paragraphs below briefly summarise the essential details.

Types of Material: Four regular categories of material are published: Papers, Short Communications, Letters to the Editor and Book Reviews. Generally speaking, a paper will discuss significant new results of computing research and development, or provide a comprehensive summary of existing knowledge with the aim of broadening the outlook of Journal readers, or describe important computing experience or insight. Short Communications are concise discussions of computing research or application. A letter to the Editor will briefly comment on material previously appearing in the Journal or discuss a computing topic of current interest.

Refereeing: Papers and Short Communications are accepted if recommended by anonymous referees. Letters are published at the discretion of the Editor, and Book Reviews are written at the Editor's invitation upon receipt of review copies of published books. All accepted contributions may be subject to minor modifications to ensure uniformity of style.

Proofs and Reprints: Page proofs of Papers and Short Communications are sent to the authors for correction prior to publication. Fifty copies of reprints will be supplied to authors without charge. Reprints of individual papers may be purchased from the Printer (Publicity Press).

Format: Papers, Short Communications and Book Reviews should be typed in double spacing on A4 size paper, with 2.5cm margins on all four sides. The original, plus one copy [preferably two for Papers] should be submitted. References should be cited in standard Journal form, and generally diagrams should be ink-drawn on tracing paper or board with stencil or LetraJet lettering. Papers and Short Communications should have brief Abstracts, Key word lists and CR categories on the leading page, with authors' affiliations as a footnote. The authors of an accepted Paper will be asked to supply a brief biographical note for publication at the end.

This Journal is Abstracted or Reviewed by the following services:

Publisher Service
ACM Bibliography and Subject Index of Current Computing Literature
ACM Computing Reviews
AMS Mathematical Reviews
CSA Computer and Information Systems Abstracts, Data Processing Digest
ENGINEERING INDEX INC. Engineering Index
INSPEC Computer and Control Abstracts
INSPEC Electrical and Electronic Abstracts
SPRINGER-VERLAG Zentralblatt fur Mathematik und ihre Grenzgebiete

Copyright © 1981. Australian Computer Society Inc.

Production Management: Associated Business Publications, 28 Chippen Street, Chippendale, NSW, 2008. Tel: 699-5601, 699-1154. All advertising enquiries should be referred to the above address.