It takes architectural flexibility to get the most out of a 330 nsec mini.

Introducing the user microprogrammable VARIAN 73.

Microprogramming you can get your hands on, coupled with the speed to make a big difference in computing power. That's the VARIAN 73. An all new approach to minicomputer design and performance. It starts with architecture. Extendable. Multiple processors interleaved for incredibly large capacities and speeds. With expandable memory and multiple busses.

But even for openers, VARIAN 73 is loaded. Extremely compact, a complete 32K system in a seven inch chassis. And fast. 330 nsec full memory cycle using the latest semiconductor technology and you can even get a wallowing 660 nsec in a core option for less money. I/O data transfers to memory at rates exceeding 3 million words per second. And a 64-bit control word gives maximum speed and flexibility.

We've built in a dual asynchronous memory bus with a total information bandwidth of over one hundred million bits per second. Configure a system by connecting processors and dualport memories to the multibus and you've got a giant multiprocessor. The software package which is available right from the moment of introduction of the VARIAN 73 is enough to make most other minis look naked. Peripherals? VARIAN 73 has every option in the book. Send for the book.

Varian Pty. Ltd. 82 Christie St., St. Leonards, 2065. Phone 43 0673 varian data machines (vS)
## Contents

<table>
<thead>
<tr>
<th>AUTHOR/S</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. W. Goldsworthy</td>
<td>Computers and Privacy—A Review of the Younger Committee Report</td>
<td>3</td>
</tr>
<tr>
<td>P. J. Malcolm</td>
<td>A Machine Independent Design for an A.P.L. Translator</td>
<td>8</td>
</tr>
<tr>
<td>J. L. C. Macaskill</td>
<td>Diagonal-Matrix Formulation of Networks for Storage Economics</td>
<td>18</td>
</tr>
<tr>
<td>A. J. Bayes, W. McWhinney, C. R. Murry</td>
<td>A Program for Process Design and Optimisation</td>
<td>23</td>
</tr>
<tr>
<td>J. C. Gwatking</td>
<td>Random Index File Design</td>
<td>29</td>
</tr>
<tr>
<td>R. S. Northcote</td>
<td>A Discussion of the Business Data Processing Cost Benefit Relationship</td>
<td>35</td>
</tr>
<tr>
<td>R. P. Watkins</td>
<td>Syntax Analysis Using FPL</td>
<td>50</td>
</tr>
<tr>
<td>C. S. Wallace and G. K. Gupta</td>
<td>Towards a Theory of Documentation</td>
<td>57</td>
</tr>
<tr>
<td>A. D. Payne</td>
<td>General Linear Multistep Methods to Solve Ordinary Differential Equations</td>
<td>62</td>
</tr>
<tr>
<td>C. S. Wallace and G. K. Gupta</td>
<td>Program Executional and F-Algebras</td>
<td>70</td>
</tr>
<tr>
<td>A. Mazanov</td>
<td>A Numerical Method for Solving Population Equations Containing Variable Lags</td>
<td>74</td>
</tr>
<tr>
<td>A. J. Bayes</td>
<td>A Dynamic Programming Algorithm to Optimise Decision Table Code</td>
<td>77</td>
</tr>
<tr>
<td>R. J. Hill</td>
<td>Staged Introduction of Manufacturing Information Systems</td>
<td>96</td>
</tr>
<tr>
<td>N. W. Bennett and P. L. Sanger</td>
<td>The Development of the ACL Language and its Implementation ACL-NOVA</td>
<td>105</td>
</tr>
<tr>
<td>K. J. McDonnell and A. Y. Montgomery</td>
<td>The Design of Indexed Sequential Files</td>
<td>115</td>
</tr>
<tr>
<td>P. Gilmour</td>
<td>A General Validation Procedure for Computer Simulation Models</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>Award of A.C.S. Prizes for 1972</td>
<td>114</td>
</tr>
<tr>
<td>A.C.S. Data Transmission Liaison Group</td>
<td>A.C.S. Data Transmission Liaison Group</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>Balance Sheet</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>Book Reviews</td>
<td>41, 47, 48, 80, 114</td>
</tr>
<tr>
<td></td>
<td>Branch Notes</td>
<td>38, 81</td>
</tr>
<tr>
<td></td>
<td>Data Banks and Privacy—The Attitude of the A.C.S.</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>Editorial</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>European Chapter of ACM—International Computing Symposium</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>Letters to the Editor</td>
<td>47, 83</td>
</tr>
<tr>
<td></td>
<td>Machine Oriented Languages Bulletin</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Seventh Annual Report to Council</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>Sixth Australian Computer Conference—Call for Papers</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Society Financial Reports</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>Special Interest Group</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td>Standards Association of Australia</td>
<td>42, 131, 140</td>
</tr>
</tbody>
</table>
Index for the Australian Computer Journal, Vol. 5

1. Index of Titles of Papers, Books Reviewed, etc.

A.C.S. Data Transmission Liaison Group. 83
A Discussion of the Business Data Processing Cost/Benefit Relationship. 33
A Draft Standard for Digital Data Transmission. 140
A Dynamic Programming Algorithm to Optimise Decision Table Code. 77
A General Validation Procedure for Computer Simulation Models. 127
A Machine Independent Design for an A.P.L. Translator. 8
A Numerical Method for Solving Population Equations Containing Variable Lags. 74
A Program for Process Control and Optimisation. 23
A Survey of Automatic Flowchart Generators. 132
Award of A.C.S. Prizes for 1972. 114
Balance Sheet. 141
Branch Notes. 38, 81
Computer Security Management. (Review) 48
Computers and Privacy—A Review of the Younger Committee Report. 3
Computers in Society. (Review) 114
Curved Surfaces in Engineering. (Review) 48
Data Banks and Privacy—The Attitude of the A.C.S. 39
Diagonal—Matrix Formulation of Networks for Storage Economies. 18
Editorial. 2
European Chapter of the A.C.M.—International Computing Symposium. 38
General Linear Multistep Methods to Solve Ordinary Differential Equations. 62
Graphic Languages. (Review) 80
Information Representation and Manipulation in a Computer. (Review) 114
Introducing Computers. (Review) 114
Letters to the Editor. 47, 83
Machine Oriented Languages Bulletin. 34
Non-linear Optimization. (Review) 41
Program Execution on F-Algebras. 70
Random Index File Design. 29
Reliable E.D.P. Application Design. (Review) 80
Seventh Annual Report to Council. 84
Sixth Australian Computer Conference—Call for Papers. 7
Society Financial Report. 90
Special Interest Group. 126
Standards Association of Australia. 42, 131, 140
Staged Introduction of Manufacturing Information Systems. 96
Structured Programming. (Review) 47
Syntax Analysis Using F.P.L. 50
The Design of Indexed Sequential Files. 115
The Development of the ACL Language and its Implementation ACL-NOVA. 105
Towards a Theory of Documentation. 57

2. Index of Authors and Other Contributors

Bayes, A. J. 23, 77
Bennett, N. W. 105

Gilmour, P. 127
Goldsworthy, A. W. 3
Gupta, G. K. 62
Gwatking, J. C. 29

Hext, J. B. 114
Hill, R. I. 96

Jones, P. R. 83
Langridge, D. J. 80

Macaskill, J. L. C. 18
Malcolm, P. J.Fl 8
Maynard, G. B. 114
Mazanov, A. 74
McDonell, K. J. 115

McWhinney, W. 23
Montgomery, A. Y. 114, 115
Muetzeleldt, M. 80
Murry, C. R. 23

Northcote, R. S. 50
Osborne, M. R. 41
Payne, A. D. 70

Sanger, P. L. 105
Saw, B. M. 35
Schmidt, L. C. 48
Sewell, J. 48
Smith, R. G. 48

Taylor, A. A. 2
Wallace, C. S. 62
Watkins, R. F. 57, 132
3. Computing Reviews Categories Index

1. General Topics and Education
   1.1 Texts. 41, 47, 48, 80, 114.

2. Computing Milieu
   2.1 Philosophical and Social Implications. 3, 39.

3. Applications
   3.5 Management Data Processing. 35, 96.
   3.9 Miscellaneous. 23, 29, 115.

4. Software
   4.2 Programming Languages. 8, 50, 105.

4.6 Software Evaluation, Tests and Managements. 18.

5. Mathematics and Computation
   5.1 Numerical Analysis. 62, 74.
   5.4 Mathematical Programming. 77.

8. Functions
   8.0 General. 18, 52, 70, 132.
   8.1 Simulation and Modelling. 127.
RESPONSIVE TO EVERY DATA PROCESSING REQUIREMENT

In industry, commerce, government and education through continuous research and development and the dynamic application of the most advanced information systems to resolving individual problems and the rapid implementation of man's greatest ideas.

HARDWARE–SOFTWARE–TIME SHARING
Supported from one totally responsible source:

Honeywell
Honeywell Information Systems

SYDNEY 69 0355	MELBOURNE 26 3351	BRISBANE 21 6683	ADELAIDE 51 6203	CANBERRA 49 7966

When you get into something as complicated as networks, it's easy to get in trouble. What's difficult is knowing where to begin. Start with us. After all, you'll be using a lot of minis. And they'll probably be our PDP-8's and 11's. Simply because they're the minis almost everybody buys.

You'll be needing all the interfaces for those minis. For synchronous lines. Asynchronous lines. Memory to memory. We already have them all. All off the shelf. Ready to go. So you can hook up your network the way you want it, when you want it.

You'll want a host computer, a big timesharing computer, that fits right in with all our minis and works the way you do.

We make that, too. It's our DECsystem-10.

In fact, almost 30% of the computers used worldwide by independent timesharing services are DECsystem-10's. Networks are their business, so they have to go with the best computer available.

If you're interested in networks, we're the people to talk to. We have more timesharing installations than anyone in the world.

You can come to us because of all our minis, or all our interfaces, or our DECsystem-10 host computer, or our timesharing experience.

But we expect you'll come to us because we're the only company that gives you all of them.

Digital Equipment Australia Pty Ltd, 123 Willoughby Road, Crows Nest, NSW 2065.
Telephone 439 2566.
Melbourne (03) 699 2888,
Perth (092) 214993,
Brisbane (072) 29 3088,
Adelaide (082) 42 1339.
Developed by Floating Floors Inc., Ohio, for use in data processing installations, the Infinite Access Air Plenum system of flooring illustrated, meets the needs of modern technology in a wide variety of situations.

The Floating Floor offers the tremendous advantage of concealing all services. Cables, piping and ductwork are completely protected at all times. Each panel is precision manufactured of high pressure die-cast aluminium to guarantee limitless rearrangement by simply lifting and moving panels. This enables maintenance and reworking to proceed with no waste of time or disturbance to operations.

The Rintoul Infinite access floor (not illustrated) was developed by this company as a less expensive solution to the usual flooring problems associated with E.D.P. equipment and open landscaped offices.

The major difference between these two types of floors is the use of ‘rails’ in the ‘Rintoul’ floor. The somewhat higher original cost of the Floating Floors’ system can be offset over many years of savings on relocation, modernisation and maintenance without any excessive rearrangement cost.

Rintoul PTY. LTD.
10 Minogue Crescent, Glebe, N.S.W.
Telephone 660 6677

Sole Australian agents, for Floating Floors, Inc., Toledo, U.S.A.
Manufacturers of Infinite Access Floors

The new NCR Data Centre COM System you don't have to make room for.

You're looking at everything you need!

Until now, putting your computer tape data on microfilm usually meant expensive investments in new equipment. And elaborate floorspace alterations. The NCR Data Centre COM (Computer Output Microform) System is different.

With NCR's COM Service, you make no such costly investments. All the necessary reducing and processing equipment is located and maintained at an NCR Data Centre. And there's no need to remodel your office. You just need enough room for a microfiche reader.

NCR's new service combines the advantages of COM generated computer reports with the low-cost convenience of an NCR Data Centre. You get an inexpensive but complete data storage and retrieval system that you don't have to make room for.

Storage space is reduced as much as 98%. Paper and handling costs are cut way down. More data is at your fingertips. And retrieval is faster and easier.

Plus you get more efficient use of your computer. Easier to handle and read microfiche originals and copies. And document security and durability.

And there's the convenience of having your NCR Data Centre do all the COM processing and duplicating. Directly from your magnetic tapes. Before you print. At a price and turnaround time that's hard to match.

Call your nearest NCR Data Centre. Part of a nationwide network of data processing centres, it's backed by NCR's more than 85 years of experience and reliable service.

Your NCR Data Centre representative will give you all the details about the new COM System you don't have to make room for.
THE AUSTRALIAN COMPUTER JOURNAL

Price: $1.00 per copy to non-members, 50 cents per copy to members.

Volume 5, Number 3

NOVEMBER, 1973

Contents:

96-104 Staged Introduction of Manufacturing Information Systems
By R. J. HILL

105-114 The Development of the ACL Language and its Implementation ACL-NOVA
By N. W. BENNETT and P. L. SANGER

115-126 The Design of Indexed Sequential Files
By K. J. McDONELL and A. Y. MONTGOMERY

127-131 A General Validation Procedure for Computer Simulation Models
By P. GILMOUR

132-140 A Survey of Automatic Flowchart Generators
By R. P. WATKINS

Special Features:

114 Book Reviews
114 Award of A.C.S. Prizes for 1972
126 Special Interest Group
131 Standards Association of Australia
140 A Draft Standard for Digital Data Transmission
141-142 Balance Sheet

EDITORS' NOTE
The editor regrets that owing to circumstances beyond control no August issue will be published. Thus there will be only 3 issues in Volume 5.

Staged Introduction of Manufacturing Information Systems

By Richard J. Hill*

This is a review paper, which documents experience gained in several installations around Melbourne, mainly associated with manufacturing control systems. The experience has shown that many implementation problems are caused by attempting to install in one step systems which are too comprehensive. Also, detailed objectives are often frozen too early, before the users have really had sufficient experience to correctly specify their needs. The paper documents strategies which help avoid these problems in the manufacturing control area.

Key words and phrases:
Manufacturing, Staged Introduction

CR Categories: 3.5

1. INTRODUCTION

At the 1969 Computer conference at Adelaide several speakers touched upon the theme of "Evolution" rather than "Revolution". These speakers emphasised that users can no longer tolerate the convulsive termination of an old procedure and introduction of a new method. The purpose of this paper is to document several examples of this step-by-step introduction philosophy for manufacturing information systems.

It is an objective of the paper to help any system designer working in this application area by recounting experience gained with many companies in the Melbourne area over the last few years.

The writer has had the privilege of observing, and participating in, the planning and implementation of stock control and production scheduling systems in more than fifteen different manufacturing companies, in widely different industries. Although many manufacturing D.P. case histories have been published, they are mostly highly specific to one company. Conversely when a generalized approach is proposed in the literature it tends to be composed of hackneyed concepts like "maximum management involvement" which, true or not, are of little help to the system designer. In this paper a few system design philosophies will be discussed, and an attempt will be made to discuss pros and cons from the viewpoint of past experience. In this paper design considerations for systems to be run at a remote data centre are mentioned. This would be of interest in the Service Bureau environment and also to the larger company Data Processing Centre, which is often remote from the company's plants.

It could be emphasised that the size of the company, its commitment to data processing, and the experience level of its personnel could have a marked effect on the approach that could be adopted. Some of the comments in this paper do not apply to relatively small companies, considering the current state of the art.

2. OVERALL SYSTEM CONCEPT

In order to achieve a step-by-step or an evolutionary approach it is necessary to have an end objective in mind. However, it is unnecessary to have the final system too well defined as the detailed objectives can change during the introduction of the system.

A main requirement is that each following step can be attached to existing systems without too much difficulty. This is discussed further under "Design Approaches for Flexibility".

2.1 Overall Objectives

From the point of the overall system it is interesting to consider the purpose of the Manufacturing Information System. Systems have been implemented in factories to achieve the following:

1. Provide accounting data.
2. Provide managers with status reports.
3. Provide managers with exception reports.
4. Process management decisions into working instructions.
5. Apply operations research techniques to business plans.

All of these objectives can be generally expressed as "Aids to Resource Management".

2.2 Resource Management

We can consider Materials, Stocks, Plant, Power, Information, Vendors and Funds as "Resources", and Sales or Factory Shipments as "output". In this case a large part of the Manufacturing Manager's job is to optimise the "allocation of resources" to generate the maximum amount of output from the minimum consumption of resources.

The other parts of the Manager's job could be put under the heading of "resource husbandry" which covers the aspects of creating and "nurturing" resources such as personnel relations, training, new designs, new methods, new markets, and all other creative and leadership oriented aspects of management. (See Figure 1.)

In real life, of course, these two aspects are bound together in every manager's job (for the purpose of this paper a manager is any person who "manages" i.e. from Foreman to Director).

*IBM Australia Limited, 60 Market Street, Melbourne, Vic., 3000, Australia

To the system designer this interweaving of allocation and husbandry of resources is one of the frustrating aspects of the job of designing a working system. The reason is that resource allocation is generally a technical O.R. problem which is solvable given enough information, and enough computing power. "Resource husbandry" is an area where information can help the manager but there are few programmable solutions to the problems today.

2.3 Summarised System Objectives

We can look forward to the future when almost all the day-to-day running and most of the long term planning of manufacturing plants can be controlled by computer programs which will perform all routine resource allocations and will feed performance and status to the managers at each level. Only when the resource allocation problem cannot be handled by the programs will human intervention be required.

Managers will be freed from routine resource allocation problems to concentrate on creative tasks such as developing new processes and products, and identifying new market needs.

Since this is too general to be of use in establishing a real system some real objectives can be re-stated.

1. Automatic conversion of broad production plans into detailed schedules.
2. Automatic monitoring of stock levels and resource utilizations.
3. Automatic allocation of production and financial resources among contending demands.
4. Interactive re-allocation and replanning of production.
5. Interactive modelling of effects of management decisions on factory plans and performances.

Above all, the company would consider the areas with greatest potential cost reduction. However, it is a theme of this paper that the ultimate goals can be approached indirectly and that a quick achievement of a smaller cost reduction may be better than slow progress with a possible greater saving.

3. THE STAGED APPROACH TO IMPLEMENTATION

The staged approach is proposed to achieve two objectives:

- Quick results for the users after putting in effort.
- Maximum benefit extracted from data available at each stage before proceeding to further steps.

Before discussing the different approaches for computer systems, it is necessary to look at the components of a generalised information system for a factory as in Fig. 2.

Not all factories will have all functions, as they may be branch plants or working on a "make to order" basis and so on, but the system is generally applicable. Computer systems can be applied in all the areas shown, but investigation does not reveal any Australian company which has completely integrated all the functions with one unified computerised information system, although several are working towards it.

Similarly a computer can be applied on a stand-alone basis to help any one function. It is often not possible to economically justify collecting data to establish master files for one application, but it is possible to justify it for several applications. This is of course a problem with the step by step approach, as any one step, especially the initial one, may not be justified in its own right, but would pay off ultimately when more benefits are obtained.

3.1 Typical Factory Systems Implementation Plan

The word "Typical" refers to the historical situation where accounting has been the main computer application in most manufacturing companies. This is still the case today, in the Melbourne area. Some authorities predict that this accounting emphasis may

- LONG RANGE STRATEGY PLANNING
- BUDGET PREPARATION
- ENGINEERING DATA CONTROL
- SALES ORDER PROCESSING
- FORECASTING
- PRODUCTION SCHEDULING
- INVENTORY MANAGEMENT
- MATERIAL PLANNING
- PURCHASING
- FACTORY SCHEDULING
- DATA COLLECTION AND FEEDBACK
- ACCOUNTING

Figure 2: Components of a Factory Control System

Staged Introduction of Manufacturing Systems

The typical company launching into a manufacturing information system has an accounting and sales analysis system installed (1). This means that invoices are being processed one way or another so that issues from finished goods stock are known. It is usually the next step to make receipts into finished goods stock part of the system so that finished goods stock record-keeping can be installed on the computer (2).

Because receipts into finished goods stock can be also factory output dockets it is usual at this point to install some sort of accounting system to summarise factory output in various ways (3). A typical next step is to get transactions of part finished components and raw materials into the system so that stock record-keeping is effective for these items (4).

With the stock records now on file some simple or more complex stock replenishment planning system can be installed (5). This usually involves projecting future demand, and calculating an order quantity and re-order point.

Systems have been installed operating at the end-product level and at the finished component level. In Australia scientifically planned replenishment at the raw material level is not usual in manufacturing companies outside the process industry.

At this stage in the development effort, attention is usually paid to parts list files (Product structures, Bills of Material) for use by Engineering departments, Costing, but primarily by Production Planning (6). With this data available an end product schedule can be exploded down to component part and raw material requirements (7).

The computer can be used to check for shortages when work is released to the factory (8). Also, at this point, the computer files can be used to prepare some of the factory order documentation. As well as saving routine clerical tasks this step is almost essential as a method of ensuring that the parts list in the Data Base are valid (9).

If operations data are loaded into the Data Base at this point the factory paperwork can be completely computer prepared (10). However, having the operations data enables a computer program to extend the material plan into the requirements for manpower and machine time in the factory (11).

Further extension of the system from this stage requires maintenance of some sort of work-in-progress file. Since the computer is probably being used to prepare the shop paperwork it is easy to prepare some turnaround cards and maintain a file of factory status regarding work-in-progress (12).

With this data available the processing power of the computer can be used to perform some form of manufacturing scheduling, possibly by regular simulation and optimisation of the schedule (13). When this is implemented the attention of the company is usually turned to the data collection problems that exist in most plants, and some form of automated data collection system may be installed (14).

Of course, while this main plan is being followed many other applications would be implemented subordinate to the main tasks.

3.2 Starting Points for a Factory System
The typical system discussed previously represents...
an approach starting from the “Sales end” of the system. This is the way followed by a number of companies in Victoria.

A substantial minority have installed systems at the Factory Scheduling level (11, 12, 13 in Figure 3) and some intend to later integrate these into an overall system.

Starting with the data on the factory floor is an interesting possibility. Although it is not reported that any company in Australia has done this, the writer has been informed that at least one company in America installed an automatic data collection system in the factory as the first step in developing an integrated factory control system. Only at a later stage was a computer installed for processing the data and preparing plans, schedules etc. The DP Manager apparently reasoned that bad data is one of the biggest problems in a factory system, and it would be a good idea to solve this problem first on the shop floor, before investing any effort into the control system.

3.3 Development Strategies

In establishing a development plan there are several goals that can guide the selection of steps in implementing the system. For example, each next step could be chosen for:

1. Maximum mechanisation of routine clerical tasks. Or
2. Maximum payoff in improved control over factory plans. Or
3. Minimum time to achieve a complete system. Or
4. Minimal DP effort to achieve significant result.

It is submitted that the “typical” approach mentioned previously implied that objective 1 was chosen, and that at each stage gain could be measured by reduction in clerical headcount. Since the protestation is made very often that the systems are being installed “to improve management control—not to save labour” it is interesting to see that in actual cases the DP effort is rarely directed down a path guided by objective 2—maximise payoff in factory control—say by use of O.R. techniques, simulations, and other “difficult” methods. There do not appear to be many manufacturing companies other than those in the continuous process industry where this is done in a significant way in Australia.

Objectives 3 and 4 are similar and it is interesting to note that if you try to minimise the DP effort to result ratio—for example by using a small team—you will often achieve objective 3, installing a complete system in minimum time. At least one large company in Melbourne started this way. Referring to Figure 4 the development plan went like this:

The starting point was collection of data for Parts List files (2) in Figure 4 and Routing files (3) and these were first used to provide standardised Master Bills of Material and Methods Operation sheets etc. The first significant application installed was Standard Cost Development (4)—this used the parts list and operation files to build up end-product costs from material prices and operation times. (Sometimes this application is known as standard cost “pre-calculation”). This particular application is extremely useful also as a method of verifying the data base. It is always difficult to prevent errors creeping into the files. A list of end-product costs will quickly reveal most data errors as they tend to show up due to multiplying factors, and are easily picked out by a cost clerk with good product knowledge. Another advantage of this application is that it uses the whole data base, and being quick to install, provides a good fast return to the factory manager for the effort spent by his people in getting the data right.

The next application installed was explosion of end product costs into sub-assembly and component requirements (5). The requirements were netted against a stock balance figure keypunched from the manually maintained stock ledger cards. This is a reversal of the “classical” procedure, in that usually it is accepted that a company should maintain a computerised stock balance before using a computer to net-out requirements against stock. However, at least two companies in Melbourne have followed this “computer explosion netting against manual stock balance” principle with success. The advantage is that the explosion, which benefits greatly from being automated, is usually only needed once a month. A monthly run is a good remote data centre application, whereas computer-based stock record keeping offers marginal benefits to the factory user, unless run daily, which is expensive on a remote data centre.

The other steps followed by this company are more oriented to the normal approach.

4. A NETWORK APPROACH TO STAGED IMPLEMENTATION

In order to summarise the experience gained and referred to in this paper a network-type approach has been developed to illustrate the logical staged implementation of the manufacturing system. Figure 5 is a table of the different steps in various plans, and factors for consideration are noted against each application. Preceding steps are noted, and these are divided into essential versus desirable preceding steps, in order to illustrate the different plans available.

If all possible steps are drawn on one diagram as in Figure 6 we get a very complicated network. To simplify the diagram, it can be re-drawn, emphasising only those stages which pay off in aiding the factory management in planning the factory (see Figure 7).

If a company is operating on a remote data centre they usually wish to process weekly or less often.

Of all the applications on the network in Figure 7, only on-line plant scheduling implies an in-house com-
computer as being most desirable. It is interesting to observe that if one is not looking primarily for gains in reducing routine clerical tasks nearly all manufacturing planning applications with big payoff in helping management could be installed on a remote data centre.

5. GENERAL SYSTEMS ACCEPTANCE PROBLEMS

5.1 Example of an Acceptance Problem
In most metal-working job-shops the allocation of work to men and machines makes up a big part of the foreman's responsibility. Systems have been implemented successfully which automate this resource allocation function. The results have been that the foreman has been able to devote his time to looking after his department and spend less time in scheduling. Major savings in stock levels, improvements in throughput, and lead time reductions have been demonstrated.

In one plant, before the computer system was installed the foreman had regular contact with top management people over scheduling problems (rush jobs, changes). With the computer scheduling system installed the senior people made their adjustments via the computer reports. The amount of contact dropped and a foreman became less "important" in the company, leading to morale and co-operation problems. Also, this situation can seem bad for the upper level people because they think they have less "control" over the day by day operations in the plant.

Solutions to management attitude problems, as above, are outside the scope of this paper. However, it is relevant to comment that the successful computer system installation is often associated with a restructuring of the organisation in the company.

5.2 Participation in Installation of Systems
Some manufacturing systems are installed successfully with consultation with management at foreman

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Description</th>
<th>Normal Running Cycle</th>
<th>Checks Out Data Files</th>
<th>Step Advantages</th>
<th>Preceding Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Below</strong></td>
<td><strong>Below</strong></td>
<td><strong>Below</strong></td>
<td><strong>Below</strong></td>
</tr>
<tr>
<td>13</td>
<td>Standard cost development</td>
<td>R, Y</td>
<td>Y</td>
<td>Y</td>
<td>9, 12</td>
</tr>
<tr>
<td>14</td>
<td>Factory paperwork preparation</td>
<td>D, W</td>
<td>Y</td>
<td>Y</td>
<td>9, 12</td>
</tr>
<tr>
<td>15</td>
<td>Plant capacity requirements plan</td>
<td>M, W</td>
<td>N</td>
<td>N</td>
<td>0, 10, 11, 12</td>
</tr>
<tr>
<td>16</td>
<td>Factory load smoothing</td>
<td>W, D</td>
<td>N</td>
<td>N</td>
<td>0, 15, 17</td>
</tr>
<tr>
<td>17</td>
<td>Work in process file maint.</td>
<td>W, D</td>
<td>Y</td>
<td>Y</td>
<td>0, 4, 14</td>
</tr>
<tr>
<td>18</td>
<td>Scheduling of work in process</td>
<td>W, D</td>
<td>Y</td>
<td>Y</td>
<td>17, 16</td>
</tr>
<tr>
<td>185</td>
<td>On-line scheduling</td>
<td>W, D</td>
<td>Y</td>
<td>N</td>
<td>5, 9, 11</td>
</tr>
<tr>
<td>19</td>
<td>Purchasing system</td>
<td>W, M, D</td>
<td>Y</td>
<td>N</td>
<td>0, 17, 15, 19</td>
</tr>
<tr>
<td>20</td>
<td>Complete system for factory planning only</td>
<td>W, M, D</td>
<td>Y</td>
<td>N</td>
<td>0, 17, 15, 19</td>
</tr>
<tr>
<td>21</td>
<td>Complete state-of-the-art system</td>
<td>W, M, D</td>
<td>Y</td>
<td>N</td>
<td>0, 17, 15, 19</td>
</tr>
</tbody>
</table>

**Note symbols**
F — Frequently — on demand
D — Daily
W — Weekly
M — Monthly
Q — Quarterly
Y — Yearly
R — Rarely — on demand

Figure 5.2: Table of Stages in Implementing a Manufacturing System (Second Part)

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Description</th>
<th>Normal Running Cycle</th>
<th>Checks Out Data Files</th>
<th>Step Advantages</th>
<th>Preceding Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Below</strong></td>
<td><strong>Below</strong></td>
<td><strong>Below</strong></td>
<td><strong>Below</strong></td>
</tr>
<tr>
<td>0</td>
<td>Start point—all manual system</td>
<td>M</td>
<td>N</td>
<td></td>
<td>0, 2</td>
</tr>
<tr>
<td>1</td>
<td>Accounts receivable</td>
<td>F</td>
<td>Y</td>
<td>Y</td>
<td>0, 3</td>
</tr>
<tr>
<td>2</td>
<td>Sales invoice preparation</td>
<td>F</td>
<td>Y</td>
<td>Y</td>
<td>0, 1, 2, 4, 14</td>
</tr>
<tr>
<td>3</td>
<td>Finished stock record keeping</td>
<td>F, D, W</td>
<td>Y</td>
<td>Y</td>
<td>0, 12, 14</td>
</tr>
<tr>
<td>4</td>
<td>Factory output accounting</td>
<td>W, D</td>
<td>Y</td>
<td>Y</td>
<td>0, 12, 14</td>
</tr>
<tr>
<td>5</td>
<td>Component and raw material stock record keeping</td>
<td>F, D, W</td>
<td>Y</td>
<td>Y</td>
<td>0, 3, 14</td>
</tr>
<tr>
<td>6</td>
<td>Statistical forecasting</td>
<td>M, W</td>
<td>N</td>
<td>Y</td>
<td>0, 1, 2, 3</td>
</tr>
<tr>
<td>7</td>
<td>Statistical order point control</td>
<td>M, W</td>
<td>N</td>
<td>Y</td>
<td>0, 6, 8</td>
</tr>
<tr>
<td>8</td>
<td>Batch size planning</td>
<td>Q, R</td>
<td>N</td>
<td>N</td>
<td>0, 6, 10</td>
</tr>
<tr>
<td>9</td>
<td>Parts list processing</td>
<td>M, W, F</td>
<td>Y</td>
<td>Y</td>
<td>0, 9</td>
</tr>
<tr>
<td>10</td>
<td>Explosion of gross requirements</td>
<td>M, Q</td>
<td>N</td>
<td>Y</td>
<td>0, 9</td>
</tr>
<tr>
<td>11</td>
<td>Explosion, net against stock</td>
<td>M, W, F</td>
<td>Y</td>
<td>Y</td>
<td>0, 9</td>
</tr>
</tbody>
</table>

Figure 5.1: Table of Stages in Implementing a Manufacturing System (First Part)
company which produces various output summaries, some for use outside Australia. However, the data is consistently bad, even meaningless, because first-line management does not participate in the system to any worthwhile extent. If a worker's supervisor does not get anything more than edit error reports from the system, the supervisor will still do his best to ensure that the documents all balance correctly. However, little effort is made by the supervisors to check that the documents reflect the true status of the situation. This could be one reason why conventional hierarchical efficiency reporting systems sometimes fail.

Analysing the above case it can be seen that the supervisors should have been given a report that was meaningful to them. This is not easy to do—since the good supervisor already knows more about his own section than a computer report can summarize. However, a computer report showing his results compared with YTD trends and with other sections in his group would be interesting and useful—in fact a report of the type his own superior would need.

5.4 Systems Design for "Participation"
Rule: "A report to a manager should contain results summarized at a level above that particular manager."

The diagram attached (Figure 8) shows the types of system—Option A is the bad case exemplified in section 5.2 whereas, Option B is the ideal to be aimed at by the system designer. Although output of useful reports of all levels of management is the objective it can be difficult to attain and may not be achievable until the system is completely installed.

5.5 Importance of Fast Results
In D.P., it is often noted that small teams working against limited targets are much more productive than
larger teams. This general rule seems also to be true for the development of manufacturing control systems.

In order to generate and maintain interest among the production executives it is wise to respond quickly to needs and to fulfill requirements quickly. In several organisations it has been noted that very quick progress is made when small sections or divisions are installing a D.P. system—for example quite a sophisticated production control system can be installed in a matter of months in a small factory or workshop. Based on this success, the D.P. team gets a mission from the company to work on company-wide manufacturing systems, and in many cases the rate of progress drops off markedly.

In other cases the fast rate of progress is maintained. For example a company based in Melbourne with many small manufacturing subsidiaries has been able to take a production scheduling system, previously installed successfully at one plant and in a period of one year install the same system in several other factories in Melbourne and Sydney. The work has been done by a very small team and against quite definite limited objectives. When this success is compared with other manufacturing D.P. case histories one is led to conclude that continuously striving for perfection in some undefined generalised planning system is not worthwhile. In some companies there is little communication between D.P. and manufacturing management for political reasons, or because the D.P. people are basically accounting oriented. This communication problem can be the main reason for delays in getting results. Often a lot of time is wasted because the D.P. people try to build a “perfect” system to foil every conceivable objection from the users, reasonable or not.

5.6 Systems Design for “Fast Results”

Try to design your manufacturing system so that it produces useful results very quickly after starting work. If you can’t see useful results appearing inside three months then perhaps your project is too large, too complicated, and probably indigestible for the users.

It should not be necessary to spend two years designing and discussing your projected system before actually producing anything useful for the production manager, but the writer knows at least two companies where this has happened.

The staged implementation approach discussed in detail earlier is an essential requirement for fast implementation.

6. DESIGN APPROACHES FOR FLEXIBILITY

In the manufacturing area flexibility of D.P. systems is a prime objective. Organisations are notoriously subject to change, for example over a period of three years one large manufacturer in Melbourne changed its factory organisation twice in fundamental ways.

Starting with one plant run by an integrated control system, they split the factory into 8 or so independent profit centres, each with its own marketing, manufacturing, purchasing, and other responsibilities—including separate control systems. After being taken over by another company the profit centres were recombined into two large divisions—still being independent. The effect of this organisation changing on D.P. systems can be dramatic if the D.P. systems are very specific to a particular organisation.

Another factor encouraging flexibility is the desire of the manufacturing executives themselves to design
systems. Because the Managers themselves wish to influence the design, it will always be subject to change, especially in the performance reporting area.

Flexibility in D.P. systems usually implies a performance trade off. If you are primarily interested in machine throughput then you cannot afford the extra steps and redundant file space necessary for a flexible design.

Flexibility is essential with staged implementation, because we want to install parts of the system before later stages have been designed.

6.1 Flexibility, With Data Handling

Data Base is a popular discussion point in the D.P. industry, and is quite well known in the application of D.P. to manufacturing. Companies in the Melbourne area have been using various disk oriented data base techniques since about 1965 for their manufacturing systems. Flexibility is given by the data base technique of separating data handling at various levels from the application programs. This enables data organisation to be changed with less rewriting of running programs. The most advanced systems available now align the program data to the data in files at execution time. This enables the utmost flexibility with some offsetting costs in larger storage and some performance degradation.

Below this level there are many well known strategies available to increase the flexibility of data handling in systems, for example:

- Copy common file definitions and record layouts from the source library at compile time.
- Allow 25%-50% expansion area in all record layouts.
- Use standard large field sizes for counters, accumulators, and numeric constants.
- Avoid use of bit switches, binary fields, unsigned numeric fields and other "smart" programming tricks.
- Avoid use of the same field for different data items.
- If using modular programming, keep the I/O areas in the I/O module—so that records can be expanded without altering the application modules.

6.2 Flexibility, With System Logical Design

An overall manufacturing control system consists of a number of sub-systems covering different functional areas, see Figure 9.

Interfacing of the sub-systems is a critical area for ensuring flexibility. Also the interfacing of the sub-systems with the user in the outside world is another key point. Let us discuss these separately.

6.2.1 Interfacing Of Sub-Systems:

The sub-systems in a manufacturing control system feed data to one another. This is capable of being done in three ways, illustrated in Figure 10. In Option A we have the more common concept. A sub-system is developed, for example sales order handling, and when a further sub-system is installed a data file is generated for interfacing—for example a summary of sales is fed to the forecasting programs. Gradually more and more applications are developed and each interface is specially developed for each new application. Eventually modifications have to be made and as the structure becomes more and more complex the cost of these modifications becomes excessive because they affect all preceding interfacing systems.

As well as this computer operation becomes an awkward problem—as every application has to be run in strict sequence. This can be very annoying when one is waiting for late data during the end of month peak. Currently, the trend is towards the Common Data Base concept—ideally every program in every sub-system extracts data from and returns data to the Common Data Base and as mentioned previously, is completely independent of data base itself. This allows possibly the highest degree of flexibility attainable at the current state of the art.

From the point of view of the system designer he may be restricted by available storage from implementing the ideal concept as illustrated in Option B. The logical compromise is shown as Option C. This basically is where data exchange between sub-systems is by means of the data base, but data exchange between programs within the sub-system is by conventional means. This method is currently being adopted by many companies, especially with intermediate size computers and batch oriented processing. Advantages are that the data base can be off-line if large work files are needed during a sub-system run, and that large calculation programs can have available more main storage if they do not need to also interface with the Data Base. By still keeping to the data base concept...
for sub-system interfacing, the advantages of flexibility are retained.

6.2.2 Interfacing With The User:
In this case we are talking about processing the cards, documents or other input from the factory and preparing reports for the managers.

This is an area where the system designer must cope with the greatest amount of variability. It has been found that some companies have difficulty in persuading managers at the same level to use common reports, even within one factory, and certainly differences arise from factory to factory within one company. Also as mentioned previously there is a continuous change in emphasis and interest on the part of the users in the factory and the systems designer must cater for this. Companies around Melbourne have used several simple ways to achieve this flexibility at the user interface.

Option (i) All systems input and output is from or to magnetic files and simple RPG or similar programs are used to generate the reports and process the cards. The magnetic files carry all available data and the report writer can select from these and summarise if necessary. On the input side valid defaults are assumed and only varied if necessary by user input. These RPG-type programs can be easily modified, or scrapped and rewritten very quickly without much cost and without changing the basic systems logic programs.

Option (ii) At card read time, or report print time a separately written formatting module can be called by the application program. This formatting module, once again carries little logic, and can be quickly rewritten and linked into the mainline program.

Option (iii) At run time a series of control cards can be read by the application program which move data and parameter values to or from fixed data areas in storage representing the card or printline, and the standard input output area. These control cards can be in the form of MOVE statements which can even be coded by non-D.P. people.

Of the manufacturing systems known to the writer, the third option above is probably the most widely used. It is especially good in a bureau or data centre environment, as only one set of programs need be catalysted for a particular application. Multiple users are catered for by the run-time parameter cards.

The three options are shown in Figure 11. The D.P. purist would probably reject all three options as all of them would degrade performance, in terms of "throughput". However, the practical advantages are real and have been demonstrated.

8. CONCLUSION
This paper has discussed implementation of Computer Applications in manufacturing companies.

Figure 11: Three ways of achieving user interface flexibility

Some simple principles are expounded in the paper as guides to systems design for easy implementation. It has been shown that there are many stages by which a complete system can be installed and that several choices in installation sequence are open to the system designer.

If the end objective is fixed it is still possible to approach it by different methods which maximise the benefits possible within the constraints of available resources.

It is possible that manufacturing companies may be able to gain benefits by following a staged sequence of installing applications which are chosen for payoff in particular directions, rather than following the "usual" implementation procedure.

REFERENCES AND BIBLIOGRAPHY
APICS Bibliography of Production and Inventory Control Literature. (Editor—G. W. Plossl). American Production and Inventory Society, Inc.
1. INTRODUCTION

The first conversational computing facility used at the AAEC Research Establishment was a one-terminal interpretive system called ACTIV-8 (Bennett 1968) developed for a PDP-8 computer. The system was designed to give individual scientists and engineers an easy, direct way of solving small numerical problems and at the same time to provide results more quickly than by use of a large batch processing computer.

ACTIV-8 was implemented on a 4K PDP-8 computer, and proved very popular with those users for whom it was designed. To expand the facilities provided by the ACTIV-8 system a multi-user conversational interpreter was written for a NOVA computer. This interpreter was based on a new conversational language, ACL, developed from the experience gained using ACTIV-8. The implementation of the language is referred to as ACL-NOVA and was carried out on a 12K NOVA computer with five teletypewriter terminals (Sanger 1971).

The development and implementation of the AC-TIV-8 language is discussed below, leading to a discussion of the ACL language and its implementation. These discussions are limited to the most important features of ACTIV-8 and ACL. For a more detailed description of the systems the reader is referred to the reports (Bennett 1968 and Sanger 1971).

2. THE ACTIV-8 LANGUAGE

2.1 General Discussion

The design of the ACTIV-8 language was influenced by the fact that the people who would be using the system were already familiar with FORTRAN. The initial ACTIV-8 language was fairly simple. The main statements were an arithmetic statement, a GO TO statement, an arithmetic IF statement, a TYPE statement and an ACCEPT statement. Variable names were restricted to a letter followed by a number, and the only mode of arithmetic was real.

Statements were entered from the teletypewriter terminal in the form

statement number — space — statement

and were used to build up a stored program. Statement numbers were optional and, if present, consisted of two digit numbers in the range 00-99. Statements were stored strictly in the order in which they were entered, and the stored program was executed on receipt of an END statement.

2.2 Dynamic Syntax Checking

The teletypewriter attached to the PDP-8 computer is operated in full-duplex mode. This means that characters may be transmitted between the teletypewriter and the computer in both directions simultaneously. In this case, it also means that a character pressed on the keyboard is transmitted to the computer but is not automatically printed on the teleprinter. In normal practice, a character entered from the keyboard is read by the computer and transmitted back, or "echoed", immediately to the teleprinter, which thus appears to operate like a normal office typewriter.

By taking advantage of full-duplex mode operation, the computer can check if the character entered is valid in syntax and, if not, the character is simply not echoed back to the teletypewriter.

The dynamic syntax checking of input was an important part of the ACTIV-8 system and later played an important part in the design and implementation of the ACL language.

To see how this syntax checking was carried out, consider the case of an arithmetic expression. In its simplest terms, an arithmetic expression must consist of operands (i.e. variables and numbers) separated by operators (e.g. * and +). An expression must start with an operand (or unary + or -) and it must finish with an operand. The (simplified) structure of an arithmetic expression is illustrated by the directed-graph shown in Figure 1, and the syntax checking ensured that this pattern was obeyed. As shown in Figure 1, a "valid" Carriage Return was also required to complete a statement. Expressions containing brackets can be checked in the same way. An opening bracket may only appear at the start of an expression or after an operand, while closing brackets may only appear after an operand. An extra condition for brackets is that there may never be more closing brackets than
opening brackets, and before a carriage return is accepted there must be an equal number of opening and closing brackets.

Reference to the standard mathematical functions took the form of a three letter function name followed by an arithmetic expression enclosed in brackets (e.g. LOG(2)). The first indication of a function name was the combination of a letter followed by another letter instead of the number required for a variable name. An attempt was then made to verify these first two letters in a table of function names. If the second letter was not verified in this way, it was rejected. The character was not echoed to the teleprinter and the program looped to request a new character in its place. If the character was verified, the character was printed and that table entry used to check the rest of the function name.

All the statements in the ACTIV-8 language were checked in this way. For example, if the first character entered is the letter G, then at this stage the statement may be either a GO TO statement or an arithmetic statement. If the second character is the letter O, the user is committed to a GO TO statement. The only sequence of characters allowed from this point is space, T, O, space, number, number and carriage return. If the second character were a number, a commitment to an arithmetic statement would be made. Variable names were restricted to a letter and closing brackets.

Dynamic syntax checking offered both advantages and disadvantages for the ACTIV-8 system. Since all statements were checked on input there could be no syntax errors at execution time. This saved program space in the system and saved time in processing each statement. However, the sophistication of checking the syntax of incoming characters was, at that time, thought to preclude the possibility of backspacing over errors, particularly on a small machine. Once an error had been made, the entire line had to be abandoned.

2.3 Interpreter versus Compiler

Two broad methods of processing the language are available. A compiler can scan the statements and produce an object program which must be stored and can later be given control. An interpreter also scans the statements, but instead of generating actions to be stored and executed later, those actions are carried out as each statement is considered. Thus the interpretive approach does not require as much space as a compiler, but to execute a program each statement must be decoded from its source form each time control passes to it. The interpretive approach was chosen for the implementation of ACTIV-8 because of the limited space available.

An interpreter has a number of advantages over a compiler. For example, when a GO TO statement refers to a statement number that is not present in the source program, the process of compilation cannot be completed in the normal way. In an interpreter execution terminates only if an attempt is made to pass control to a statement with that statement number. The case where two source statements have the same statement number is very similar.

Another feature of an interpreter is that there is always a convenient link between the name of the variable and its value; that is, via the symbol table. With a compilation system, all variable names are translated into machine locations during compilation and the symbol table is not usually accessible during execution. The ACTIV-8 system takes advantage of this feature.
in the ACCEPT statement. The ACCEPT statement allows the user to input a value during execution which will be associated with a variable. Because the symbol table is available, ACTIV-8 allows the user to enter an arithmetic expression rather than just a numeric value. This expression is evaluated and its value is assigned to the variable. Note that this arithmetic expression is entered by the user during stored program execution. This proved to be very useful. For example, when the statement ACCEPT A1 is executed, the variable A1 could be increased by one by entering A1+1, A1 could be increased by twenty percent by entering A1*1.2, A1 could be given the same value as another variable by entering this variable name e.g. D4, and if A1 was expected to have a value in centimetres then 7 inches would be entered as 2.54*7.

With a compiler it is not normally possible during execution to check whether the program is using a variable which has been set to a particular value. In an interpreter, values can only be obtained by using a variable name to access the symbol table and this allows undefined variables to be detected.

2.4 The Communicate Statement

The generality of the possible responses to the ACCEPT statement led to the design of the communicate statement. This statement consisted of the question mark character stored as part of the program. The user can easily determine the value of a variable which has been set to a particular value. In an interpreter, values can only be obtained by using a variable name to access the symbol table and this allows undefined variables to be detected.

The communicate statement also allows the system to act like a desk calculator, since the program

\[ A2 = \sqrt{(S1^2 - A1^2) (S1^2 - B1^2) (S1^2 - C1^2)} \]

where the last line causes the value of the variable A2 to be printed.

To allow for the case where a communicate statement may not have been stored as part of a program, a facility was added so that the communicate state could be entered if the user pressed the question mark character during program execution. Processing of the current stored statement is completed and control is given to the communicate state, with the position in the program where the interruption took place being printed. The user can now interact with his program using the responses described above. This feature is also convenient for another user who wants to do a quick “desk” calculation and return control to the stored program (provided, of course, that variables in the suspended program have not been altered unintentionally).

2.5 Implementation

The ACTIV-8 language was implemented on a 4K PDP-8 computer supporting an Anlex line printer, a Burroughs card reader and one teletypewriter terminal. This computer was already being used to provide a card listing facility.

The PDP-8 computer has a 12-bit word which results in a fairly limited instruction set and an addressing scheme which can directly refer to only 256 words from any particular location. The combination of word length, restricted instruction set, fixed page addressing and limited storage available in the PDP-8 computer influenced the design of the ACTIV-8 system. To simplify the task of developing and testing the ACTIV-8 system, a program was written for the IBM 360/50 computer to accept PDP-8 source language from cards, assemble the PDP-8 program into a pseudo-core storage and simulate the actions of executing the assembled program on the PDP-8 computer. This type of program is known as an assembler/simulator.

3. THE ACL LANGUAGE

3.1 General discussion

To expand the facilities provided by the ACTIV-8 system a multi-user conversational interpreter was written for a NOVA computer. This interpreter was based on a more sophisticated language, ACL, developed as a result of the experience gained with ACTIV-8 and a clearer understanding of the needs of this class of user.

3.2 Development of the Language Structure

The most useful development had its origins in the communicate statement. Here the user was able to perform arithmetic statements and GO TO statements in an immediate mode. The ACL language recognised that, in theory, there need be no barrier to any statement being used in this way; and it would be very convenient to have this mode available on first entering the system. Hence there would be two modes

1) The system would accept statements put in from the teletypewriter; execute some statements in an immediate mode and store others.
2) The system would execute previously stored statements in an automatic mode.

To allow statements to be referenced for storage, deletion and editing, a sequence number was introduced. This appeared at the start of statements which were meant to be stored, distinguishing them from statements meant for immediate execution. A requirement on the exact layout of the statements was that the dynamic syntax checking which had proved useful in ACTIV-8 should be easily implemented. In this instance it led to a space character separating the sequence number and the actual statement. The two types of statement looked like this:

sequence number — space — statement

For the syntax checking, the sequence number was indistinguishable from a number as part of an arithmetic expression (this possibility is raised later, Section 3.5). The space, which was not allowed in arithmetic expressions, served to give an indication that a sequence number was meant.

Because of the FORTRAN background of most users and because statement numbers were used in ACTIV-8, it was felt that the new language should have statement numbers. Sequence numbers can also serve as targets for transfers of control, but in some cases statement numbers are more convenient. For example, in debugging a program one may wish to change the statement number of a statement. If only the sequence number was available a change might also mean a change in the order in which statements were executed sequentially. As both sequence and statement numbers can be used for transfer of control and they were both integer numbers, an ambiguity arose. This was resolved by having statement numbers from zero to 99 and sequence numbers from 100 to 999.

Again, a space was introduced between the statement number and the statement, this time for purely aesthetic reasons. The two types of statement now had this format:

sequence number — space — statement number — space — statement

For those stored statements which did not include a statement number, two spaces could have been used. To simplify the process the user could enter a single space and the system responded with 3 spaces: two for the statement number and one for the compulsory space. Also the user could ask for a new sequence number by giving a space at the start of the line. This sequence number is 10 greater than the last sequence number specified. Thus for a statement to be stored, a new sequence number and a null statement number may be nominated by two spaces.

3.3 Statement Modification

Provision was made for deleting stored statements by entering:

sequence number — space — carriage return

Statement numbers could be altered by entering:

sequence number — space — new statement number — space — carriage return

A facility was introduced to allow the users to backspace n characters by typing <n, to be able to recon-struct the current line in an EDIT mode (see below) by typing <<, and to be able to cancel the current line by typing <<<.

A statement was provided to allow users to modify stored statements in what is called edit mode. The statement was:

EDIT n

where n is an arithmetic statement or expression (see Section 3.5) indicating the sequence number of the statement to be changed. This statement is first displayed on the teletype. The user is now able to compose a new line consisting of characters taken sequentially from the old line or characters inserted from the keyboard. The ability to skip characters from the old line was provided. A complete syntax analysis is carried out on the new line as if it were being entered from the keyboard.

3.4 Variables

Variable names were expanded from a set format of a letter followed by a number in ACTIV-8 to the general form of a letter which may be followed by up to three letters or numbers. Arrays were introduced with array names which were variable names one or two characters long. The arrays could have either one or two subscripts and each subscript was allowed the generality of an arithmetic statement or expression. Function names remained as three characters, such as LOG.

3.5 Arithmetic Statements and Expressions

The basic operands in an expression are variables and numbers. The arithmetic operators are +, —, *, / and ^ (power) with the usual mathematical hierarchy. Assignment are indicated by = instead of the equals sign used in FORTRAN and ACTIV-8.

Immediate statements were designed to include the capability of evaluating arithmetic expressions and printing the results. Thus if one entered 2+3, the system would respond with the value 5.

If one wished to assign a value of LOG(2) to a variable, one could use A1 <-LOG(2). Statements should behave exactly the same way whether they were executed as part of a stored program or in immediate mode. Since a stored program would normally contain arithmetic assignment statements, it is important that these should not cause printing. Thus it was decided that the statement A1 <-LOG(2) should not cause printing. To inspect the value of A1, one could simply type in A1.

The first rule that determined whether the value of an expression should be printed was the absence of an assignment. A development of this led to the form (A1>6) which would cause both the assignment and its value to be printed. It was convenient to consider the form (A1>6) as having a value like any other operand and to include it as such in expressions; e.g. A2 <-14*(A1+6). This device of enclosing a complete arithmetic expression in brackets resulted in the first rule being modified.

If an expression began with a variable followed by an assignment arrow, then the expression was called an "arithmetic statement" and its value was not printed; if it commenced in any other way, it was called an
“arithmetic expression” and its value was printed. Thus the values of 2+3 and 4+(A1+B6) should be printed, while the values of A1+B6 and A2+B6+(C3-A1) should not.

Finally, the extra bracket pair around assignments used as operands was dropped on the understanding that assignment should proceed from right to left, with the value of the expression on the right of the assignment arrow being assigned to the variable on the left. Thus the expression A2+B6+(C3-A1) becomes A2+B6+C3-A1. This allows a very general structure for multiple assignments and includes the type of multiple assignments that occur in ALGOL e.g. A1=B1+C1+2 is allowed.

3.6 ACCEPT and TYPE Statements

The ACCEPT and TYPE statements were included in ACL. While the ACCEPT statement was unchanged, the generality of the TYPE statement was increased in a number of ways. Firstly, literals were allowed so that descriptive information as well as numeric values could be printed. Secondly, a variety of separators provided line control. Thus, if a comma were used as a separator, the next operand would be given on the same line. If a semi-colon were used, the next operand would be given on a new line. The form <arith expression> was introduced as an operand. This form was used to position the carriage to an absolute location. Lastly, a function called DPT was provided to allow numbers on different lines to be typed so that their decimal points were aligned vertically. The value of the DPT function is the position of the decimal point relative to the start of the number. If the number A3 has a value of 89.5 then DPT(A3) would have a value of 3. To ensure that the decimal point of a number is printed in column 10 the statement TYPE<10-DPT(A3),A3 may be used.

3.7 Control Statements

The GO TO statement was expanded to allow an expression instead of an absolute statement number. If the value of the expression was in the range zero to 99, the reference is to a statement number. If the value was in the range 100-999, a sequence number was required. This form encompassed the FORTRAN ASSIGN statement (GO TO I) and also the computed GO TO (GO TO A3(I)).

A CALL statement and its associated RETURN statement were introduced to allow the user to reference other statements within the program as subroutines. A logical IF statement was provided to give decision making capabilities. The format was

IF(expression,relational operator,expression)

statement

where the relational operator could be any of the six common FORTRAN relational operators. The generality of the expressions allowed the IF statement to be used as a loop control e.g.

IF(I=I+1.LE.100) GO TO 23

3.8 Supervisory Statements

A number of statements were introduced for auxiliary purposes. The EDIT statement falls into this category. Because of their usage, these statements are only available as immediate statements. A list of all the statements with an indication of these restrictions is given in Table 1.

<table>
<thead>
<tr>
<th>Table 1. List of ACL statements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Immediate statements</strong></td>
</tr>
<tr>
<td>Comments</td>
</tr>
<tr>
<td>Artistic expressions</td>
</tr>
<tr>
<td>FOR</td>
</tr>
<tr>
<td>GOTO {arith stmt or exprn}</td>
</tr>
<tr>
<td>list {45 arith stmt or exprn[,arith stmt or exprn]}</td>
</tr>
<tr>
<td>SYMBOLS [:]</td>
</tr>
<tr>
<td>CLEAR [6 variable[, variable][]]</td>
</tr>
<tr>
<td>SPACE</td>
</tr>
<tr>
<td>TRON{arith stmt or exprn}</td>
</tr>
<tr>
<td>TROFF {arith stmt or exprn}</td>
</tr>
</tbody>
</table>

3.9 Supervisory Statements

A number of statements were introduced for auxiliary purposes. The EDIT statement falls into this category. Because of their usage, these statements are only available as immediate statements. A list of all the statements with an indication of these restrictions is given in Table 1.

<table>
<thead>
<tr>
<th>Table 1. List of ACL statements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Immediate statements</strong></td>
</tr>
<tr>
<td>Comments</td>
</tr>
<tr>
<td>Artistic expressions</td>
</tr>
<tr>
<td>FOR</td>
</tr>
<tr>
<td>GOTO {arith stmt or exprn}</td>
</tr>
<tr>
<td>list {45 arith stmt or exprn[,arith stmt or exprn]}</td>
</tr>
<tr>
<td>SYMBOLS [:]</td>
</tr>
<tr>
<td>CLEAR [6 variable[, variable][]]</td>
</tr>
<tr>
<td>SPACE</td>
</tr>
<tr>
<td>TRON{arith stmt or exprn}</td>
</tr>
<tr>
<td>TROFF {arith stmt or exprn}</td>
</tr>
</tbody>
</table>

3.9 Supervisory Statements

A number of statements were introduced for auxiliary purposes. The EDIT statement falls into this category. Because of their usage, these statements are only available as immediate statements. A list of all the statements with an indication of these restrictions is given in Table 1.
A RUN statement is used to begin execution of a stored program starting at the statement with the lowest sequence number. A LIST statement is provided to give a listing of the stored statements. A companion SUSPEND statement is designed to record the state of the program as a series of intermediate arithmetic statements followed by an immediate GO TO statement. These two statements may be used to prepare a paper tape so that execution may be resumed at some later time.

Trace statements allow assignments that occur in individual stored statements to be printed, or allow every stored statement that is executed to be listed. Two conditional PAUSE statements are available. A pause before (PB) and a pause after (PA) statement will cause control to be returned to the user just before or just after a specified statement is executed. These two statements provide a valuable checkpoint facility and, together with the trace statements, form a powerful aid to program debugging.

4. DEVELOPMENT OF THE ACL-NOVA SYSTEM

4.1 General Discussion

The ACL language was implemented as a multiuser conversational interpreter on a 12K NOVA computer supporting five teletypewriter terminals.

The NOVA computer has a 16-bit word length, a powerful instruction set and four registers, two of which can be used as index registers. The addressing scheme on the NOVA allows direct reference to page zero (the first 256 locations), 256 locations centred about the program counter and 256 locations centred about each of the index registers. Each instruction occupies one NOVA word. An assembler/ simulator for the NOVA computer written to run on the IBM 360/50 computer (Sanger 1970) was also used to develop and test the ACL-NOVA system.

The resident part of the ACL-NOVA system uses the first 6K of the NOVA computer. Parameters such as the number of simultaneous users to be supported by the system, the size of each increment of work area to be allocated to a user and the total amount of work space available to the system are not assembled into the program but must be specified when the system is first loaded into the computer.

Work space may be reserved for terminals or left "floating" and, when a user signs on at a terminal either the space that was reserved for that terminal or one increment of work space is initialised. Dynamic allocation of work space allows the "floating" work space to be allocated to users who need more space, and this is returned to the system when the user completes work at his terminal.

4.2 Arithmetic

All arithmetic is performed on 32 bit floating point numbers stored as two NOVA words. These numbers have a sign bit, a 7 bit characteristic and a 24 bit fraction. This form corresponds to the shortest floating point number on an IBM 360 computer and can represent numbers approximately in the range 5.4E-79 <= number <= 7.2E+75. This choice simplified the development of the floating point software because the results obtained from the NOVA simulations could easily be compared with the corresponding IBM 360 hardware floating point instructions. The only difference between the two was that the NOVA routines were set up to round the result of each arithmetic operation to produce more uniform error distributions. The times taken to perform the arithmetic operations were: addition and subtraction, 0.55 msec; multiplication, 1.72 msec; division, 2.96 msec.

4.3. The ACL-NOVA Program

The ACL-NOVA program has two main parts. One part, the Interrupt Handler, syntax checks the input from the terminals and stores it essentially as a string of source characters in a buffer area located in the appropriate user work area. This part of the program also handles any output that must be sent to the terminals.

The second part, the Background Program, controls statement execution. Requests for statement execution from each terminal are treated at the same priority level. Each terminal is serviced in a round-robin approach with the computer processing one statement at a time for each user. Thus the Background Program: (i) executes an immediate statement by interpreting the source string stored in the user buffer area and performing the indicated operations, (ii) stores a statement for later execution by moving the character string from the buffer area to another part of the user work area, and (iii) executes a stored program by fetching one statement at a time from the user work area and moving this into the buffer area for processing as in (i).

The ACL-NOVA program spends most of its time in the Background Program executing statements or checking whether there is a statement to be executed. However, the Background Program can be interrupted at any time by input or output operations at a terminal and these interrupts are serviced completely before control is returned to the Background Program.

4.4 Allocation of Space within a Work Area

The first 135 words of each user’s work area is used for various pointers and buffers for the ACL-NOVA system. It includes two flag words and nineteen words of pointers used by the system as well as an input buffer, an internal code buffer and an output buffer. The remaining part of the user work area is used for storing statements and for the symbol table. To provide the most efficient use of core storage, statements are stored starting from the end of the buffer areas and continuing towards the end of the work area, while symbol table entries are stored starting at the end of the work area and continuing backwards towards the start of the work area. In this way a program with many statements but few variables, or a program with few statements but many variables can be handled. By setting up the work area in this way, the task of expanding user areas was simplified and this is discussed in Section 4.9.

4.5 Internal Code Buffer

When a statement is being entered at a terminal, it is syntax checked character by character and a copy of the “valid” characters is kept in the input buffer in input or external code form. At the same time, the
syntax checker translates these characters into an internal code to simplify statement processing and stores them in an internal code buffer. When the input statement reaches a point where the statement type can be determined, an appropriate indicator is stored at the front of the internal code (IC) buffer.

4.6 Editing Statements and Error Correction

When a stored program is listed at a terminal, each statement is translated back into external code form and stored in the output buffer from which it is sent one character at a time to the terminal. To allow statements to be modified using the EDIT statement, the required statement is listed at the terminal in the above way. The pointer to the current position in the output buffer is reset to point to the start of the output buffer.

When the space character is entered, the character indicated by the output buffer pointer is stored in the input buffer as virtual input from the terminal and is syntax checked in the normal way. If the character is valid, the IC buffer is updated, the input buffer pointer is increased by one and the output buffer pointer is increased by one; otherwise it is rejected. In the same way, new characters can be entered from the terminal during edit mode and these affect only the input buffer, the input buffer pointer and the IC buffer. When each DEL or RUB OUT character is entered, the output buffer pointer is increased by one, thus deleting one character from the statement.

A similar procedure allows input from the terminal to be modified in edit mode. The only difference is that after the carriage return characters are entered, the current contents of the input buffer are copied into the output buffer before sending a carriage return, line feed to the terminal. The original input is then edited as described above.

The correction of input from a terminal by deleting the last n characters may also be carried out simply. If there are m characters in the input buffer (m is always greater than n), then the input buffer pointer is reset to m-n and the contents of the input buffer are syntax checked in a loop that analyses 1 character, then 2 characters, then 3 characters, ... , then m-n characters so that the contents of the IC buffer are updated correctly. This multiple scan is particularly important in the case of an IF statement, and at the point in the input where the statement type may be altered by the deletion of input characters.

4.7 Symbol Table

Each symbol table entry uses four NOVA words; two words to store the variable name and two words to store the floating point value. This allows a simple variable name consisting of a letter, which may be followed by up to three letters or numbers, stored in internal code form padded with blanks where necessary as shown in Figure 2(a).

Subscripted variable names were chosen to be a letter, which may be followed by a letter or a number, stored in the first word in internal code form padded with a blank if necessary with the appropriate subscript or subscripts stored in the second word. This allows a singly subscripted variable to have a subscript in the range 0 to 65535, and each of the subscripts of a doubly subscripted variable to be in the range 0 to 255 as shown in Figure 2(b) and 2(c).

The fact that the internal codes for letters and numbers required at most 6 bits was used to distinguish between simple variables, singly subscripted variables and doubly subscripted variables. This means that the top bits of each character in the first word of a symbol table entry can be used as indicators. A singly subscripted variable is set up so that the top bit of the first character in the name is set to one. Doubly subscripted variables are set up with the top bit of each of the characters in the first word of the symbol table entry set to one. This is also shown in Figure 2(b) and 2(c). This scheme has the advantage that there is no conflict when the variables A, A(1), A(1,1) are referenced by a user, as occurs for example in FORTRAN.

The symbol table entries are not ordered and a given entry is located by a sequential search of the symbol table. The search is carried out by adding an entry to the end of the symbol table for the variable required and ensures that a match is obtained from the sequential search. If the match occurs on the last entry of the symbol table, this indicates that the variable is undefined. The choice of a sequential search method simplified the structure of each work area and is adequate for this application where the symbol table is quite small for most users.

4.8 Stored Statement Processing

Execution of a stored program begins with the execution of a RUN statement or an immediate GO TO statement. This causes the sequence number of
the first statement in the stored program or the sequence number of the statement referred to in the GO TO statement to be stored as the sequence number of the next statement to be executed and the stored program to be executed flag is turned on.

When this terminal is next serviced by the round-robin background program this statement is located in the user work area and is copied into the IC buffer for processing. The sequence number of this statement is stored as the sequence number of the statement being executed (this is referenced in error messages, noting pause conditions, for trace output etc.) and the sequence number of the next stored statement (which can easily be located by using the length of the statement being processed) is stored as the sequence number of the next statement to be processed. The stored statement is now executed from the IC buffer. In this process, the IC buffer is partially overwritten by replacement operands (see Section 4.10) and this is why the statement cannot be executed directly from the user work area. At this point control is returned to the Background Program.

Sequential statement processing continues in this way until a statement that alters the path of control is executed. The simplest way to do this is to execute a GO TO statement. In this case, once the statement has been processed the sequence number referred to in the GO TO statement is stored as the sequence number of the next statement to be processed.

The CALL statement can also be used to transfer control to another group of stored statements. This statement causes (i) the sequence number of the statement referred to in the CALL statement to be stored as the sequence number of the next statement to be executed, (ii) the sequence number to be used by the RETURN statement to be stored in this CALL statement in the user area, and (iii) the sequence number of this CALL statement to be stored as the sequence number to be used by the RETURN statement. This scheme allows nested CALL statements to be used.

When a RETURN statement is executed, the sequence number to be used by the RETURN statement contains the sequence number of the last CALL statement executed. In this case, the sequence number of the statement following the last CALL statement executed is stored as the sequence number of the next statement to be executed, and the sequence number stored within this CALL statement is stored as the new sequence number to be used by the RETURN statement.

4.9 Expansion of a User's Work Area

When an arithmetic statement or expression that occurs in any statement is executed, an initial scan through the expression counts the number, n, of assignment arrows. During execution of this expression, a maximum of 4n words may be added to the symbol table, and if this space is not available in the user's own area the system allocates the user an extra increment of work space from the "floating" space if this is possible. Similarly if a statement to be stored cannot fit into the user's work area, the system attempts to increase this work area.

When additional space is allocated to a user, that user's symbol table and the other user work areas that are stored above this user area must be copied into higher core locations as shown in Figure 3. The index to the start of the symbol table and the index for the next entry in the symbol table must be updated for this user while the amount of "floating" space available to the system and pointers to the other user work areas must also be updated.

After the user has completed his calculations by executing an END statement, the extra space that was obtained is returned to the system with the other user work areas stored above this user being shuffled down into lower core locations.

This dynamic allocation of work space gives the system considerable flexibility. The structure of the user work area made this scheme possible, while care was taken to ensure that the pointers in the 135 word area were used as displacements from the start of the user area rather than as absolute addresses.

4.10 Expression Evaluation

If a variable is followed by an assignment arrow, then during statement execution the expression to the right of the assignment arrow is evaluated and its value given to that variable. The variable name and its value in floating point form are stored in the symbol table. When multiple assignments occur in an expression, assuming first of all that the expression is bracket free, then the rightmost assignment arrow is located. The expression to the right of this is evaluated and the resulting value given to the variable. This process is continued until all the assignments have been carried out.

In more complicated expressions that contain a number of levels of brackets plus multiple assignments,
ACL-NOVA

the expression is evaluated by searching first for the rightmost opening bracket. The expression enclosed between the corresponding pair of opening and closing brackets (an expression at zero bracket level) is then evaluated by searching for the rightmost assignment arrow and proceeding as described above. This process is continued until the whole expression is reduced to zero bracket level and evaluated.

During the process of reducing an expression to zero bracket level, subscripted variables are replaced by subscripted variable replacement operands. A four character subscripted variable replacement operand of the form \( R'[i:j] \) was chosen (to cope with the simplest form of subscripted variable, namely \( A(1) \)) where the character \( R' \) indicates the start of a subscripted variable replacement operand, the character \( p \) is a pointer to the position of the next operator or delimiter in the expression, and the two characters \( ii \) form a 16-bit index to the appropriate symbol table entry relative to the start of the user work area.

Expressions at zero bracket level (and the bracket pair surrounding the expression, if any) are replaced by simple replacement operands. A three character simple replacement operand of the form \( Rpi \) was chosen (to cope with the simplest form of arithmetic expression, namely \( 3+2 \)) where the character \( R \) indicates the start of a simple replacement operand, the character \( p \) is a pointer to the position of the next operator or delimiter in the expression, and the character \( i \) is an index to the floating point value of the expression. If an expression already contains simple replacement operands, then the simple replacement operand describing the value of the expression is given the same index as the last simple replacement operand in the expression. By using the same simple replacement operand a number of times the number of NOVA words required for saving simple replacement operand values was minimised. The maximum number of simple replacement operands occurs for the expression \( A^*A+B*B+C+C+ \ldots \) up to 71 characters and turns out to be 18. (Because of the mathematical hierarchy the multiplications must be done first.)

However, if statements are being traced by a user, then statement processing is temporarily suspended when a symbol table assignment occurs so that trace output may be printed at the terminal. During this time, statements can be processed for other terminals and consequently the replacement operand index and the values of simple replacement operands must be stored in each user area. The input buffer is not being used at this stage and it is exactly 36 words long (i.e. the space required to store the floating point values of 18 simple replacement operands). Thus the simple replacement operand index is an index to the floating point values relative to the start of the input buffer.

5. CONCLUSIONS

The one-terminal ACTIV-8 system gave individual scientists and engineers an easy, direct way of solving small numerical problems. Statements entered at a terminal were stored in the user work area strictly in the order in which they were typed and the resulting stored program could later be executed. Stored program execution could be interrupted at any time to examine the contents of the symbol table, or to evaluate arithmetic statements, or to examine the value of individual variables or to allow tracing to occur. Stored program execution could then be continued from the same point or some other point in the program.

The ACL-NOVA system allows for stored program execution or for the execution of immediate statements to perform one-time or “desk calculator” calculations, to control the execution of a stored program and to perform various editing and debugging functions. Stored program execution can again be interrupted at any time by pressing the ? character. However this time stored statements can be inserted, modified or deleted as well as any of the immediate statements being executed, before program execution is continued from the same point or some other point in the program.

The EDIT statement is the most important of the error correction facilities provided in the ACL language and it provides a novel way of allowing characters to be copied from, inserted into or deleted from an existing statement. This facility combined with other special features such as ways of suspending stored program execution and powerful tracing statements provide interaction with the user. The flexibility of the IF statement and the generality of the arithmetic statements and expressions, with the freedom of multiple assignments, also adds a great deal of power to the ACL language.

Consideration of the full duplex mode of operation of teletypewriter terminals played an extremely important part in the design of the ACTIV-8 and ACL-NOVA systems. The dynamic syntax checking of keyboard input provides interaction with the computer and protects the user from trivial typing errors. It also has the advantage that statements do not have to be checked for syntax at run time thus improving program execution times.

The implementation of the ACL language as a multi-user conversational interpreter has provided a powerful conversational computing facility for many users in a time-sharing environment. The number of terminals to be supported, the size of work area increments and the available user area are specified when the ACL-NOVA system is first loaded into the computer and work space may be reserved for each terminal or left “floating” to provide the greatest flexibility for the system. The dynamic allocation of work space in the ACL-NOVA system provides for a more efficient use of the available user area than can be obtained from a system having fixed user partitions.

6. ACKNOWLEDGEMENTS

The authors thank Dr. D. J. Richardson and Mr. R. P. Backstrom for valuable discussions throughout this work.

It is interesting to relate the work described in this paper to time, and the above projects were carried out over the following periods: ACTIV-8, January 1968 to July 1968 (NWB); ACL language development, January 1969 to March 1970 (NWB, PLS); ACL-NOVA implementation, May 1969 to May 1971 (PLS, NWB).
Book Reviews


According to a recent report, all educated persons should have a knowledge of (1) the development of information processing, (2) the basic concepts of computer hardware and software, (3) the social impact of computer usage, and (4) the ways in which computers are applied. The author sets out to meet this need and divides his book into four sections accordingly. Much of Parts 1 and 2 is derived from his earlier book Computers in Business (now in paperback), but the remainder is original material discussing the role of computers in society.

The result is a sound but uninspiring book. It presents the material of a journalist through the pen of an academic, and the effect is neither exciting nor profound. Part 3, on ‘Computer influence in a changing society’ is especially disappointing in view of the book’s title. It discusses the effect on children and on individuals in a superficial, short-term way, ignoring some of the far greater long-term social consequences.

The publishers are to be commended for the low price. However, the design by Progressive Typographers leaves much to be desired. The layout of paragraphs, the black line down each page, the position or absence of page numbers, the sideways chapter headings, the excessive use of italics and the ridiculous cover can hardly be called ‘progressive’. A good design should help, not hinder, the reader.

J. B. HEXT


This book deals, at an introductory level, with the important topics of in-core data structure representation, searching and sorting as such should be a standard text in every Computer Science course. In addition this book should find itself on the shelf of every programmer to whom the succinct presentation of Knuth or the reading of algorithms in MIX are too difficult since all algorithms are presented in ALGOL which will be found. I believe, immediately readable to anybody familiar with any ‘high-level’ computer language. Each chapter is made complete by a well chosen bibliography and sets of short test examples, longer examination questions and yet longer topics suitable for project work. Outline answers to the examples are given in the Appendix.

In more detail then, what are the contents of the book? Chapters 1 to 3 deal with the subject of representation of data within a computer; neither the approach nor the material were found of very great interest to this reader. However, the meat of the book starts in Chapter 4 which contains a summary section on basic operations on Information Structures and is followed by a good coverage of arrays viewed as data structures, including introductions to the problems of handling sparse arrays. Chapter 5 deals with lists, giving a gentle introduction to simple linked lists, rings and doubly linked lists as well as stacks, queues and double ended queues. There follows a sensible summary of the costs and benefits of sequential versus linked allocation of storage. Chapter 6 deals with Trees which are viewed as the single most important non-linear data structure and this importance is reflected in the length of the chapter and the range and variety of tree representations and uses discussed. Briefly stated this chapter treats in detail the most common representations of binary trees including two linked and threaded tree implementations. Chapter 7 deals with Searching. Scanning methods — including the controlled scan method of Binary search for linear data structures — retrieval, insertion and deletion algorithms for binary trees, as well as collision strategies for randomly accessed (hash storage) tables are all well treated. Finally Chapter 8 addresses the topic of Sorting. Internal methods discussed are simple selection exchange and insertion methods, binary tree sort, radix exchange sort and address calculation sort. Merging methods described include balanced and polyphase, and polyphase. The chapter ends with a tabular comparison of sorting techniques and a sensible discussion on the process of selection of a sorting method for an actual data set.

A. Y. MONTGOMERY.

Introducing Computers, Murray Laver, Her Majesty’s Stationery Office, 1973, pp. 64 (Price 60p).

The author was formerly head of the UK Treasury’s Organization and Methods Division, Director of the Ministry of Technology’s Computer Division, and Director of the National Data Processing Service. He is now the Post Office Board Member for Data Processing.

The first edition of Laver’s book was published in 1965. With the second (1973) edition, the author has revised and updated the various sections to take account of the progress in computer technology during recent years. In particular, the sections dealing with on-line and operating systems software have been expanded.

The book includes chapters on the evolution of computers, their structures and functions, and their uses in offices, for management purposes, and in professional work. It also includes a range of diagrams, illustrations, photographs, and a reading list.

The publication is designed to provide an introduction for a wide range of readers in commerce, government, and industry. It should also be useful as introductory reading for in-house or college courses in data processing and computing.

G. B. MAYNARD

Award of A.C.S. Prizes for 1972

The only prize that has been awarded so far in 1972, has been the ANCAAC prize. The Case Study prize and the ACS Lecturer awards have not yet been decided. The Case Study prize has not been awarded for the past two years.

The ANCAAC prize for 1972 was awarded to Dr. T. Pearcey for his paper “Distributed Computer Systems” published in the Australian Computer Journal, February 1972, being judged the year’s best paper.

With regard to other awards, Branch Executive Committees should send their nominations to the National Secretary for the ACS Lecturer before 1 November. Papers eligible for nomination should have been presented to a Branch meeting during the twelve months ending 1st September.

The Case Study prize was inaugurated at the Council meeting in May 1971. Branches should submit their nominations in December of each year to the National Secretary.

Mr. Alan Taylor was appointed Convenor of the Prizes and Awards Committee in July 1973 at the Council Executive Committee meeting held in Sydney.
The Design of Indexed Sequential Files

By K. J. McDonell* and A. Y. Montgomery*

The paper commences with a tutorial discussion on the IBM Indexed Sequential Access Method (ISAM).

A method is then presented which gives a quantitative approach to the design of indexed sequential files. Formulae are presented which use results from a parametric simulation model to compute the expected file processing time.

The broad determining factors on file performance are considered, especially block size, cylinder index placement and overflow usage. Some suggested improvements to the basic indexed sequential organization are presented.

1. INDEXED SEQUENTIAL FILE STRUCTURE

Many manufacturers provide software for the establishment and maintenance of indexed sequential files: these differ in specific design, depending on hardware, software and equipment peculiarities, however the overall structure remains unaltered, IBM (1966), HONEYWELL (1968).

An indexed sequential file can be thought of as a sequential file with an hierarchy of indices. As the indices are scanned from the uppermost level to the lowermost level, the limits of the area (within the file) where the requested record may be found are narrowed. The terminal elements (lowest level entries) of the index hierarchy are address pointers to sections of the file within which data records are held in logical sequential order. This logical order is maintained by either,

(1) holding the records in physically contiguous storage locations (e.g. main or prime data area of the file), or

(2) chaining the records together by appending an address field to each record which holds the actual address of the next record in the logical chain (e.g. overflow areas of the file).

The structure of an indexed sequential file is sufficiently flexible to allow a number of possible processing modes. Random access to any record on the file can be achieved using the indices. Unlike a simple sequential file, an indexed sequential file can accommodate insitu updating involving additions and deletion transactions. (See Section 3).

It should also be noted that if the terminal elements of the index hierarchy can be accessed in sequential order, then the file can be processed sequentially, either in an insitu or a father-son type of updating procedure. In general, these terminal elements or address pointers occur in blocks at the beginning of each cylinder, and hence they can easily be located when processing transfers from one cylinder to the next, without scanning the upper levels of the index hierarchy. Figure 1.1 shows the structure of the index hierarchy.

According to Chapin (1969), the main advantage an indexed sequential file has over a sequential file is the ability to permit effective access to randomly selected records. The disadvantages are additional storage requirements, longer minimum time to access a record and additional overhead associated with file maintenance. Compared with a random file organization, he considers the main advantages of an indexed sequential organization to lie in the omission of time consuming key transforms and the ability to allow effective sequential processing, whilst the disadvantages are longer minimum access time to a record and the additional overhead associated with index maintenance.

2. INDEXED SEQUENTIAL ACCESS METHOD

We shall proceed to investigate a specific implementation of indexed sequential files, namely IBM's Indexed Sequential Access Method (ISAM) as implemented on the series 360 machines, IBM (1966).

ISAM files are designed with three main areas:—

(1) Prime Data Area (PDA)
(2) Overflow Areas, and
(3) Indices.

In the following we define

(1) a record as a discrete piece of logical information, and

Figure 1.1. Index hierarchy as used in an indexed sequential file.

* Department of Information Science, Monash University, Clayton, Victoria, 3168.

2.1 Prime Data Areas
These are the areas of the file into which most records are placed, in logical and physical sequence, ordered on one key field. Records may be "blocked" (more than 1 record per block) or "unblocked" (1 record per block). Blocks are formatted with keys, see Figure 2.1.

Note that if the records are unblocked, then the record key need not be repeated in the data area, however for blocked records, the record keys must appear in the data area and the key area contains the highest record key in the following data block.

2.2 Overflow Areas
Initially the file is loaded with all the records in the PDA, however if any insitu processing involving additions is to occur, some overflow space must be allocated.

Overflow records must be unblocked, formatted with keys and include a 10-byte link or address field. Note that if the PDA records are blocked, then the record key must appear in both the key area and the data area for an overflow record. This is to maintain a constant format for logical records in all sections of the file. The link field is used to chain overflow records together into logically sequential chains from sections of the PDA (usually one possible chain per track). The physical order bears no resemblance to the sequential order, which is preserved by following the chains from each track.

Overflow areas may be provided at either or both of the following levels within the structure:

(i) cylinder overflow -- a user assigned number of whole tracks is reserved in each cylinder, to accommodate overflow from within that cylinder
(ii) independent overflow -- a user assigned number of whole cylinders is reserved to accommodate overflow from any cylinder in which the cylinder overflow area has been exhausted.

2.3 Indices
Index sequential organization requires the creation and maintenance of two or more levels of indices.

(i) Track Index -- this is the lowest level index and there is one such index for each cylinder containing a prime data area. It is normally situated at the beginning of the cylinder. There is one entry for each track in the cylinder and each entry contains sufficient information to determine the range of key values both in the PDA track and the overflow chain from that track, as well as any information required to find the address of the start of the PDA track and the address of the beginning of the overflow chain.

For ISAM, this requires 4 fields per entry:

(i) Highest key on this PDA track
(ii) Address of this PDA track
(iii) Highest key to overflow from this PDA track
(iv) Start of overflow chain (address of lowest key to overflow from this PDA track).

Each entry comprises 2 blocks of information as shown in Figure 2.2. In addition the track index contains the Cylinder Overflow Control Record (COCR) which is used, when cylinder overflow is specified, to designate the next available record space in overflow. Figure 2.3 shows a typical track index and associated data areas.

(ii) Cylinder Index -- a higher level index whose entries indicate the highest key held on a cylinder and the address of the track index for that cylinder. This index is normally located on disk.

(iii) Master Index -- an optional higher level index, which is used when the cylinder index is too large to scan fully for each transaction. Each entry contains the highest key reference in a particular section of the cylinder index and the starting address of that section of the cylinder index. This index may be either core or disk resident, and may itself be referenced by further higher level indices.

Figure 2.4 shows the arrangement of indices and data areas in an indexed sequential file.

3. PROCESSING AN INDEXED SEQUENTIAL FILE
As previously mentioned, three modes of processing can be achieved with an indexed sequential file, namely

(1) father-son sequential
(2) insitu sequential, and
(3) random insitu.

A father-son update, effectively means a total re-organization of the file, with no records loaded into overflow. Therefore overflow need not be provided if only this mode of updating is to be applied to the file.

On the other hand, insitu processing involving addition and deletion transactions requires the allocation and use of overflow areas.

Random insitu processing is available on ISAM files. It is perhaps interesting to note that as Nijssen (1971) points out, a hash addressed random access file may be processed efficiently in a sequential mode, thereby providing both modes of processing available on an indexed sequential file without the overhead of index maintenance. He proposes that the transactions file be sorted into ascending transformed address sequence. Two possible disadvantages of this approach may be
an additional sort may be required to produce update reports in a usable sequence, and
(2) random access files incur a processing overhead in following and maintaining overflow chains, which may offset the index maintenance and processing involved in indexed sequential files.

3.1 Random Record Retrieval
Two main operations are involved in retrieving a random record from an indexed sequential file:
(1) index searching, and
(2) either PDA or overflow searching.
Since all indices are held in ascending key sequence, index searching is simply a matter of scanning the master index until a key value greater than or equal to the request key is found. The address corresponding to the last extracted index key value is the address of the start of the correct section of the cylinder index. Once the cylinder index section has been located, the scan algorithm shown in Figure 3.1 will extract the address of the correct track index. Once the track index has been located, the scan algorithm will extract either the address of the correct PDA track or the address of the beginning of the correct overflow chain.

If a PDA search is required, the scan algorithm will return the correct block. If the PDA is unblocked then this is

the requested record or the first record with a key greater than the requested key (if the requested record is not on the file). If the PDA is blocked then the block must be searched to determine if the requested record is present.

An overflow search merely involves retrieving the records in the chain in their logical sequence until either the requested record is found or a record is retrieved with a key value greater than that of the requested record (requested record is not on the file).

It must be emphasized that with the chain-channel-command facility available on IBM systems, the scan algorithm takes a maximum of one disk revolution, once the seek has been completed and the index point found. This is possible because there is no interrupt and return to the application program until the routine is exited, therefore the testing in loop-1 and loop-2 is a channel/controller function and can be executed in the time between consecutive key areas appearing under the heads. Consideration to systems not equipped with chain-channel-command is given in Appendix 1.

3.2 Insitu Record Deletion
In order that a record may be deleted, it must first be retrieved (either sequentially, or as outlined in Section 3.1). The only option available under ISAM is to 'tag' the record

The requested record or the first record with a key greater than the requested key (if the requested record is not on the file). If the PDA is blocked then the block must be searched to determine if the requested record is present.

An overflow search merely involves retrieving the records in the chain in their logical sequence until either the requested record is found or a record is retrieved with a key value greater than that of the requested record (requested record is not on the file).

It must be emphasized that with the chain-channel-command facility available on IBM systems, the scan algorithm takes a maximum of one disk revolution, once the seek has been completed and the index point found. This is possible because there is no interrupt and return to the application program until the routine is exited, therefore the testing in loop-1 and loop-2 is a channel/controller function and can be executed in the time between consecutive key areas appearing under the heads. Consideration to systems not equipped with chain-channel-command is given in Appendix 1.

3.2 Insitu Record Deletion
In order that a record may be deleted, it must first be retrieved (either sequentially, or as outlined in Section 3.1). The only option available under ISAM is to 'tag' the record

The requested record or the first record with a key greater than the requested key (if the requested record is not on the file). If the PDA is blocked then the block must be searched to determine if the requested record is present.

An overflow search merely involves retrieving the records in the chain in their logical sequence until either the requested record is found or a record is retrieved with a key value greater than that of the requested record (requested record is not on the file).

It must be emphasized that with the chain-channel-command facility available on IBM systems, the scan algorithm takes a maximum of one disk revolution, once the seek has been completed and the index point found. This is possible because there is no interrupt and return to the application program until the routine is exited, therefore the testing in loop-1 and loop-2 is a channel/controller function and can be executed in the time between consecutive key areas appearing under the heads. Consideration to systems not equipped with chain-channel-command is given in Appendix 1.
3.3 Insitu Record Addition

In effect an attempt is made to retrieve the record which is to be added. This should fail, but will return the record which is logically the next one after the record to be inserted.

If the record is to be inserted into overflow, the next available overflow space is found from the COCR (if provided) or from the master index. The new record is written into that location and the link field set to the address of the next logical record. The previous logical record must be retrieved and its link field updated to point to the address of the new record. It is possible that the inserted record is the first in the overflow chain, in that case, the previous logical record is in fact the track index overflow entry, and its address part must be updated.

A PDA insertion requires the new record to be inserted in the correct physical and logical sequence. This necessitates "shuffling" all the records between the new record and the end of the track, along one position. The key areas of each block must be updated as the block is written back to reflect the new contents of the block, also the prime data entry in the track index must have its key area changed to show the highest key value currently held on the PDA track. Note this is the only time the track index key values are altered, and they can only ever be reduced in value.

The record which is "bumped" off the end of the track is treated as an addition to overflow, in this case being inserted as the logically first record in the chain. The only exception to this procedure is if the bumped record is one which had previously been tagged as deleted. In this case it is forgotten and is the only way, short of reloading the file, that a deleted record is physically removed from the file.

4. SPACE ALLOCATION

Firstly, the following notation shall be used

\[ [x]^+ = \text{next integer} \geq x \]
\[ [x]^-= \text{next integer} \leq x \]
We shall assume an IBM 2314 disk, IBM (1966); the method would require minor modification for most other devices.

4.1 PDA blocks/track. (PBPT)
The following parameters are known:
- key length (KL)
- record data length (RDL)
- records per block (RPB)
- Block data length (BDL) = \( RPB \times RDL = RDL, RPB = 1 \)
- \( RPB > 1 \)

For most disk devices, the following physical constants are known:
- TC - full track capacity (bytes)
- BO - block overhead for all blocks but the last (count area, address marker and gaps)
- LBO - overhead for last block
- VG - variable gap factor (accounts for effect of block length on the gap length).

As an example, the IBM2314 device has the following constant values:
- TC = 7294 bytes
- BO = 146 bytes
- LBO = 45 bytes
- VG = 2137/2048

\[ \text{PBPT} = \frac{\text{TC} - \text{LBO} - (\text{KL} + \text{BDL})}{\text{BO} + \text{VG} \times (\text{KL} + \text{BDL})} + 1 \]

4.2 Overflow blocks [records]/track (OBPT)
Overflow block data length (OBDL) = \( RDL + 10, RPB = 1 \)
- \( RDL + 10 + \text{KL}, RPB > 1 \)

Note: 10 bytes required for 'LINK' field
\[ \text{OBPT} = \frac{\text{TC} - \text{LBO} - (\text{KL} + \text{OBDL})}{\text{BO} + \text{VG} \times (\text{KL} + \text{OBDL})} + 1 \]

4.3 Index blocks/track (IBPT)
Index block data length (IBDL)
- IBDL = 10
\[ \text{IBPT} = \frac{\text{TC} - \text{LBO} - (\text{KL} + \text{IBDL})}{\text{BO} + \text{VG} \times (\text{KL} + \text{IBDL})} + 1 \]

Note: 10 bytes required for address part of index entry.

4.4 Cylinder overflow
If the file is to be processed in an insitu mode, allowing additions to the file, then some overflow space must be allocated.

Cylinder overflow is the most efficient (in terms of processing time) form of overflow, since a probe into cylinder overflow does not require a head movement, unlike general overflow. On the other hand, cylinder overflow must be allocated in multiples of one track, the same amount in each cylinder – obviously this could be wasteful of space, especially if the mean number of additions per cylinder before re-organization was small.

If cylinder overflow is required, the problem is how much will be necessary? The only systematic solution to this problem is to simulate the operation of the file. Simulation results have shown the following broad trends to hold for files which are not expanding, i.e. number of insertions and deletions are equal. If less than 10% of all records are in overflow, then virtually all additions to the file result in a record being added to overflow. Above about 10%, overflow, approximately 90% of all additions cause a record to be added to overflow, see Figure 4.1.

It must be stressed that this growth is only slightly affected by the frequency of deletions. Because index entry key values can only ever become smaller (when a record is bumped from a PDA track), the index points progressively map more of the possible key values into the overflow area. This, combined with the fact that deleted records are not removed from their overflow chains, means that more and more transactions are directed to overflow and none of the overflow spaces can be re-used, therefore the use of overflow space is little affected by deleting records from the file.

4.5 An Iterative solution to the Space Allocation Problem
The solution to the space allocation problem is as follows,
1. Choose a starting number of PDA tracks/cylinder (NPT)
2. Calculate the number of blocks required for the track index (TIB)
   \[ \text{TIB} = 2 \times \text{NPT} + 1 \]
3. Determine the number of PDA records/cylinder (PRPC), given the number of records/block (RPB)
   \[ \text{PRPC} = \text{NPT} \times \text{PBPT} \times \text{RPB} \]
4. If the number of data tracks/cylinder is NT, then calculate number of overflow tracks/cylinder (NOT), assuming the PDA occupies an integral number of tracks. (This assumption is not strictly correct, however it is a trivial matter to modify steps (4) and (5) to reflect the fact that the last track index block and the first PDA block may share the same track.)
   \[ \text{NOT} = \text{NT} - \text{NPT} - \frac{\text{TIB}^+}{\text{IBPT}} \]
5. Determine the number of overflow record spaces/cylinder (ORPC)
   \[ \text{ORPC} = \text{NOT} \times \text{ORPT} \]
(6) Using the number of records on the file (NREC), determine the size of the file in terms of cylinders (NC):

$$\text{NC} = \left\lceil \frac{\text{NREC}}{\text{PRPC}} \right\rceil$$

(7) Given the number of additions expected (NA), determine the mean number of additions per cylinder before re-organization (MAPC):

$$\text{MAPC} = \frac{\text{NA} \times \text{PRPC}}{\text{NREC}}$$

(8) Using the expected rate of overflow growth (ROG), determine if sufficient cylinder overflow space has been allocated, i.e., if MAPC \times ROG \leq ORPC.

If insufficient overflow space has been allocated, choose a smaller value of NPT and repeat steps (2) through (8).

Note this method assumes the designer has an idea of both the file size (number of records) and the mean number of additions before re-organization. Re-organization requires re-loading the file with no records in overflow, which may be accomplished by running a father-son sequential update against the file.

An example of a space allocation calculation is given in Appendix 2.

5. TIME TO PROCESS THE FILE

In the following presentation, we shall use the undermentioned set of symbols.

(a) terms relating to space allocation

- NPT: number of PDA tracks/cylinder
- IBPT: index blocks/track
- PBPT: PDA blocks/track
- FI: fraction of track occupied by the whole track index
- FP: fraction of track occupied by one PDA block
- FO: fraction of track occupied by one overflow record

(Note: FI, FP and FO include associated gaps, count areas, address markers etc.)

(b) terms relating to the physical characteristics of the disk device.

- SMIN: track to track head movement time in units of disk revolutions
- SAVE: average head movement time from the cylinder index to a data cylinder (for this file) in units of disk revolutions
- T: disk revolution time in milliseconds.

(c) terms relating to accessing the data

- PPR: probability that a retrieval transaction is directed to PDA.
- NPSR: average number of PDA blocks scanned per PDA retrieval
- POR: probability that a retrieval transaction is directed to overflow (equals 1.0 – PPR).
- NOSR: average number of overflow records scanned per overflow retrieval
- PPI: probability that an insertion transaction is directed to the PDA
- NPSI: average number of PDA blocks scanned to locate the correct position for a PDA insertion
- PBD: probability that a deleted record is ‘bumped’ off the end of the track during a PDA insertion

POI: probability that an insertion transaction directed to overflow causes a record to be added at the beginning of an overflow chain

POFI: probability that an insertion transaction directed to overflow causes a record to be added into the middle or end of an overflow chain (note POI = POFI + POMI)

POMI: probability that an insertion transaction causes a record to be added into the middle or end of an overflow chain

NOSI: average number of overflow records scanned to locate the correct position for an overflow insertion into the middle or end of a chain.

Note: At first sight, one could expect terms like NOSR and NOSI, PPR and PPI, NPSR and NPSI to be equal, however a closer look at Figure 2.3 will show why this is not so. Let us consider PPR and PPI, for tracks 1, 2 and 3. The probability of retrieving a record from PDA (PPR) is the ratio of the number of records in PDA to the number of records in PDA plus the number of records in overflow.

For tracks 1, 2 and 3

$$\text{PPR} = \frac{12 + 6}{12} = 0.67$$

On the other hand, the probability of inserting a record into the PDA (PPI) is the ratio of those unused key values mapped into the PDA to all the unused key values mapped into this area. Assuming the highest key value on the previous cylinder was 1084, then the total number of key values mapped onto tracks 1, 2 and 3 is

$$1413 - 1084 = 329$$

then the total number of unused key values = 329 – 18 = 311.

The number of key values mapped into the PDA is

$$1236 - 1084 + 1304 - 1280 + 1371 - 1304 = 243$$

number of unused key values in PDA = 243 – 12 = 231

$$\therefore \text{PPI} = \frac{231}{311} = 0.74$$

The terms in part (c) must be found from the use of a simulation model because the complex movement of records within a file which is having insertions and deletions passed against it makes analytical methods intractible.

Figure 5.1 shows the flow of data through the design stages.

5.1 Father-Son Update

Assume

(i) single buffered I/O
(ii) no records in overflow:. no access to indices required
(iii) PDA starts on beginning of a new track
(iv) each file is on a dedicated device
(v) processing time/block < \frac{\text{PBPT} - 1}{\text{PBPT}} \text{ revolution} if \text{PBPT} < 1

or < 1 if \text{PBPT} = 1

File Read Time:

time per cylinder (disk revolutions)
The Design of Indexed Sequential Files

Figure 5.1. Parametric approach to indexed sequential file design.

\[ \text{File Read Time:} \]
\[ = \text{track to track head movement time} + \text{latency time} + \text{time to read all blocks} \]
\[ = \text{SMIN} + 0.5 + \text{NPT} \times \left( \text{PBPT} \times \left(1 + \frac{1}{\text{IBPT}} \right) \right) \]
\[ = \text{SMIN} + 0.5 + \text{NPT} \times (\text{PBPT} + 1) \]

File Write Time:
\[ \text{time per cylinder (disk revolutions)} \]
\[ = \text{seek time from cylinder index to this cylinder} + \text{latency time} + \text{time to write all blocks} + \text{latency time} + \text{time to write track index} + \text{seek time from this cylinder to cylinder index} + \text{latency time} + \text{time to write one cylinder index block} \]
\[ = \text{SAVE} + 0.5 + \text{NPT} \times (\text{PBPT} + 1) + 1 + \text{FT} \]
\[ + \text{SAVE} + 0.5 + \frac{1}{\text{IBPT}} \]
\[ = 2 \times \text{SAVE} + 2 + \text{NPT} \times (\text{PBPT} + 1) + \text{FT} + \frac{1}{\text{IBPT}} \]

Note that the file processing elapsed time will depend upon the overlap between the file read, file write, and block CPU processing times.

The file read time can be adjusted to account for records in overflow if the distribution of those records is known. Again, this will usually require the use of a simulation model to determine the degree of overflow utilization before reorganization.

5.2 Insitu Sequential Update
This type of an update will not be analysed specifically, since it is a modified form of an insitu random update. The essential difference between the two is that the amount of index accessing (especially to the higher levels of indices) is reduced in the sequential case, however the magnitude of this reduction is heavily dependent on the transaction frequency and distribution. Separate calculations would be necessary if the transaction density was sufficiently high to involve multiple transactions being directed to one PDA track or overflow chain. Under these circumstances, mean search time per transaction would be reduced.

5.3 Random Update
Assume:
(i) single I/O buffer for each of PDA and overflow areas
(ii) PDA starts at beginning of a new track

Timing expression shall be developed as the mean time per record processed, excluding accesses to higher level (above track index) indices. Conventional timing analysis (Montgomery (1971), Van Well Groeneveld et al (1971))
allow calculation of the time required for higher level index accesses. All expressions are in terms of disk rotations.

1. **Record Retrieval**
   (i) If the record is in PDA (probability = PPR) then time to retrieve consists of,
   \[ \text{latency + track index scan + latency + PDA scan} = (0.5) + (FI/2) + (1.0 - FI/2) + NPSR \times FP \]
   \[ = 1.5 + NPSR \times FP \]
   (ii) If the record is in cylinder overflow (probability = POR = 1.0 - PPR) then time to retrieve consists of,
   \[ \text{latency + track index scan + overflow scan} = (0.5) + (FI/2) + NOSR \times (0.5 + FO) \]

2. **Record Update**
   = Retrieval time + 1 revolution

3. **Record Deletion**
   = Update time
   = Retrieval time + 1 revolution

4. **Record Insertion**
   (i) If the record is inserted in the PDA (probability = PPI) then the time to insert consists of,
   \[ \text{latency + track index scan + latency + PDA scan + ‘bumping’ + (possibly) an overlap insertion and track index re-write} \]
   \[ = (0.5) + (FI/2) + (1.0 - FI/2) + (NPSI \times FP + 1.0) + (PBPT - NPSI) \times (1.0 + FP + 1.0) + (1.0 - PB) \times (0.5 + FO + 0.5 + FI/2) \]
   \[ = 2.5 + PBPT \times (2.0 + FP) - 2.0 \times NPSI + (1.0 - PB) \times (1.0 + FO + FI/2) \]
   (ii) If the record is inserted directly at the beginning of a chain (probability = POFI) then the time to insert consists of,
   \[ \text{latency + track index scan + overflow read + overflow write + track index re-write} \]
   \[ = (0.5) + (FI/2) + (0.5 + FO) + (0.5 + FO) + (0.5 + FI/2) \]
   \[ = 2.0 + FI + 2.0 \times FO \]
   (iii) If the record is inserted in the middle or end of an overflow chain (probability = POMI) then the time to insert consists of,
   \[ \text{latency + track index scan + overflow scan + overflow write (new) + overflow read and write (previous record in chain) + COCR re-write} \]
   \[ = (0.5) + (FI/2) + NOSI \times (0.5 + FO) + (0.5 + FO) + (0.5 + FO + 1.0) + (0.5) \]
   \[ = 3.0 + FI/2 + 2.0 \times FO + NOSI \times (0.5 + FO) \]

So, for a file which has the following transactions passed against it:

**NI** - number of insertions
**ND** - number of deletions
**NU** - number of updates
**NR** - number of retrievals

the time to process the file (at the cylinder level) is given by the expression,

\[ (NU + ND + NR) \times (PPR \times (1.5 + NPSR \times FP) + POR \times (0.5 + FI/2 + NOSR \times (0.5 + FO))) \]
\[ + (NU + ND) \]
\[ + NI \times (PPI \times (2.5 + PBPT \times (2.0 + FP) - 2.0 \times NPSI + (1.0 - PB) \times (1.0 + FI/2 + FO) + \]
\[ POFI \times (2.0 + FI + 2.0 \times FO) + POMI \times (3.0 + FI/2 + 2.0 \times FO + NOSI \times (0.5 + FO))) \]

To estimate the time to process an indexed sequential file using an insitu update it is necessary to:
1. calculate the time spent in accessing higher level indices and the associated head movement time
2. use a simulation model to extract the relative frequency of transactions being directed to various parts of the file and the mean scan lengths in those areas of the file
3. apply the simulated figures to the above formulae
An example calculation is given in Appendix 2.

6. **INDEX PLACEMENT**

This particular aspect of the file design has been investigated by Lum, Senko et al (1968, 1969) and the results show that the location of the cylinder index, in particular, has a marked effect on the average time to access a random record.

For a file which is to be optimized with respect to performance under random processing, careful attention to the index placement will give the largest possible improvement in performance under the existing ISAM software.

Obviously, a core-resident cylinder index will give the best possible performance, however this is a luxury few organizations can afford, especially if the file is of reasonable size. For example, a file resident on one disk device of 200 data cylinders, with a key length of 10 bytes and an address length of 10 bytes, requires 200 \times (10 + 10) = 4000 bytes of cylinder index.

If the cylinder index must reside on disk, then in all cases placing it on a dedicated device separate from the...
The Design of Indexed Sequential Files

rest of the file gives the best performance, between 12% and 46% improvement over placement on the same device. If this is not possible, then the optimum placement of the cylinder index depends upon

1. the file size (in terms of disk packs), and
2. the transaction distribution over the file space.

Figure 6.1 summarizes some experimental results by Lum, Senko et al (1969).

In general, for a file of 1 pack or less, the cylinder index should be placed in the middle of the peak of transaction activity. For multiple pack files, the index should be placed at the peak within the pack of lowest activity.

The use of a core resident master index, can also provide significant improvements in performance, especially for large files, with a correspondingly large cylinder index.

7. FACTORS INFLUENCING PERFORMANCE

Many factors influence the performance of an indexed sequential file. The effects these factors can have may be determined in one of the following manners

1. analytic analysis of some trial designs
2. simulated operation of trial designs, or
3. use rough techniques gleaned from simulation studies.

Of these methods, the first is currently too complex, the second requires a simulation model – an overhead which can be extremely profitable, especially if software modifications are planned.

The following discussion outlines some ‘rough guidelines’ formulated from our simulation studies.

Figures 7.2 and 7.3 show performance times derived from simulation studies for 6 file designs; these designs have the physical attributes outlined in Figure 7.1.

We shall use these 6 designs and associated graphs in the following discussion regarding the effects of block size and percentage overflow on file performance. The times shown are those derived for activity at the cylinder level and do not include access to or head movement associated with probing the cylinder index. The effects of changes in the file size are not considered in this comparative analysis.

To reiterate, in terms of the magnitude of the possible improvement/degradation of performance, the location of the cylinder index is the single most important design decision. Next, the designer should concentrate on the record layout at the cylinder level.

We shall assume the record data length and key length are predetermined.

7.1 Block Size

Assuming sufficient core store for I/O buffers, the most economical choice of block length in terms of space is one track since this gives minimal waste space for count areas, gaps etc.

However, a block of one track means that each access into the PDA will require reading a full track, whereas if there were n blocks per track (n, say > 5) then the average number of blocks scanned per PDA access is approximately n/2, i.e. only half a track. Therefore, as Figure 7.2 shows, long blocks give generally poorer retrieval times, however this effect diminishes as more transactions are directed to overflow.

Exactly the opposite trend is true for inserting records. This is because PDA insertion involves ‘bumping’ records along the track, and the more blocks, the more bumping. (See Figure 7.3).

The designer is left with the problem of balancing the relative merits of space utilization and insertion time against retrieval time and available core store.

<table>
<thead>
<tr>
<th>FILE DESIGN</th>
<th>RECORD DATA SIZE (BYTES)</th>
<th>RECORD KEY SIZE (BYTES)</th>
<th>RECORDS/BLOCK</th>
<th>BLOCKS/TRACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100</td>
<td>5</td>
<td>33</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>100</td>
<td>20</td>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>500</td>
<td>5</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>500</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>500</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>F</td>
<td>500</td>
<td>5</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 7.1. Physical parameters for 6 sample file designs.

Figure 7.2. Time to retrieve 100 records v. fraction of file in overflow.
7.2 Percentage of the File in Cylinder Overflow
At the design stage, a decision must be made regarding how much overflow space is required to accommodate the expected transaction distribution and frequency of re-organizations. This decision implies an estimate of the point at which the percentage of the file held in overflow is sufficiently high to render the processing of the file inefficient.

We shall concentrate on cylinder overflow, since if records start overflowing into the general overflow area, processing will quickly slow down due to the extra head movement associated with accesses into the general overflow area.

Figures 7.2 and 7.3 show two general trends.

1. Retrieval (and hence deletion and update) performance declines with an increase in the fraction of the file held in cylinder overflow. This is because PDA scanning is quicker than following overflow chains of more than 1 or 2 records. The effect is more marked with long records, because the overflow read time increases, but the PDA scan time is unaltered.

2. Insertion performance improves with increased cylinder overflow. This is because, re-linking overflow is generally less time consuming than the 'bumping' process required for a PDA insert. As would be expected, this improvement is most significant in cases where the number of PDA blocks per track is large (and hence the 'bumping' time longer).

APPENDIX 1
Improvements to ISAM and Other Considerations
Simulation studies by McDonell (1972) and work done by Coyle (1971) have revealed a number of areas in which the ISAM software could be upgraded.

Improved space utilization during file processing could be achieved by implementing the following:

1. Reducing the incidence of records being 'bumped' into overflow from a PDA insert. This would result in improved retrieval time and slowing down the movement of the track index entries, i.e. the fewer records are 'bumped', the less the track index entries must be altered, the more transactions are directed into the PDA instead of overflow. Possible ways of implementing this are as follows:
   (i) allow imbedded overflow space in the PDA at loading time to handle subsequent insertions
   (ii) use 'intelligent bumping', i.e. if a deleted record is found in a block through which records are being bumped, then the deleted record should be used and the bumping process terminates on that block
   (iii) set the PDA track index entries to be 'the last bumped key value minus one', rather than 'the next highest key on the track', after a record is 'bumped' into overflow.

2. Index Scanning Algorithms should be converted to 'binary chop' procedures to speed up large index block searches.

3. Re-use cylinder (and in fact all) overflow spaces after deletion. This procedure could be easily

Figure 7.3. Time to add 100 records v. percentage of file in overflow.

NOTE: Asumption behind these simulated results is that the 100 additions do not make a significant difference in the file size.
APPENDIX 2
Sample Calculations

1. Space Allocation

The file has the following characteristics:
- record data size (bytes) (RDL) = 764
- record key size (bytes) (KL) = 12
- number of records (NREC) = 3500

The expected transaction distribution:
- retrievals (NR) = 8000
- updates (NU) = 1500
- deletions (ND) = 750
- insertions (NI) = 750

We shall assume it is the processing time for the total of all transactions, not just one transaction type which should ideally be minimized. Therefore, since additions are clearly in the minority, small blocks should be selected to minimize the time for the updates, deletions and retrievals to the detriment of the addition performance.

The file will be stored on an IBM 2311 disk device (the space allocation constants may be found in the publication IBM (1966)).

Therefore, the following equations apply:

\[ TC = 3625 \text{ bytes} \]
\[ BO = 811 \text{ bytes} \]
\[ LBO = 20 \text{ bytes} \]
\[ VG = \frac{537}{512} \]

So, the following relationship connects blocks per track (BPT), block data length (DL) and record key length (KL):

\[ BPT = \left\lfloor \frac{3625 - 20 - (KL + DL)}{81 + 537 (KL + DL)} \right\rfloor + 1 \]

Core store considerations set the maximum allowable block size at 4095 bytes, however the maximum track capacity is 3625 bytes.

(i) PDA blocks/track (PBPT)
- Choose records per block = 1
- block data length (BDL) = 764

\[ PBPT = \left\lfloor \frac{3625 - 20 - 12 - 764}{81 + 537 (12 + 764)} \right\rfloor + 1 \]
\[ = \left\lfloor \frac{2829}{81 + 813} \right\rfloor + 1 \]
\[ = 4 \]

(ii) Overflow blocks (records)/track (OBPT)
- overflow block data length (OBDL) = 764 + 10 = 774

\[ OBPT = \left\lfloor \frac{3625 - 20 - 12 - 774}{81 + 537 (12 + 774)} \right\rfloor + 1 \]
\[ = \left\lfloor \frac{3503}{81 + 905} \right\rfloor + 1 \]
\[ = 3 \]

(iii) Index blocks/track (IBPT)
- index block data length (IBDL) = 10

\[ IBPT = \left[ \frac{3625 - 20 - 12 - 10}{81 + 537 (12 + 10)} \right] + 1 \]
\[ = \left[ \frac{3583}{104} \right] + 1 \]
\[ = 35 \]

Since the 2311 device has 10 tracks per cylinder, the maximum number of track index blocks would be \( 2 \times 10 + 1 = 21 \). Therefore the track index will occupy only 1 track.

(iv) Iterative Solution
- Choose PDA tracks/cylinder (NPT) to be 8.
- Since NT = 10,
  \[ NOT = NT - NPT - 1 \]
  \[ = 10 - 9 \]
  \[ = 1 \]
- number of PDA records/cylinder (PRPC)
  \[ PRPC = NPT \times PBPT \times RPB \]
  \[ = 8 \times 4 \times 1 \]
  \[ = 32 \]
- number of overflow records/cylinder (ORPC)
  \[ ORPC = NOT \times ORPT \]
  \[ = 1 \times 4 \]
- file size (NC)
  \[ NC = \frac{NREC}{\text{PRPC}} \]
  \[ = \frac{3500}{32} \]
  \[ = 110 \]
- mean number of additions/cylinder (MAPC)
  \[ MAPC = \frac{NREC}{\text{NC}} \]
  \[ = \frac{750 \times 32}{3500} \]
  \[ \approx 7 \]

But we have only allotted 4 overflow record spaces per cylinder, therefore try NPT = 7
- NOT = 9 - 7
  \[ = 2 \]
- PRPC = 7 \times 4 \times 1
  \[ = 28 \]
- ORPC = 2 \times 4
  \[ = 8 \]
- NC = \[ \frac{3500}{28} \]
  \[ = 125 \]
- MAPC = \[ \frac{750 \times 28}{3500} \]
  \[ \approx 6 \]

The percentage of additions resulting in a record being added to overflow, by the previously outlined 'rule of thumb' will be about 90%. Therefore the 8 cylinder overflow spaces allocated should provide sufficient overflow space in most cylinders. However at least one cylinder of general overflow must be allocated.

2. Random Update Time

(i) Cylinder index access time
- Since the file occupies 125 cylinders, the probability of 2 consecutive random transactions applying to the same cylinder is 1/125, therefore we shall assume a probe
is required to the cylinder index for each transaction.

If the transactions are distributed randomly across the file, then the cylinder index should be located in the middle of the file. Hence the average head movement between cylinder index and data cylinder is across \( \frac{125}{2} = 62.5 \) cylinders. This corresponds (IBM (1966)) to 67 msec.

Cylinder index contains 125 entries and therefore spans \( 125/IBPT = 125/35 = 3.58 \) tracks.

To this point in time, we have been unable to verify these calculations against actual run times, however the results produced by this simulation model and the associated calculations show a very high correlation to the results produced by Lum et al. (1969) and those results were bench mark tested against actual run times.

It must be stressed that there is an additional processing overhead, namely re-organizing the file after all these transactions have been processed. If the file was not re-organized, the percentage of records in general overflow would increase dramatically on the next run, as would the file processing time. By applying the formulae of Section 5.1, it can be shown that this re-organization would take about 2.5 minutes.

To this point in time, we have been unable to verify these calculations against actual run times, however the results produced by this simulation model and the associated calculations show a very high correlation to the results produced by Lum et al. (1969) and those results were bench mark tested against actual run times.

**REFERENCES**


**HONEYWELL (1968):** “Series 200, Mod I (MSR), Data Management Subsystem”, Section II.


**SPECIAL INTEREST GROUP**

The Australian Computer Society Incorporated has formed a Special Interest Group on Simulation.

The principal aim of this group is to encourage the interchange of information on all aspects of the computer solution of problems expressible in the form of differential or difference equations, especially in those areas where the mode of solution is intended to provide insight into the nature of the problem as well as to yield particular solution (simulation).

Mr. Leslie G. Kemeny of the School of Nuclear Engineering of the University of New South Wales, has been appointed as convenor of this group.
A General Validation Procedure for Computer Simulation Models

By Peter Gilmour*

This paper describes a generalized procedure for testing the validity of computer simulation experiments. Before this procedure can be applied the experimental results must satisfy the problem-dependent conditions of face validity testing. Then the general procedure, or a subset of it if cost or computational restrictions exist, is applied. An index of validity provides a basis for intramodel analysis and inter-model comparison.

1. INTRODUCTION

Validation of the results of a computer simulation model is a process which increases the confidence of the user of the model in the ability of the model to accurately reflect the real world behaviour under examination.

Whether this process can be generalized is a question of recent interest. Naylor and Finger (1967) propose a multistage validation procedure which can be applied generally to computer simulation models of economic systems. But Van Horn (1971) claims that validation is problem-dependent and that a generalized procedure does not exist.

The validation procedure is considered to comprise both a specific component and a general component. Design validity concerns the consistency of the basic underlying processes of the model with the situation under examination. This is the specific component. The general component, output validity, examines and establishes the requisite quality of the model's endogenous data streams.

Design validity is established when the operating managers who will eventually use the model accept the general quality of the results produced. These managers must examine the values and variability of all the endogenous data streams generated. Is the nature of these output streams acceptable? This procedure, although elementary in nature, is important as cost-benefit considerations limit the more comprehensive output validity testing to the several key variables. Management must also examine the nature of the assumptions embodied in the model. Do these assumptions agree with known facts? Is the model internally consistent? Does the model “make sense”? This approach is sometimes known as establishing the model's face validity. A more formal variation of this type of testing is the Turing test (Turing 1959). If a person knowledgeable with the processes to be modelled cannot distinguish the model from the real system when provided with responses from both, then the model is realistic.

Output validity is established using a generalized procedure employing a set of germane statistical tests. These tests are applied to:

1. Establish the stability or viability of the model over an extended period of time. (Stability)
2. Measure the extent to which the model is an accurate representation of the real system. (Predictive ability)
3. Perform sensitivity analysis on the nature of the central assumptions of the model. (Sensitivity of major assumptions)

Gross malfunctions of a particular model are usually discovered by analysis for face validity or design validity. Once the model has satisfied these criteria, the more general and sophisticated procedures for establishing output validity are applied. So in the total validation procedure the specific component (design validity) plays a minor role to the general component (output validity). This is logically consistent if one believes, as Friedman (1953), that the validity of a theory, or model, is not based on the realism of its assumptions (complete “realism” is unattainable), but on the accuracy of its predictions.

2. A COMPUTER SIMULATION EXPERIMENT

The nature of this validation procedure is best examined when it is used to analyze the results of an actual computer simulation experiment. The procedure is applied to a dynamic simulation model for planning physical distribution systems. (Bowersox et al., 1972; Gilmour, 1972). In this model the five basic components of an integrated physical distribution system (the fixed facility network, transport capability, inventory allocation, communication, and unitization) are evaluated at the following three stages in the channel structure:

1. The manufacturing control centre which produces a partial product line and distributes these products from the adjoining replenishment centre.
2. The distribution centre which provides a product selection at a location from which customer service requirements can be satisfied.
3. The demand unit which is an individual customer’s demand or the agglomeration of several customers’ demands.

The model itself is constructed in three main parts: The Data Support Subsystem, the four subsystems which comprise the actual operating model (the Demand and Environment Subsystem, the Operations Subsystem, the Measurement Subsystem, and the

*Economics Department, Monash University.

Monitor and Control Subsystem), and the Report Generator Subsystem. This structure is shown in Figure 1.

The Data Support Subsystem generates the input tape for the model. Contained on this tape are the constant exogenous variables for a particular experiment using the model and also the amount and timing over the ten year planning horizon of changes in controllable variables. The controllable variables are order characteristics, product mix, new products, customer mix, facility network, inventory policy, transportation, communications and unitization.

The second main segment of the model contains a mathematical representation (difference equations) of demand generation and allocation, the driver of the model, and the five elements of the physical distribution system: transportation, inventory control, facility location, unitization and communications.

The final segment of the model is the Report Generator Subsystem which organizes the output data of the model into management reports.

The main aspect of the model which sets it apart from previous studies is the consideration of both the spatial and temporal dimensions of the physical distribution system in one model. This provides a means to analyze cost and service trade-offs between all elements of the physical distribution system caused by any given sequence of decision made over a long-range planning horizon.

3. VALIDATION OF THE RESULTS OF THE COMPUTER SIMULATION EXPERIMENT

Design Validity

Lengthy consultation with management established that the performance of all endogenous variables were within reasonable limits. For a ten year simulation of the operation of the firm’s physical distribution system, with no externally introduced changes to the system, the information generated appeared reasonable to corporate management.

Output Validity

Many statistical and graphical techniques have been proposed and used in an attempt to validate the output of computer simulation models (Cyert, 1966; Naylor and Finger, 1967). Table 1 shows the type of testing carried out for each of the three types of output validity procedure. A reference is given in the table to a complete description of each test rather than to include such detail in the body of this paper. Four statistical tests (sequential analysis, multiple ranking, the Kolmogorov-Smirnov test and response surface analysis) are not used. This is a function of the particular data requirement and structural formulation of the model and not a deficiency in the individual test. But even without these tests the total computation required for the output validation procedure is considerable, and this is only for the key variables. There becomes a point where the expense of additional com-

---

**Figure 1: Physical distribution model (Bowersox et al, 1972)**

---

A General Validation Procedure for Computer Simulation Models

TABLE 1. Use of Statistical Techniques

<table>
<thead>
<tr>
<th>Statistical Techniques</th>
<th>Stability</th>
<th>Predictive Ability</th>
<th>Sensitivity of Major Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphical Analysis (Yule &amp; Kendall, 1950)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sequential Analysis (Sasser et al, 1970)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis of Variance (Naylor et al, 1967)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple Comparison (Dunnett, 1955)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple Ranking (Naylor et al, 1967)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The F Test (Naylor et al, 1967)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correlation Analysis (Yule &amp; Kendall, 1950)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Regression Analysis (Draper &amp; Smith, 1966)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Kolmogorov-Smirnov Test (Kraft &amp; Vaneeden, 1958)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Chi-square Test (Gilmour, 1971)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Theil's Inequality Co-efficient (Theil, 1966)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Spectral Analysis (Fishman &amp; Kiviat, 1965)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Factor Analysis (Harmon, 1960)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response Surface Analysis (Box, 1954)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results of each statistical test do not provide equally valuable information on the validity of the model. Inherent in several of these techniques are stringent assumptions (Table 2). It is reasonable to weight the value of the information provided by a statistical test by the inverse of the number of assumptions upon which the test is based and which do not necessarily hold for simulation modelling. The results of each statistical test, the assumptions violated by the technique and the subsequent weight are contained in Table 3. From the weights and results an index of validity for each of the three classes of output validity is calculated, viz.:

\[
\text{Index of validity} = \frac{\sum_{i=1}^{n} \left[ \left( \frac{\text{percentage of favourable results}}{\text{number of assumptions violated + 1}} \right) \left( \frac{1}{\text{number of assumptions violated + 1}} \right) \right]}{n}
\]

The index of validity for the model (for output validity testing should not commence until successful design validity testing is completed) is the mean of the three sub-indices (Table 3).

This index is clearly a heuristic tool for the simulation experimenter. Validation is a relative concept and the index should be used in this context. The relative values of the indices for the three classes of output validity indicate whether additional developmental effort is needed and where this effort should be applied. Changes in the model validity index provide additional information to the decision maker about the desirability of any proposed alterations to his simulation model. The absolute value of the model index

1. The results for the model's predictive ability are unreliable because of the unavailability of adequate historical data. The actual data stream provided was too short (103 time units) and the time increment of one day introduced inordinate variability especially for products with low annual sales volume.

A General Validation Procedure for Computer Simulation Models

TABLE 2. Assumptions of Statistical Techniques

<table>
<thead>
<tr>
<th></th>
<th>Equality of Variance</th>
<th>Normality</th>
<th>Number of Assumptions Violated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphical Analysis</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Analysis of Variance</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Multiple Comparison</td>
<td>X</td>
<td>X</td>
<td>3</td>
</tr>
<tr>
<td>The F Test</td>
<td>X</td>
<td>X</td>
<td>3</td>
</tr>
<tr>
<td>Correlation Analysis</td>
<td>(Number of observations must be large)</td>
<td>X</td>
<td>2</td>
</tr>
<tr>
<td>Regression Analysis</td>
<td>(Number of observations must be large)</td>
<td>X</td>
<td>2</td>
</tr>
<tr>
<td>The Chi-square Test</td>
<td>(Number of observations must be large)</td>
<td>(Theoretical cell frequency minimum of 10)</td>
<td>2</td>
</tr>
<tr>
<td>Theil’s Inequality Coefficient</td>
<td>(Does not discriminate between signs)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Spectral Analysis</td>
<td>(Stationary time series)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Factor Analysis</td>
<td>(Observed variables must be linear functions of factor variables)</td>
<td>X</td>
<td>3</td>
</tr>
</tbody>
</table>

*Assigned by Judgement

TABLE 3. Indices of Validity

<table>
<thead>
<tr>
<th>Number of Assumptions Violated</th>
<th>Weight</th>
<th>Stability</th>
<th>Predictive Ability</th>
<th>Sensitivity of Major Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphical Analysis</td>
<td>4</td>
<td>0.20</td>
<td>100.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Analysis of Variance</td>
<td>3</td>
<td>0.25</td>
<td>100.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Multiple Comparison</td>
<td>3</td>
<td>0.25</td>
<td>0.0</td>
<td>88.0</td>
</tr>
<tr>
<td>The F Test</td>
<td>2</td>
<td>0.33</td>
<td>50.0</td>
<td>75.0</td>
</tr>
<tr>
<td>Correlation Analysis</td>
<td>2</td>
<td>0.33</td>
<td>0.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Regression Analysis</td>
<td>2</td>
<td>0.33</td>
<td>50.0</td>
<td>100.0</td>
</tr>
<tr>
<td>The Chi-square Test</td>
<td>2</td>
<td>0.33</td>
<td>88.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Theil’s Inequality Coefficient</td>
<td>1</td>
<td>0.50</td>
<td>50.0</td>
<td>75.0</td>
</tr>
<tr>
<td>Spectral Analysis</td>
<td>0</td>
<td>1.00</td>
<td>100.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Factor Analysis</td>
<td>3</td>
<td>0.25</td>
<td>13.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

\[ I = 0.877 \quad I^* = 0.466 \quad I = 0.867 \]

4. SUMMARY

A general validation procedure for computer simulation models of industrial systems is proposed. Not all statistical tests must be used for every model under consideration. Selection of statistical techniques to be used should begin with the most robust (i.e. violates fewest assumptions) and proceed to the limits imposed by the total time and money commitment for any particular model. The index gives the experimenter a method to relate the quality of his results to his and others' past experience.

REFERENCES


A General Validation Procedure for Computer Simulation Models

The Standards Association of Australia

ISO/TC97/SC6 (Australia-7) 682
Title: Australian Contribution on Communication Systems architecture.
Date: October, 1972.
No. of Pages: 5.

Abstract: The document is in response to point 3 of Resolution No. 5 passed by ISO/TC97/SC6 at its Berlin (1972) meeting. The document comments on the architecture of communication systems and suggests areas where international standards should be developed by ISO. A possible "network control" approach is outlined which may be applied equally to the case of "conventional message switching" as well as data transfer between data devices.

SAA Action: For Information Only.

ISO/TC97/SC6 (CCITT-13) 710
Title: Final Report on the Work of Joint Working Party on "New Data Networks".
Date: December, 1972.
No. of Pages: 12.

Abstract: This is a CCITT document and is the final report of the work of the CCITT Joint Working Party studying new data networks. This report contains the final editing for the previously submitted preliminary report. In addition, a new question is proposed for study during the 1973/76 period. The question deals with security in data networks and concerns the technical and operational means which should be provided in new data networks to fulfill an administration's responsibility for security of a communication.

SAA Action: For Information Only.

ISO/TC97/SC6 (U.K.-53) 717
Title: Data Communications.
No. of Pages: 8.

Abstract: This contribution consists of a comparison of two proposals (SC6 N 568 (U.K.) and SC6 N 636 (U.S.A.)) The comparison addresses the following features which affect facilities available to the data terminal equipment: (i) bit synchronism (ii) character synchronism (iii) bit sequence independence (iv) network control signalling (v) data link control (vi) data signalling at rates less than maximum of the class (vii) call collision resolution (viii) not ready for local mode condition

SAA Action: For Information Only.

Draft Standard for Alphanumeric Keyboard

(September 1973)

The Standards Association is seeking comment on a draft Australian standard for alphanumeric keyboards implementing the Australian standard 7-bit coded character set, issued as DR 73102.

This draft specifies layouts for the alphanumeric area of keyboards implementing the 95 graphic characters set specified in AS X1. Its main aim is to facilitate international information processing interchange. The draft provides layouts for keyboards having 47 or 48 keys and gives rules for deriving layouts for keyboards with fewer keys. The draft specifies the pairing of the characters and their allocation to various keys. The pairing rule follows the principle of simple bit inversion, but full bit pairing is possible only with the 48 key layout. The layouts given are concerned only with specifying nominal relative positions of keys. Physical factors and positions of function keys are not given. The draft is derived from and is fully compatible with ISO DIS 2530 issued by the International Organization for Standardization (ISO).

Copies of DR 73102 may be obtained without charge from the various offices of the Standards Association of Australia in all capital cities and at Newcastle.

The Association now invites constructive comment on the provisions of this draft from persons or organizations experienced in the field of information processing and suppliers of equipment incorporating keyboards. Such comment should reach the Headquarters of the Association, 80-86 Arthur Street, North Sydney, N.S.W. 2060 or any branch office before 30 November, 1973.

A Survey of Automatic Flowchart Generators

By R.P. Watkins*

The value of flowcharts for documentation has led to many attempts to generate them automatically from computer programs. This survey examines the methods by which flowcharts can be constructed and considers whether the existing systems adequately satisfy the requirements of documentation.

1. INTRODUCTION

The primary function of flowcharting is to communicate the structure and purpose of algorithms to humans in a form which can be readily comprehended. This documentation role is mainly facilitated by the pictorial representation of steps and their inter-relation. The appeal to vision enables the overall form of an algorithm to be scanned very rapidly, and "the desirability of concise, accurate and complete flowcharts as an integral part of computer program documentation is a well established fact" (Anderson, 1965).

If flowcharts are to be used for documentation, their three most significant features must be:

(a) pictorial layout in two dimensions;
(b) segmentation into conveniently sized sections, or pages; and
(c) the ability to represent the algorithm (or program) at a number of levels of detail.

(The arguments supporting this view may be found in Watkins, 1972, 1973.)

The problem of providing standardized, up-to-date documentation has led to many attempts to automatically generate flowcharts by computer analysis of programs. The purpose of this survey is to discuss the design principles of a selection of these automatic flowchart generators in relation to the above criteria. In doing so, it is shown that these criteria provide an invaluable aid to the classification and comparison of different generators. The main emphasis is on the layout of symbols and their interconnection because, as will become apparent, relatively little work has been done in either of the other two areas.

This work is part of a Ph.D. thesis submitted at Melbourne University (Watkins, 1972), and was supported by a Melbourne University Research Grant.

2. ONE DIMENSIONAL GENERATORS

"In their details the packages differ on a number of features that are not obvious upon a casual inspection" Chapin (1971).

The most common type of flowchart generator is that which produces one-dimensional output, where all the flowchart symbols appear in a single column, one underneath the other; see, for example, Figure 1, reproduced from Scott (1958).

The paper by Scott (1958) is the first to describe this type of generator. Here, as in later systems, the source program to be flowcharted is listed in input sequence with no manipulation of the statement order. Statements are isolated, either in groups or singly, by surrounding them by frames (blocks) and connecting lines are added to indicate non-sequential transfers of control. The output is completed by annotating the figure with the source program statements; hence it is described as a 'flowchart listing'.

Due consideration must be given to the programming philosophy of this generator for, although very simple, it provides the foundation on which many of

![Figure 1: Flowchart listing (from Scott, 1958)](image-url)

*Department of Information Science, University of Melbourne, Victoria. Present address: Department of Mathematics and Computer Science, R.M.I.T., Melbourne, Victoria.
the later systems are built. The input code is split up into groups, each of which is surrounded by a frame. Simultaneously transfer points (labelled statements) and the coding which causes transfer of control are located for use in drawing the non-sequential transfer lines. Lines representing forward transfers are drawn on the right of the frames, backward transfers to the left. The transfer lines are assigned to 'channels' on the output medium so that the number of intersections is minimized within the above restriction. The selection of channels involves examining the points at which the flowlines enter and leave the blocks; the position of a point being given by the vertical coordinate of the corresponding statement relative to the top of the flowchart. The terminal points of pairs of flowlines are tested to determine whether:

1. the points of one line lie inside those of the other (the first line then being assigned a channel closer to the frames);
2. the points overlap, in which case an intersection must occur; or
3. the coordinates of one line are both smaller than those of the other line, and the two lines are then independent and can be assigned to the same channel.

The process of annotation involves listing the source coding within the frames and the associated comments to one side. The source language considered by Scott was an assembler, but the method is applicable to high level languages as well. However, in this case the comments, often being separate input records, could not be correlated so closely with the frames.

Scott's paper gives rise to several criticisms. First, in reference to the frames, he states: "Whenever a break in blocks is indicated, one block is terminated and the next started". He gives no rationale for the positioning of the breaks, and they appear to be imposed quite arbitrarily upon the data (see Figure 1). Secondly, the annotation is not of the type one would intuitively expect in a flowchart. It consists of a program listing and is not a description of the process or of the logical steps involved in the computation. Finally, although Scott suggests the possibility of developing a two dimensional picture, like that illustrated in Figure 4 of his paper, he gives no indication of how such a flowchart might perhaps be produced.

Knuth (1963) describes a similar generator which is developed from a consideration of the role of flowcharts in documentation. To begin with, he stresses the need for multilevel documentation. Using the example of compiler documentation, Knuth points out the need for describing the system at a number of levels of detail ranging from simple block diagrams through to full program listings. Note that the essential point is that the purpose of documentation is to provide teaching material. That is, given a documentation, a person who knows nothing at all about the program concerned should be able to gain a full understanding of it in the least possible time and with the minimum assistance. If this is to be achieved one or more parts of the documentation must be at a low level of detail and basically descriptive in nature (see Watkins, 1973). Knuth proposes three levels:

1. Flow outline.
2. Flowchart.
3. Formal language (program listing).

The flow outline, suggested by Gant (1959), is a "step-by-step English language description of the algorithm, where every step is numbered or otherwise named". Knuth suggests coupling flow outlines with flowcharts and removing all annotation from the latter so that the flowchart can become purely a graph of flow. Figure 2, reproduced from Knuth (1963), reveals the effectiveness of this approach. However, the three levels proposed by Knuth provide, in fact, equally detailed descriptions of the algorithm in different notations. Obviously each description emphasizes a different aspect of the source program, but the flowchart and flow outline, for example, do not present gradations of detail as would a flowchart and a block diagram. When discussing documentation, Knuth stated his concept of the three levels as:

(a) General description;
(b) Description of the main steps; and
(c) Detail of individual steps.

Thus there has been a shift in the meaning of the phrase 'levels of detail'.

The techniques employed by Knuth are fundamentally the same as those used by Scott, except that Knuth generates the flowcharts from special statements. These statements provide details of the type of flowchart symbols to be drawn, and the annotation for the symbols and associated flow outline. The programmer splits up the program into logical sections, giving each section a label and a set of sub-labels (one for each flowchart symbol). The labels are punched either on the source program cards or on separate cards, the desired annotation being included after the labels. Non-sequential transfers of control are indicated by tagging the destination label with the symbol # and using the label field on the card to indicate the exit condition. Thus:

A3. T(M) : KEY
GR: IF GREATER, TO # A5
produces a decision box labelled 'A3' with one exit to the box 'A5'. This path is marked with the characters 'GR:' (see Figure 2). Each part of the program is given distinct labelling and appears as a separate 'page' of the flowchart.

The flowchart generator consists of two phases; the first sets up an internal representation of all necessary data and the second produces the output, line by line. Channel assignment for the non-sequential transfer lines is by rule of thumb and no attempt is made to minimize the number of crossovers.

The two methods of Scott and Knuth seem to differ primarily in their approaches to the problem of annotation. In Knuth's system the commentary has to be specially prepared by the programmer; and hence its excellence, or lack thereof, depends on that person's ability. Automatically prepared annotation is rejected for the sake of producing better flowcharts even though this step destroys the tremendous advantage of a fully automatic system. However, a far more significant difference is that Knuth's method does not flowchart the program. It is quite possible, by careless design of the coded comments, to produce a flowchart which in no way represents the original program. The
A Survey of Automatic Flowchart Generators

---IN---

0006:

A1. INITIALIZE

0112:

A2. GET MIDPOINT

0019:

A3. T(M):KEY

0024:

A4. FIX LOWER

0027:

A5. FIX UPPER

---OUT---

---IN---

0013:

A3. T(M):KEY

0024:

A4. FIX UPPER

0027:

A5. FIX UPPER

---OUT---

Figure 2: One dimensional flowchart and flow outline (from Knuth, 1963)

danger of this occurring in a semi-automated system cannot be stressed too strongly.

Several other flowchart generators use essentially similar techniques to produce pictures, but they have different approaches to annotation. Saalbach and Sapovchak (1965), using a simpler generation algorithm, adopt Knuth's use of prepared comments, whereas the Calcomp (1968) FLOWGEN/F generator follows Scott's method. O'Brien and Beckwith (1968) discard annotation altogether because they feel that automatically produced comments are not particularly helpful, and it is better to leave room for the user to fill in the picture as he chooses. Sherman (1966) shows a more sophisticated approach, employing a syntax driven input phase for language recognition which allows the user to select his own type of annotation.

These systems introduce two other changes. They place single statements, rather than a number of statements, in blocks; and, with the exception of O'Brien and Beckwith (1968), they automatically produce the output on a number of 'pages', making it easier to handle the flowcharts. The flowchart is printed until one page is full and then a new page is begun. Transfers from one page to another are indicated by connectors. Sherman (1966) also acknowledges the need for multilevel and two dimensional flowcharting but he suggests no way in which these features can be achieved.

3. TWO DIMENSIONAL GENERATORS

Two dimensional flowcharts are distinguished by their layout from the type discussed in the previous section. Their symbols are laid out in a number of columns with horizontal as well as vertical connecting lines (see, for example, Figure 3). Thus, QUICK-DRAW, although described as a two dimensional flowchart in the User's Manual (N.C.A., no date), is only a moderately good one dimensional system. It is not a 'good' system because a large number of unnecessary connectors are used where flowlines might easily be inserted, and because the flowlines can only be drawn on one side of the columns of symbols. Its output compares unfavourably with that of FLOWTRACE (Sherman, 1966) for example.

A number of attempts have been made to produce two dimensional output. The first of these, by Haibt (1959), provides more insight into the problem of documentation:

"We also felt it was desirable not to attempt to show the whole program as one chart, which for a moderate size program would either present a confusion of detail or be too general to serve the purpose. In order to provide both a good picture of the program or part of it, and a more detailed description of smaller pieces of the program, the Flowcharter produces a series of flowcharts on a number of levels of detail; each part of a chart is shown in more detail on a succeeding chart." (My emphasis.)

This concept of multilevel flowcharting, which has since been restated by Gieszl (1970), Marshall (1970) and Watkins (1972, 1973), is well developed. The aim is to split the program up into six or seven blocks at the most, and flowchart them as a single page. Each block is then sub-divided and flowcharted in the same way, and so on, until single statements are flowcharted. Figure 3, from Haibt (1959), shows a high level flowchart to illustrate this point.

The method of subdivision, which leaves statements in input sequence without significant re-ordering, involves two processes:
A Survey of Automatic Flowchart Generators

(a) The first, the combination routines, iteratively forms regions from groups of sub-regions. The smallest sub-regions are individual instructions and these are combined by looking for groups which form 'string', 'diamond' or 'test' regions. For example, the graphs of Figure 4 represent a string and a diamond. A test region is best defined as a multiple branch section such as the type of coding associated with a Fortran computed GO TO statement. The process of forming regions from sub-regions is continued until no more combinations can occur.

(b) The above step is followed by a division process where the program as a whole is broken up into large blocks by means of 'splitting' and 'unwrapping' routines. The splitting routine seeks program sections which are self-contained in the sense that they have only one entry and one exit point. The unwrapping routine searches for loops and reduces these. These routines never split the groups formed by combination and are only employed if the combination routines cannot reduce the program to the required number of blocks.

The flowcharting is very simple and it is not discussed in detail by Haibt. Having only about six blocks to lay out two dimensionally on a page, it is sufficient to use an ad-hoc method and, if necessary, re-arrange them to get a reasonable layout. For example, the longest group of linearly linked blocks can be quickly found and assigned to a central channel. The remaining blocks, of which there can be no more than three, can be grouped around this main flowpath.

The annotation is somewhat more sophisticated than that of the previous references:

"Another feature of a flowchart is a description of the procedures represented by each box. The Flowchart provides a summary of the machine input and output done in the box and a summary of the computation done in the box, listing the quantities computed and those used in the computation of each of them." (Haibt, 1959)

Here, the annotation, although automatic, is a compromise between the approaches of Scott (1958) and Knuth (1963) with an attempt at providing only the significant information required by the user.

The most important aspect of Haibt's work is the two dimensional output produced. However, this is unsatisfactory in one significant way; namely, the limitation of the number of regions per flowchart page to about six which necessitates artificial splitting of the program. For example, if a program consists of ten distinct and equally important sections they should, if possible, be shown as such on the flowchart, but Haibt's method forces three or more of these sections to be combined. FLOG (Watkins, 1972) puts up to 16 blocks on a page but still has trouble with segmentation. Of course, some artificial grouping will always occur, but the greater the number of blocks per page the less frequently this will happen. In addition to this problem, the very small number of symbols per page means a very large number of pages will be produced for a program of any reasonable size.

Another somewhat similar system has been developed by Krider (1964). The same type of two dimensional picture is built up, but the method of forming blocks is entirely different. Krider defines a linear representation for directed graphs called 'well formed formulae' (see also Berztiss, 1971). By a simple process of re-ordering and examination, the formulae corresponding to a program can be split into blocks in a way similar to that used by Haibt, and then the method is repeated on each block to get successive levels of detail. The paper is concerned with the theory of the techniques
Figure 5: AUTOFLOW two dimensional flowchart
and very little is said about the actual generation of flowcharts. It appears, however, that the method for forming blocks was chosen for convenience rather than quality of flowcharts. The segments produced are reminiscent of those generated by Scott (1958). A more sophisticated analysis of these formulae is used by FLOG (Watkins, 1972), which is discussed in section 5.

By far the most sophisticated two dimensional system, for which documentation is readily available, is AUTOFLOW (Goetz, 1965 and Applied Data Research, 1967). The program is copyright and so, in the absence of primary documentation, this discussion is based on the paper by Goetz (1965) and the information deduced from sample output. The flowchart in Figure 5 displays most of the features of AUTOFLOW. The output is two dimensional with a significant amount of re-ordering of the source program statements. It will accept both prepared and unprepared programs as input but variable depth of flowcharting is not possible. The output consists of four symbol channels per page, each having two transfer channels available for non-sequential transfers of control (making eight altogether).

The analysis process appears to contain the following basic steps, commencing at the start of the program:

(a) Find a block of contiguous statements and place these in a symbol channel producing a one dimensional flowchart of this section of the program. If 'appropriate' (it is difficult to deduce the criteria used) the flowline can be extended into an adjacent channel.

(b) Scan the block from its first statement looking for branch statements:

(i) If the branch refers to an instruction in the block and there is an available transfer channel, insert a connecting line. If a transfer channel is not available, insert a connector symbol.

(ii) For the first branch instruction which refers to a statement outside the block being considered repeat steps (a) and (b), until the page is full, using the block of instructions starting with the statement which is the successor of the selected branch.

Although AUTOFLOW seems to be the best two dimensional generator, often the output does not visually suggest a two dimensional layout. This can be partly understood if AUTOFLOW is regarded as a one dimensional system in which inter-related flowchart segments are placed side-by-side on a single page and connected together. In addition, there is no delineation between different paths through the program segments. An essential property of flowcharts would appear to be that the overall structure of a program should be visible at a glance, and this requires a clear separation of the different paths of flow. In AUTOFLOW no such distinction is made.

The last system to be discussed is that developed by Hain and Hain (1965a, b). This is radically different because it sets up a tree representation of the program and is designed to handle other structures such as PERT networks. This tree representation requires that all flowlines go in the one direction and no loops can occur:

"... in the case of a program containing loops, they have to be specially handled. The starting element of a loop (which is not necessarily unique), is found out and then at the end of the loop repeated (with an interrupted enclosure), so that actually, the flow goes back from this element to the original one with the same name." (From Hain and Hain, 1965b, pp. IV-1, IV-2.)

A sample of the output is shown in Figure 6 (reproduced from Hain and Hain, 1965a). Alternatively, instead of the circles in this diagram square boxes, which may be annotated after the style of Knuth (1963), are provided for program flowcharting. The generator begins by analysing the program structure and breaking all loops. Then the elements (instructions) are sorted to determine the x and y plotting coordinates which minimize the number of intersecting lines. The system also contains routines which allow reduced flowcharts to be produced, representing different levels of detail.

Thus the generator of Hain and Hain represents a radically different approach to the problem of producing two dimensional pictures and enables more complex graphs to be drawn. However, by no means all users would agree with the authors that a tree structure is as good as the conventional form of flowchart. The main criticism is that by breaking paths a lot of the visual impact of the flowchart is lost, but this seems to be its most useful aspect.

Figure 6: Flowchart based on a tree representation (from Hain and Hain, 1965a). A hand drawn flowchart of a program is shown on the left. The same program, with the flowchart drawn automatically in a sequential way, appears on the right. Dotted circles indicate the newly introduced elements which are used to break up the loops.
4. THE COMPARISON OF FLOWCHART GENERATORS

This comparative study has concentrated on the proposed criteria for classifying flowchart generators. A number of other systems have not been included because those described above illustrate the most significant differences in approach, and it has been found that the others are usually re-statements of the same ideas. For example, QUICK-DRAW (N.C.A., no date), FLOWRITE (Harris, 1968) and the techniques discussed by Anderson (1965) and Roberts (1967) fall into this category. Also, none of the non-automatic systems, such as FLOCODER (Morris and Kennedy, 1971), have been considered, being little more than specification writing systems.

Two other comparative studies of flowchart generators are readily available and provide an interesting contrast with this paper. Abrams (1968) tries to compare the 'quality' of different systems, whereas Chapin (1971a, b) is concerned with how the user is to choose a flowcharting package which will suit him.

The study of Abrams (1968) suffers from two drawbacks. Firstly, only three generators are compared, FLOW2, AUTOFLOW and MADFLO. Although this is perfectly understandable (because of the problem of obtaining copies of the programs) it is to be hoped that the comparison can be extended in the future. Secondly, and much more importantly, no distinction is made between the types of generator; one and two dimensional. This classification is crucial because they produce different types of output and the two dimensional generators need much more sophisticated paging and layout algorithms compared with their one dimensional counterparts (for it is almost inevitable that significant re-ordering of the source program statements and the flowchart layout will be required).

In particular, time comparisons between the two types of system are rather meaningless unless this factor is taken into account. As well, qualitative evaluation of the output of generators should largely be restricted to comparisons between systems of the same type.

Of the three generators discussed by Abrams, AUTOFLOW has already been considered. In MADFLO the flowchart symbols occur in basically the same order as the source program statements and there is no paging. It can undoubtedly be called a pseudo two dimensional generator because the output can be prepared by a simple modification of the one dimensional algorithm. Most symbols are placed in a single column with coding from the alternative branches of decision instructions being pushed out to the right. Each of these blocks is flowcharted as one dimensional output with links to other parts of the program indicated by connectors. Two dimensional layout algorithms are not required and the output can be paged in exactly the same way as a one dimensional flowchart. The figures in Abrams (1968) clearly illustrate this technique, which is closely related to that used by AUTOFLOW. The output of FLOW2 is more obviously two dimensional but suffers from similar faults. The paging consists simply of breaking up the completed flowchart into arbitrarily sized blocks. No restrictions on length or width are imposed because the flowlines are extended across page boundaries both horizontally and vertically. Because connectors are not used, the output has to be pasted together before it can be employed efficiently. The layout algorithms, as in MADFLO, are elementary. Alternative paths from decisions are pushed out to the right and flowlines inserted in any available space. This results in a visually cluttered and obscure diagram which can only be improved by employing a re-arrangement procedure to optimize the positions of symbols and lines, or else by resorting to a simpler system like that of MADFLO.

Abrams attempted to get an evaluation of the 'quality' of output by submitting flowcharts produced by each system to a panel, but the results were inconclusive:

"The output of each program was ranked first by some panelists and last by others." (Abrams, 1968, p. 745.)

Insufficient details are given of the criteria which the panelists used in this comparison, but the lack of a decisive result is probably due, in part, to the choice of generators; more interesting ratings may perhaps be gained by including some one dimensional output (from QUICK-DRAW for example).

Chapin (1971b) also fails to distinguish between one and two dimensional generators. This is most clearly borne out by his classification of packages for flowcharting (Figure 1 of Chapin, 1971b which is also in Chapin, 1971a), where QUICK-DRAW is listed as a successor to AUTOFLOW when no real relationship exists. Similarly, to imply that the system of Haibt (1959) is a derivative of Scott (1958) and that Knuth (1963) derives from Haibt involves the failure to recognize the substantial differences between them. The 'family tree' for automatic generators would be greatly enhanced if at least two roots were shown: one for one dimensional systems starting with Scott (1958) and a separate one beginning with Haibt (1959) for two dimensional packages. The failure to make this distinction must inevitably make the evaluation of different generators very much more difficult.

The main purpose of Chapin's article (Chapin, 1971b) is to propose a number of criteria by which flowcharting packages can be compared. These include the type of output produced (even though the dimensional distinction is not made), handling of loops, cross-referencing, availability and cost. This is extended in a later article (Chapin, 1972) in which conformity to the ANSI standards is discussed. It is clear that no single package will satisfy all users, but at the same time there is no critical evaluation of which systems are best suited for particular tasks, and very little information is given about the packages mentioned.

5. FLOG

To conclude this survey a few comments on the Fortran flowchart generator FLOG (Watkins, 1972) are appropriate. From the above discussion it appears that most developments to date have been directed towards producing 'practical' systems. As a consequence ad-hoc techniques and compromises seem to have been necessary, but these have led to relatively poor output from the point of view of acceptability to humans. In contrast to this attitude, FLOG was de-
developed on the basis of purely theoretical considerations, and practical difficulties were initially ignored. A comparison of Figures 7 and 8 with the other output illustrated in this paper seems to vindicate this approach.

The generator is based on the three criteria stated in Section 1; as a consequence it was found necessary to perform a graph-theoretical analysis of the program’s structure. This analysis, which will be described in detail in a later paper, contains the following sections:

(a) Set up a graphical description of the program using the notation of Krider (1964), which is sufficiently compact to enable a Fortran program of up to about 3000 statements to be handled in core as a single entity.

(b) Segment the graph into ‘pages’ corresponding to about 16 Fortran statements. This process extracts sub-pages (see Figures 7 and 8) which, in theory, will lead to multilevel flowcharting.

(c) Plotting each page two dimensionally by means of an ad-hoc algorithm (there being no theoretical planarity algorithm which is adequate). No annotation is included for the same reasons as are put forward by O’Brien and Beckwith (1968).

Although this rigorous method can yield excellent results, it is at present inadequate for a practical flowchart generator. Failures can occur in two ways:

(i) If the segmentation procedure produces pages of more than 16 statements (or symbols), these pages cannot be plotted. Although increasing the permissible number of symbols per page will alleviate the problem, some intractable segments will still occur. This necessitates the addition of a back-up routine to perform an ad-hoc analysis of the segments when the theoretical techniques break down.

(ii) Although some sub-paging is done, this is not sufficiently extensive to warrant being termed multilevel flowcharting. Thus, like all the other systems (including Haibt, 1959), FLOG does...
not satisfy the criteria of documentation proposed by Watkins (1973).

REFERENCES


A Draft Standard for Digital Data Transmission

SEPTEMBER 1973

The Standards Association is seeking comment on a draft Australian standard for digital data transmission, covering extensions to the basic mode control procedures. The draft is issued as DR 73103 and is intended to be Part 5 of the standard on digital data transmission.

A data communication system may be considered as the set of terminal installations and the interconnecting network which permits information to be exchanged. A data link comprises terminal installations connected to the same network operating at the same procedure and in the same code. Data link control procedures are classified by reference to their mode of operations. The basic mode is described in AS 1484, Parts 1 to 4.

This draft extends the basic mode control procedures for data communication system to cover information transfer without code restrictions, the interchange of information messages in a fast conversational manner and describes recovery procedures, abort and interrupt procedures and multiple station selection. It supplements the information given on basic mode control procedures in Part 1 of an Australian standard now being prepared on digital data transmission.

This draft is technically identical with the requirements of three International Standards, published by the International Organization for Standardization (ISO).

Copies of DR 73103 may be obtained without charge from the various offices of the Association in all capital cities and at Newcastle.

The Association now invites constructive comment on the provisions of the draft, from persons or organizations experienced in the field of data communications. Such comment should reach the Headquarters of the Association, 80 Arthur Street, North Sydney, N.S.W. 2060, or any branch office before 30 November, 1973.
### AUSTRALIAN COMPUTER SOCIETY INCORPORATED

#### CONSOLIDATED STATEMENT OF INCOME AND EXPENDITURE

for the year ended 31st December, 1972

<table>
<thead>
<tr>
<th></th>
<th>Canberra</th>
<th>N.S.W.</th>
<th>Queensland</th>
<th>South Australia</th>
<th>Victoria</th>
<th>Western Australia</th>
<th>The Council</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INCOME</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gross subscriptions</td>
<td>6,128</td>
<td>20,700</td>
<td>6,648</td>
<td>4,410</td>
<td>17,291</td>
<td>2,267</td>
<td>—</td>
<td>57,444</td>
</tr>
<tr>
<td>Capitation fees (Council)</td>
<td>504</td>
<td>616</td>
<td>135</td>
<td>42</td>
<td>533</td>
<td>12,219</td>
<td>—</td>
<td>19,330</td>
</tr>
<tr>
<td>Interest receivable</td>
<td>3,857</td>
<td></td>
<td>445</td>
<td>302</td>
<td>7,234</td>
<td></td>
<td>—</td>
<td>24,057</td>
</tr>
<tr>
<td>Other income</td>
<td>—</td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td><strong>TOTAL INCOME</strong></td>
<td>$6,632</td>
<td>$25,173</td>
<td>$7,228</td>
<td>$4,754</td>
<td>$25,058</td>
<td>$2,328</td>
<td>$32,749</td>
<td>$103,922</td>
</tr>
</tbody>
</table>

|                |          |        |            |                 |          |                   |             |        |
| **EXPENDITURE**  |          |        |            |                 |          |                   |             |        |
| General purposes        | 195      | 19,090 | 464        | 221             | 5,319    | 486               | 10,520     | 36,295 |
| Meetings and functions | 674      | 358    | 610        | 695             | 7,226    | 412               | —          | 9,975  |
| Publications (gross cost)| 1,966   | 4,490  | 1,310      | 704             | 3,899    | 161               | 9,792      | 22,322 |
| Capitation fees (branches) | 2,604   | 7,350  | 1,934      | 1,654           | 4,916    | 872               | —          | 19,330 |
| Special activities and awards | 222     | —      | 2,273      | 342             | —        | (25)              | 5,272      | 8,084  |
| **TOTAL EXPENDITURE**    | $5,661   | $31,288| $6,591     | $3,616          | $21,360  | $1,906            | $25,584    | $96,006 |

|                |          |        |            |                 |          |                   |             |        |
| Surplus for year       | 971      | —      | 637        | 1,138           | 3,698    | 422               | 7,165      | 14,031 |
| Deficit for year       | —        | 6,115  | —          | —               | —        | —                 | —          | 6,115  |
| **Surplus/Deficit**    | $6,632   | $25,173| $7,228     | $4,754          | $25,058  | $2,328            | $32,749    | $103,922 |
# AUSTRALIAN COMPUTER SOCIETY INCORPORATED
## CONSOLIDATED BALANCE SHEET—31st DECEMBER, 1972

<table>
<thead>
<tr>
<th>ASSETS</th>
<th>Canberra</th>
<th>N.S.W.</th>
<th>Queensland</th>
<th>South Australia</th>
<th>Victoria</th>
<th>Western Australia</th>
<th>The Council</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED ASSETS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(at cost less depreciation) .................</td>
<td>—</td>
<td>2,084</td>
<td>314</td>
<td>98</td>
<td>846</td>
<td>—</td>
<td>1,009</td>
<td>4,351</td>
</tr>
<tr>
<td>INVESTMENTS (at par)</td>
<td>7,286</td>
<td>13,150</td>
<td>400</td>
<td>3,500</td>
<td>16,500</td>
<td>—</td>
<td>28,944</td>
<td>69,780</td>
</tr>
<tr>
<td>CURRENT ASSETS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cash at bank and on hand ....................</td>
<td>65</td>
<td>878</td>
<td>6830</td>
<td>1221</td>
<td>—</td>
<td>1,189</td>
<td>7,110</td>
<td>17,293</td>
</tr>
<tr>
<td>Sundry debtors ................................</td>
<td>88</td>
<td>929</td>
<td>79</td>
<td>157</td>
<td>—</td>
<td>—</td>
<td>767</td>
<td>767</td>
</tr>
<tr>
<td>Prepaid expenses</td>
<td>—</td>
<td>—</td>
<td>124</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>2,184</td>
<td>2,308</td>
</tr>
<tr>
<td>(Stocks of publications at cost) ...........</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>TOTAL ASSETS</td>
<td>$7,439</td>
<td>$17,041</td>
<td>$7,747</td>
<td>$4,976</td>
<td>$17,346</td>
<td>$1,189</td>
<td>$42,247</td>
<td>$97,985</td>
</tr>
</tbody>
</table>

Less

| LIABILITIES                                |         |        |            |                |          |                  |             |       |
| SUNDARY CREDITORS AND ACCRUALS             | 87      | 15,209 | 41         | —              | 269      | 29               | 811         | 16,446 |
| SUBSCRIPTIONS PAID IN ADVANCE              | 29      | 2,459  | 186        | 566            | 704      | 34               | 1,163       | 5,141 |
| TOTAL LIABILITIES                          | $116    | $17,668| $227       | $566           | $973     | $63              | $1,974      | $21,587 |

Surplus—Members' Equity

| SURPLUS—MEMBERS' EQUITY                    | $7,323  | ($627) | $7,520     | $4,410         | $16,373  | $1,126           | $40,273     | $76,398 |

Represented by

| ACCUMULATED FUNDS as at 1st January, 1972 | 6,335   | 5,488  | 4,283      | 3,272          | 12,475   | 704              | 27,524      | 60,081 |
| Add                                       | 971     | (6,115)| 637        | 1,138          | 3,698    | 422              | 7,165       | 7,916 |
| Balance as at 31st December, 1972         | 7,306   | (627)  | 4,920      | 4,410          | 16,173   | 1,126            | 34,689      | 67,997 |

Special Purpose Funds

| SPECIAL PURPOSE FUNDS                      | 17      | —      | 2,600      | —              | 200      | —                | 5,584       | 8,401 |

TOTAL MEMBERS' EQUITY

| TOTAL MEMBERS' EQUITY                     | $7,323  | ($627) | $7,520     | $4,410         | $16,373  | $1,126           | $40,273     | $76,398 |

### REPORT OF THE AUDITORS

In our opinion, based on our examination and the reports of the auditors of the State Branches not audited by us, the consolidated accounts set out above and on page 1 give a true and fair view of the state of affairs of the Australian Computer Society Incorporated at 31 December 1972 and of its results for the year ended on that date.

6 September, 1973

Coopers & Lybrand
Chartered Accountants

CORRECTION: Page 90 of the May 1973 issue should include the name of the auditors—Coopers and Lybrand, 13th April, 1973.
"BEEP... BEEP... BEEP... BURP!"

At last a data transmission test set that tells you what the trouble is — the Racal-Milgo 220 Test Set. As manufacturers of Modems, multiplexers and other data products we felt it was time to produce the ideal test set.

So now you can quickly set up or isolate faults in your data communications network. The 220 is compatible with any modem (not just our's) which operates at 10 to 300,000 BPS using either CCITT V24 or EIA RS 232 interfacing (it has APO permit to connect C72/7/638).

Bit and block error rates and bias distortion can be displayed on the three digit LED display. Lamps indicate 'error', modem 'ring', sync loss, transmit and receive clock. A range of options enables full local testing using only one modem and either synchronous or asynchronous operation (using frequency translation). The 220 is really compact — only 15¾" x 12½" x 3¾" — and weighs just 15 lbs.

But the biggest innovation is the little loudspeaker — listening to the line is often the quickest way to eliminate the burps!!!
Computer technology changed a lot last year—so did our text books.

Lecturer, Librarian, Instructor, Student . . . whatever your position . . . you need all the knowledge you can gather on the ever-changing and expanding science of computer technology. SRA, the educational specialist and a subsidiary of IBM, has met that need with its new, updated and expanded library of computer texts for 1973—which studies all the latest advances. Here’s how they can help you.

Lecturers.

These aren’t simply basic texts. You’ll probably find all your requirements for a complete course between the two covers. All the titles have been updated for 1973. The reading material has been designed for both graduate and post graduate students. And in many cases the reading material also includes, as an optional extra, Instructors’ Guides, Class Room Transparency Masters and Student Material—which helps to make these particular texts so much more effective and allows you to bring out the best in each individual student.

Librarians.

If it’s your responsibility to build up a professional library then you’ll certainly want to know everything about the new 1973 range of SRA Computer Texts. The prices are realistic. They offer exceptional value and will form a valuable reference and source of knowledge in your library.

Staff Training Personnel.

If you’re involved in “in-Company” training programmes you’ll find SRA Computer Texts a real help in implementing your lectures. And that’s because you can combine SRA’s fine text books with optional teachers’ aids. These aids have been designed to work with the text books. So in essence they form a complementary learning system. A more effective method of communication and understanding.

Students.

As a student the kind of knowledge you receive is vitally important. Obviously, it must not only be accurate but also current. That’s why SRA texts will appeal to you. These highly informative references provide up-to-date knowledge of your particular subject, carefully researched and fully documented—in a way which will fit in with your study requirements.

SEND NOW FOR FREE CATALOGUE

which lists and describes all the titles in the SRA 1973 range of Computer Texts.

Titles such as:

• Computers & Society
• Flowcharting Techniques
• Computer Systems Fundamentals
• Punch-Card Data Processing
• Programming with Standard COBAL
• Concepts of Programming Languages

To secure your copy of the SRA Catalogue of Computer Texts mail this coupon or phone if you wish.

NAME

ADDRESS

STATE POSTCODE

Science Research Associates Pty. Limited
82-84 Waterloo Road, North Ryde 2113
Or phone: 888-7833
News Briefs from the Computer World

“News Briefs from the Computer World” is a regular feature which covers local and overseas developments in the computer industry including new products, interesting techniques, news-worthy projects and other topical events of interest.

PERTH HOSPITALS TO USE PDP-11/40s

Two Perth Hospitals will soon have the most extensive on-line real-time computing systems for medical use in Australia. The Sir Charles Gairdner Hospital and the Royal Perth Hospital will each have a Digital Equipment PDP-11/40, following recommendations of a joint committee established by the two hospitals.

In each hospital, the computing facilities will be developed and administered independently by the departments of medical physics, with the co-operation of the WA Government Medical Department, and will be used for clinical and research work in the fields of nuclear medicine and lung physiology.

Storage capacity of the system at the Sir Charles Gairdner Hospital will be 40K and the Royal Perth Hospital system will have a capacity of 32K words of core storage.

Both installations will use the Digital Equipment RSX-11D real-time operating system and will have RK05 disk drives, paper tape readers and punches, and storage CRT displays.

One of the most interesting applications will be the processing of data from a gamma camera, which will display an image of a radio-isotope in a living organ, to detect malignancy or organ malfunction.

about that terminal...

Whatever your terminal requirement, Datagraphix will build it... around the ALPHA OR NAKED MINI 16

5 different I/O systems combined with the most complete set of I/O instructions (33) of any computer in its class give ALPHA/NAKED MINI® the most powerful and easy to use I/O structure in its class.

This, combined with the 1.6 micro second cycle time, 152 basic instructions, and a wide variety of low cost peripheral equipment, enables us to configure the precise system to meet your terminal requirements.

Get the full facts now from DC DATAGRAPHIX

ACTA USES SPACE TECHNOLOGY FOR MORE EFFICIENT SERVICE

ACTA Pty Ltd will shortly be using a space satellite system to provide a world-wide computer link-up for the ultimate benefit of its shipping client.

All this will become possible through the introduction of a new item of computer hardware—an STC telex remote terminal linked into a Honeywell H-1648 computer in Melbourne.

By the use of special transmission lines, ACTA’S traffic staff through the computer can call on the increased power of the H-1648 as and when desired to assist in forecasting container requirements throughout Australia weeks in advance.

(ACTA’S systems and data processing division uses a Honeywell H-200 for its day to day processing of documentation and research projects.)

Reports from the various ACTA marketing divisions throughout Australia are fed to the terminal H-1648 and provide the basis of calculations for container needs in the various centres throughout the country.

Figures on the inflow of containers arriving on both the ACTA/ANL Line and PACE Line ships as well as the various slot-charter vessels available to ACTA are also fed to the computer which is then able to provide forecasts of the container at any particular time, and any particular location within the next six weeks period.

UNIVERSITY OF OTAGO
Dunedin — New Zealand

SENIOR LECTURER OR LECTURER IN COMPUTER SYSTEMS

Applications are invited for the position of Lecturer or Senior Lecturer in Computer Systems in the Faculty of Commerce. The position involves developing and teaching courses in the analysis and design of information processing systems.

Practical experience is desirable, both in the technical design aspects and in the management of systems projects. Opportunities exist for consulting work.

Salary scales: Lecturer: $NZ6,335-$NZ8,103 per annum.
Senior Lecturer: $NZ8,449-$NZ10,606 per annum with a bar at $NZ9,723 per annum.
(Note: $NZ100 = approx. £60 sterling; $US147; $A100.)

Salary scales are subject to both triennial review and regular cost-of-living adjustments.

Further particulars are available from the Secretary-General, Appointments Department, Association of Commonwealth Universities, 36 Gordon Square, London W.C.1, or from the undersigned.


J. W. Hayward,
Registrar.

THE UNIVERSITY OF TASMANIA

CHAIR OF INFORMATION SCIENCE

(AMENDED ADVERTISEMENT)

The University has approved the establishment of a Department of Information Science and invites applications for appointment as foundation Professor of Information Science. The appointee will be required to plan and establish the Department of Information Science.

The present professorial salary is $19,102 per annum. Further information is available from the undersigned.

D. A. Kearney
Registrar.

The University of Tasmania,
Box 252C, G.P.O.,
HOBART 7001.

PROGRAMME YOURSELF FOR A COMPUTER COURSE

-OR BUY TIME IF YOU LIKE

For complete information call for our commercial handbook which contains details of all courses designed to meet the syllabus accreditation standards of the Australian Computer Society. If you are interested in buying time enquire about availability of hands on time. Our Calcomp drum plotter provides plotted output for most engineering applications.

TAYLOR'S COMPUTER CENTRE
300 Little Collins St., Melbourne. 3000. 63 2566

Your other computer.

CSA

INFONET
AUSTRALIA'S FULL SERVICE
COMPUTER UTILITY

* General Timesharing
* Conversational Remote Job Processing
* Remote Job Entry

Australia's most extensive specialised Library of Programs

Sydney:
CSA Centre
460 Pacific Highway
St. Leonards, NSW 2065
Tel: 439 0033

Melbourne:
Folkestone Building
33 Albert Road
Melbourne, Vic 3004
Tel: 267 2533

Canberra:
Phoenix House
88 Northbourne Avenue
Braddon, ACT 2601
Tel: 47 8611
THE AUSTRALIAN COMPUTER JOURNAL

Volume 5, Number 3
NOVEMBER, 1973

Price: $1.00 per copy to non-members, 50 cents per copy to members.

The Australian Computer Journal is the official publication of the Australian Computer Society Incorporated.

Office-bearers to May, 1974:
President: P. R. MASTERS
Vice-President: A. W. GOLDSWORTHY
National Treasurer: K. G. ERNST
National Secretary: K. S. ARTER
Executive Secretary: Mrs. Jennifer PRESDEE
P.O. Box 640,
Crows Nest, N.S.W. 2065.
Tel.: Sydney 439 2434

Editorial Committee:
P.O. Box 150,
Kingston, A.C.T. 2604.
(No please indicate whether correspondence is for the Editor or Business Manager.)
Editor: T. PEARCEY
Associate Editors: J. M. BENNETT
A. Y. MONTGOMERY
Publications Officer: Mrs. E. BODLEY

Published by: Australian Trade Publications Pty Ltd., 28 Chippendale Street, Chippendale, N.S.W. 2008. Tel: 699 1433.

Special Features:
114
114
114
126
131
140
141-142

INDEX

Contents:
96-104 Staged Introduction of Manufacturing Information Systems
By R. J. HILL

105-114 The Development of the ACL Language and its Implementation ACL-NOVA
By N. W. BENNETT and P. L. SANGER

115-126 The Design of Indexed Sequential Files
By K. J. McDONELL and A. Y. MONTGOMERY

127-131 A General Validation Procedure for Computer Simulation Models
By P. GILMOUR

132-140 A Survey of Automatic Flowchart Generators
By R. P. WATKINS

STAGE ONE

141-142 Balance Sheet

EDITOR’S NOTE
The editor regrets that owing to circumstances beyond control no August issue will be published. Thus there will be only 3 issues in Volume 5.