J. M Bishop
24/3/91

South African Computer Journal
Number 4
March 1991

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 4
Maart 1991

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

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

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Suid-Afrikaanse Rekenaarvereniging en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

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

Assistant Editor: Information Systems
Dr Peter Lay
P. O. Box 2142
Windmeul 7630

Subscriptions

	Annual	Single copy
Southern Africa:	R32-00	R8-00
Elsewhere:	$32-00	$8-00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Computer Science Department, University of Pretoria, Pretoria 0002
Phone: (012) 420-2843 Fax: (int) + (012) + 43-6454

Sigma Press

Guest Editorial

Does Today's Industry Need Qualified Computer Scientists?

This guest editorial consists of two contrasting views on the value to industry of a professional degree in computer science. Both authors, one local and one from Germany, are managing directors of well-respected software houses. (Editor)

Viewpoint I

Hans G Steiner
MBP Software and Systems GMBH
Semerteichestrasse 47-49
D4600 Dortmund 1

I would like to begin by recounting from my student days a story that I consider to be relevant. While attending a career forum for computer scientists, mathematicians and physicists, the personnel officer from IBM Germany was asked if he would consider taking on mathematicians. The gist of his answer was as follows: "Of course I must admit that I could just as well give the mathematician's job to a theologian. What is important is the ability to think logically. It is only there, on the job, that he learns how to become productive for us."

This episode occurred 14 years ago at a time when graduating mathematicians did not necessarily learn programming and when computer scientists were few and far between. The situation has improved immensely since then. Mechanical engineers, electrical engineers and physicists, all with programming knowledge, have for the most part taken over many programming jobs. This shows industry that, as time goes by, the answer to the opening question is becoming an ever-louder and more frequent "NO".

I support this opinion and in the remainder of this essay I will expand on my reasons, as well as highlight some exceptions.

An employee who is recruited directly from a university should possess the following four capabilities:
1. An ability to think logically: One of the basic requirements in our business is the ability to recognise, analyze, structure, break down and solve a problem as well as to fully synthesize the solution. The important thing is to break down the problem in such a way that the individual components can feasibly be solved. This is what distinguishes an engineer/scientist from an arts scholar. The latter usually concentrates on the complete problem and tends to settle for a contentious, complex and partially non-feasible solution. In our business, it is not enough to merely ask the "right" questions.

This ability to think logically may be accentuated in computer science; however engineers/scientists will generally possess the ability to an equal extent.

2. Programming skills: Our employees' prime tool of the trade is their ability to encode solutions to problems. Ideally, this ability ought to be held as abstract as possible. In other words, the further away from the "bit", the better. FORTRAN programmers who, for example, concentrate on the multiple use of memory space of all variables will never be successful programmers in an object-oriented programming language.

The difference can be seen, even in today's universities. For example, one only has to read a PROLOG program from a student who learned PASCAL in his first semester and PROLOG in his fifth. On average, this is always a "PASCAL program in PROLOG". The various possibilities offered by a predicate calculus language are only recognised and used by the best students. Again, we do not need the average computer science scholar who has spent between six and eight years writing complicated PASCAL programs, but rather the "thinker" with basic programming knowledge who is capable of abstracting the task. Once again, the ability is independent of faculty.

3. Teamwork skills: Working successfully in a team

This SACJ issue is sponsored by
Department of Computer Science
Rhodes University

requires assertiveness, tolerance, stability and one's own ideas. Very few problems have solutions that can be managed by one person successfully in the allocated time. Out of 700 employees, we can only afford approximately five "lone warriors" who are, in turn, the leading specialists in a wide field. They have a strategic vision which we follow. All remaining employees are evaluated, for better or worse, on their team performance. Some people have an in-built ability to work in teams. A few universities - unfortunately not enough - encourage this team-thinking. Again we see that the ability is independent of university faculty.

4. *Motivation:* The ability to enjoy one's particular job is a major driving force in every employee. Whereas in the sixties everything had to be "bigger, faster and better" and in the eighties "things had to be meaningful to society", the theme for the nineties is self-realization. Those companies who succeed in incorporating different employees (ie employees with different driving forces) into the company culture and who motivate each employee optimally will be successful in the nineties and beyond. There are huge productivity gains to be had from motivating employees. Compared with this, the possibilities offered by CASE tools pale into insignificance.

One basic requirement is thus the recruitment of a self-motivated employee who should at no stage become demotivated, whether it be by company culture, superiors or working conditions.

Again, this is not linked to a specific university faculty and is independent of know-how.

As none of these four capabilities are necessarily restricted to studies in computer science, the technical/-scientific background of new employees who are being recruited is largely irrelevant.

I would now like to point out a few exceptions which might give a computer scientist the upper hand in an interview. I refer exclusively to our own company and our specific company tasks.

1. Porting our COBOL Compiler onto the latest UNIX machine from the manufacturer XY. Knowledge of the UNIX operating systems could be very valuable and enable the new employee to rapidly become productive.
2. Programming the 37th interface (special customer request) for our ISDN card. Knowledge of interface protocols or experience with protocol conversions would be very useful and could be a decisive factor. Such specialized knowledge is usually very rare.
3. Adapting our integrated office automation system to the 17th foreign language. The employee must command the language perfectly. Simply outsourcing the translation would mean that this language version could not be maintained or supported. From this example one can see that specialized knowledge not only refers to knowledge gained from computer science studies.

In the product business, it sometimes happens that computer scientists with specialized knowledge are sought. (This is almost impossible in the project business, due to the variety of tasks to be performed.) However such a "knowledge" advantage over others usually only lasts about a year. After that, the achievements of two different employees (one with specialized knowledge and the other without) tends to even out.

Most applicants who start out do not know our products, as the flow of employees in this industry is almost always from manufacturer to user. Hardware and software manufacturers often lose their products specialist to the products' users. Seldom do employees change in the other direction.

In my opinion, universities can learn two things from this essay:

1. Studies in computer science give basic knowledge that can be used in various jobs. The student should however be careful not to place all his eggs in one basket.
2. Teamwork should be encouraged more. Time allows for very few geniuses, acting as 'lone warriors', to initiate progress in our society.

I have taken the liberty of basing my interpretation and answer to the opening question on my own judgement and experiences. I would be grateful for other opinions and experiences on this topic.

I would like to conclude by expressing my gratitude for having had this opportunity to express my views.

Viewpoint II

Pierre Visser
Grinaker Informatics, P.O.Box 29818, Sunnyside, 0132

The title question currently generates as many viewpoints as a counterpart question: "What is the correct curriculum for a computer science qualification?" Such questions stem from the many and diverse requirements expected to be fulfilled by the still developing applied science. A basic assumption of this editorial is that we need to have an explosion and consolidation in computer science theory. Only after this has occurred will a more general consensus of opinion exist - as is the case in other matured sciences.

An argument is presented here for the current approach of striving towards a balance between immediate industry needs and long term perceived theoretical requirements of industry, even though the balance, as viewed from either side, will always be imperfect.

Industry can, of course, do without qualified computer scientists - that is how it was established. Dedicated mathematicians, physicists, engineers and other scientists will, as in the past, continue to effect improvements. However, as one of those scientists from the

early days, it is difficult for me to understand why one would choose to continue this way.

A computer science qualification is viewed here as a university education (4 years) into theory that is not obtainable otherwise. By definition, therefore, a qualified computer scientist is not trained to conform to specific job requirements. Rather, the computer scientist will possess knowledge that will serve him long past the present day's computing technology.

Whether industry needs qualified computer scientists depends on two issues. Firstly, can an education be provided for computing technology that will serve as a foundation for the student's next 45 years in industry; and secondly, can industry build upon this foundation to created wealth more effectively than without qualified computer scientists.

It is widely accepted that, in broad terms, the teaching of fundamental theory will serve the first purpose. However, what subject matter to include from the wealth of mathematics, physics, OR, and from computing fields such as networking, operating systems and others, remains the illusive issue. Universities can merely strive to select the right mix for the perceived future needs of industry. This requires insight into the evolution of computing technology. I will later discuss such insight as a basic requirement for a qualified computer scientist.

What is important in teaching is to focus on fundamental theory. Just as the natural science student needs to breed fruit flies in order to gain insight into the dynamics of inheritance, so too the computer science student needs to develop software. The purpose should be to create understanding and insight into fundamental theory, and, just as in the case of the breeder of fruit flies, the software developed should never be measured against efficiency requirements from industry.

The second issue is whether industry can build on this theoretical foundation to create wealth.

A depth of insight into computing technology, more so than with other training, can be identified as the focus of the potential value of a qualified computer scientist to industry. Three areas which require such insight are discussed below, namely organisation, product definition and the application of new computing technology in industry.

Computing products form an integral part of an organisation, and represent a significant capital investment aimed at increasing efficiency. These products are incorporated in an evolutionary way to match changing organisational requirements with improving product capabilities. Decisions to use products determine the long term efficiency and cost-effective replacement. Such decisions require insight into computing technology and its evolution. A qualified computer scientist can improve such decisions only if he gains enough insight into computing technology as well as its interaction with business through years of practice.

The success of products in some areas is dependent on market requirements which depend on computing technology and its evolution. The correct definition of characteristics of products that interface to computers is such an example. Insight into computing technology is able to create the versatility, simplicity or other improved selling features which can open new market segments.

The third area where insight into computing technology plays an important role is in the application of new computing technology (or a new trend) in an organisation. Examples include the introductory period for networking, DBMS-technology, distributed processing and document image processing. In areas such as these, the newly qualified computer scientist can be applied effectively and at the same time build up insight through experience which he will require for the other areas of organisational and product decisions mentioned above.

A major dilemma in the continuous development of insight into computing technology by qualified computer scientists is their correct application in industry. The identification of the opportunities within the three areas discussed above, requires insight into computing technology itself. Winning companies that depend on computing technology have this ability. In such companies the insight of the qualified computer scientist into computing technology as well as its contribution to the business is constantly stimulated, turning the qualified computer scientist into a valuable company resource.

What has been neglected in this whole discussion is the role of the "technician" and of the casual user of computing technology. Such personnel are required to implement selected computing technology of the day efficiently, whether in accounting, chemical engineering or other specialised disciplines. Their role and place is unquestioned. However, it cannot be expected of them to evaluate the potential of new computing technology, formulate algorithms from fundamental theory or any such decisions which require insight built upon a sound theoretical knowledge of the field.

The final aspect in answering the opening question is whether the qualified computer scientist can outperform other professionals who build up their own experience in computing technology. Many examples could be cited of improvement brought about by non-computer scientists in the past. However, these individuals formed part of the bootstrapping for computer science theory and education. We should have faith in this bootstrapping of computer science qualifications, because computing technology will increasingly diversify into many directions of specialisation in years to come, each requiring a body of fundamental theory.

This complexity cannot be left to a casual development of insight - industry requires qualified computer scientists to experience interaction with business objectives in order to cope successfully with future computing technology.

An Analysis of the Usage of Systems Development Methods in South Africa

S Erlank, D Pelteret and M Meskin
Department of Accounting, University of Cape Town, Private Bag, Rondebosch, 7700

Abstract

The systems development process is commonly described in terms of a System Development Life Cycle (SDLC). There is a wide range of tools and techniques available to the system developer, and these are typically used for different activities at different stages of the life cycle. Some of these tools have been the focus of much research but are not widely used in practice. Others are widely used, but ignored in the literature. Many of the traditional tools have been incorporated into so-called systems development methodologies, while others have fallen into disuse over time. For many organisations, the extent of automation of a tool will dictate its effectiveness in developing systems.

This exploratory study examines the incidence of usage of tools and techniques for systems development in South Africa. The findings of the study are compared to those of previous researchers to ascertain whether some stages of the SDLC are neglected, and to compare the South African situation to the international environment. The implications of these findings for structured methodologies and CASE are discussed.

Keywords: *System Analysis, Systems Design, Methodologies, CASE, Systems Development*
Computing Review Categories: *K.6.1*

Recieved May 1990, Accepted October 1990

1. Introduction

During the last three decades a variety of methodologies, tools and techniques has been developed to aid in the systems development process. With the evolution of software engineering and the systems development life cycle (SDLC) the process of system development has become highly structured. Lyytinen [8] refers to several "generations" of systems development techniques identifiable in this evolution process.

Many early tools and techniques such as flowcharts, Structured Design, Jackson diagrams, and Nassi-Shneiderman charts applied only to the latter stages of the SDLC and concentrated on design. Later research focused on the early phases of the SDLC, with emphasis on the requirements specification phase and the importance of user involvement [5, 12, 14]. Currently, integrated computer-aided software engineering (CASE) is receiving attention.

Some systems development techniques have proven to be very popular [2]. Others have fallen into disuse over time [17]. Some are used inappropriately, possibly because of a lack of awareness of alternatives [11]. The progression of systems analysis practices has been "punctuated by techniques and methods that ebb and flow with rapid frequency" [15]. Arguably, each new technique grew out of a deficiency in the existing available tool set, or evolved from a need which was not yet addressed by these tools [13].

Several authors have categorized systems development methods in an attempt to provide a framework for comparing and evaluating them [4, 6, 9, 11]. Others have examined the incidence of usage of isolated tools and techniques with conflicting findings [13, 2, 10, 7]. However, there appears to be no empirical basis for the selection of specific methods for study; nor is there any commonality between the methods selected by separate authors.

This exploratory study surveys a comprehensive list of systems development tools and techniques to determine the incidence of usage and the practitioner's level of knowledge of each in South Africa, thus providing a practical focus for the paper, and for future research.

The findings are applied to a framework developed by Colter to determine how comprehensively the systems development process is served by these tools [4]. Since this study also measures the level of automation of each tool as used, the implications for CASE and structured methodologies are discussed. In the interests of brevity, methods are identified by their commonly known acronyms in this paper.

2. Previous research

Existing research relating to Systems Analysis has been categorised by Vitalari [15]. Of relevance to this study

are attempts by previous authors to develop conceptual frameworks for the **classification of systems analysis methods**, and previous studies on **usage of such methods** in practice.

Classification of tools, techniques and methodologies

Hackathorne and Karimi [6] clarify the ambiguity in the literature regarding the terms "tool", "technique" and "methodology". They define a **technique** as "a procedure for accomplishing a desired outcome". A **tool** is an "instrument for performing a procedure", being in particular a tangible aid such as a form or computer program. Thus prototyping would be a technique, but a screen-painting program a tool used in prototyping.

A **methodology** is a conceptual approach that defines what is required to address a given problem. It can perhaps be considered as a collection of tools and techniques within a conceptual framework. The generic term **method** is used to describe any of these terms.

Wood-Harper and Fitzgerald [16] argue that there are six major approaches to systems analysis, each with its own underlying paradigm and objective. They identified the General Systems Theory, Human Activity, Participative, Data Analysis, Traditional (SDLC) and Structured Systems Analysis Approaches, and developed the taxonomy shown in Figure 1. This taxonomy is derived in part from the methods used in the systems analysis process.

Hackathorn and Karimi [6] developed a framework for comparing information engineering (IE) methods. The framework is a 2-dimensional grid whose axes are "breadth" and "depth". The breadth dimension corresponds to the phases of an adapted and extended SDLC; the depth dimension refers to a conceptual-to-practical continuum, along which a given method could be located. This continuum effectively dictated whether a method was a methodology, technique or tool. In verifying their framework, they selected 26 methods from a sample of candidate IE methods, the selection of such methods being based largely on their usefulness for testing the model.

Shemer [12] takes a narrower view than Hackathorn et al in his analysis of a conceptual model of systems analysis in that his critique has its foundation in what the latter would term the Traditional or Structured Systems Analysis approaches. He mentions several techniques used in developing a requirements specification, citing some of the "prominent and most used" as SSA, SADT, PSL/PSA, SREM, information analysis and entity relationships. The empirical basis for this selection is not clear.

Mahmood [9] found that the effectiveness of a system design method is influenced by a project's requirements, characteristics, user and designer satisfaction, and the impact on the decision-making process. The implication of this finding for practitioners who use a limited portfolio of systems development methods is that they may not in all cases be using the most effective method available.

In order to compare different methods of systems analysis, Colter [4] identified a number of dimensions, drawn from both Traditional and Structured approaches to analysis. These dimensions evaluate a method in terms of its ability to refine the structure or detail of a system, and whether the method is appropriate at a high or low level of abstraction. The model also evaluates how well the method can be used

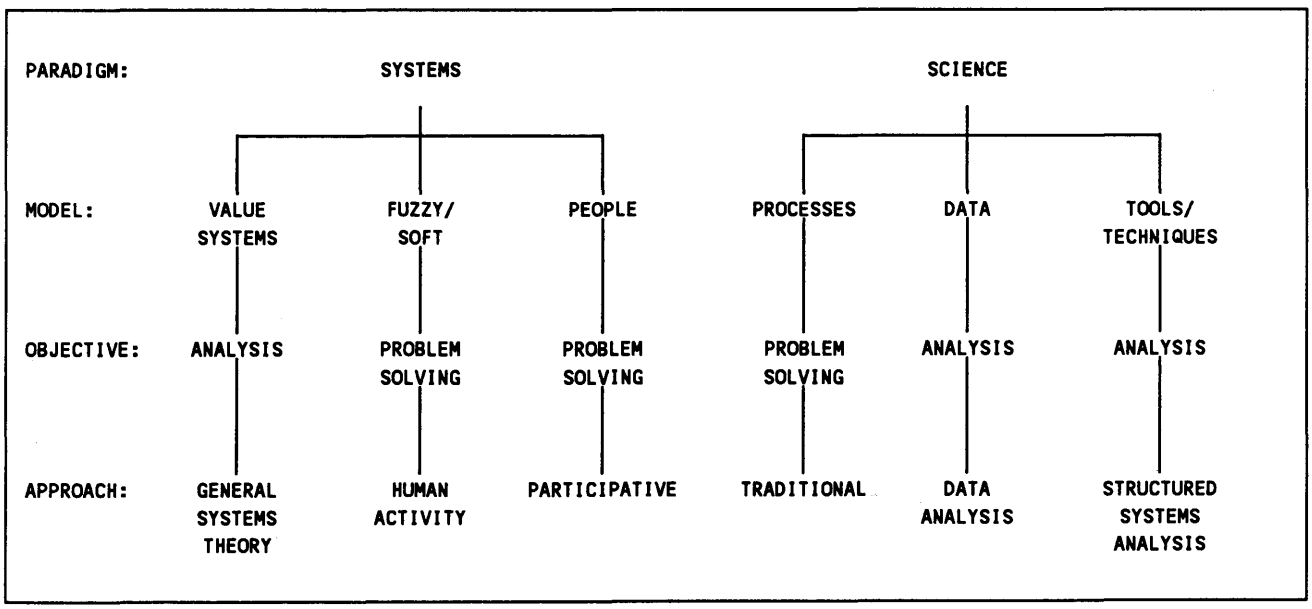

Figure 1: Taxonomy of Systems Analysis Approaches.

to communicate between members of the development project. The structure dimension is divided further into general structure, data flows, data structure or control structure. Methods are evaluated on their performance in mechanism clarification, function analysis, procedure detail definition, input or output design. Also measured is whether a method is problem- or solution-oriented. Figure 2 shows a selection of the techniques evaluated by Colter. The extent of coverage of the systems analysis process by each technique is shown.

Other multi-dimensional classifications of system development "methodologies" exist. Practitioners might be aided and guided by such a classification in selecting the most appropriate methodology for systems development. However, most practitioners know only very few methodologies, making it more difficult to assume that a real choice can take place.

respondents) and systems flowcharts (75%) were the tools most often used, while least popular methods were decision tables (5%) and Nassi-Shneiderman (7%). Comparatively unpopular were HIPO (25%) and Warnier-Orr (25%). This study did not measure level of knowledge of the methods, nor the extent of automation of the method.

A study by Necco, Gordon and Tsai [10] overlapped the study by Carey and McLeod in the choice of methods. They found that the most popular documentation aids were narrative descriptions, system flowcharts, file layouts and input/output layouts. The most widely used techniques for structured analysis and design were data dictionaries, structured walkthroughs, data flow diagrams and structure charts. The only widely used automated tools were screen design facilities and data dictionary packages.

A survey by Sumner and Sitek [13] of the members

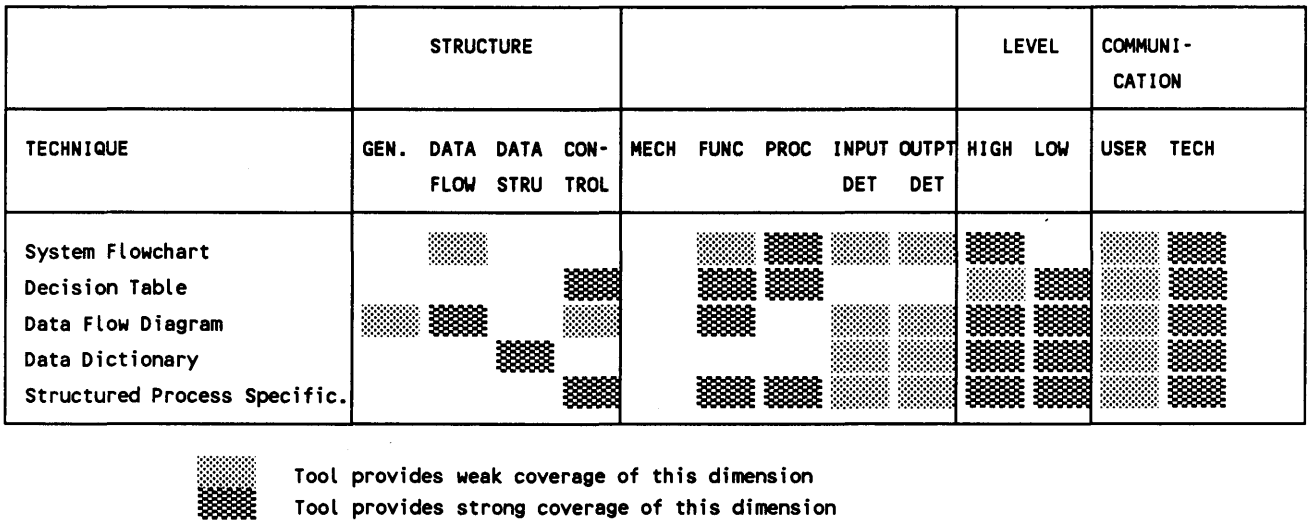

Figure 2. Framework for evaluation of systems analysis methodologies

No single tool, technique or methodology can support the complete analysis process, and many organisations have adopted a 'package' of methods, typically including function descriptions, hierarchy charts, DFDs and data dictionaries [4].

In order to provide a practical focus for research, several authors have attempted to determine usage statistics for different systems development methods.

Usage statistics

Carey and McLeod [2] examined the usage of 11 systems development methods in Texas. They found that 5% of the organizations that were included in the survey made use of all 11 tools and 16% of the sample did not use any of the tools surveyed. Whether the latter statistic includes organisations that used other methods (not included in the survey) is not clear. The study showed that data flow diagrams (79% of

the Association of Systems Managers in St Louis requested, inter alia, information about the tools they used for systems development. Of 45 recently initiated projects, they found that in more than half of the projects interviews, data dictionaries, systems flowcharts, and structured walkthroughs were used. Tools which were not significantly used were decision trees/tables, HIPO, SADT, Warnier-Orr, Nassi-Shneiderman and Chapin Charts.

Kievit and Martin [7] selected certain analysis tools for further study, finding much higher levels of usage than in the studies by Sumner and Sitek or Carey and McLeod.

While these four studies address the broad area of systems development, they focus on different sets of methods. Where commonality exists, the results do not always correlate, as will be shown.

Although this previous research was performed in the broad area of systems development methods, there was little commonality in the methods reviewed. Methods reviewed by one author would be ignored by others although prototyping, DFD's and SADT were discussed by several authors. Most methods however were discussed by only one author. Most authors postulated frameworks for comparison or evaluation of methods, selecting methods to illustrate or prove their models without a firm basis for the selection of such methods.

Some anomalies and gaps exist, then, in previous research. Theoretical models and frameworks have been developed and tested on methods which are apparently not widely used. Methods which are of great relevance to practitioners are omitted from the theoretical models, and conflicting findings exist in the empirical work that has been done.

3. Research objectives

The objectives of this exploratory study are to address the following questions:

1. What is the incidence of usage of systems development methods in South Africa?
2. What is the extent of knowledge amongst practitioners of the available methods?
3. What is the level of automation for the most widely used techniques?
4. To what extent has the systems development process been formalised in organisations by the use of a standard methodology.

4. Research method

A list of system development methods was compiled based on an extensive search of the literature. A questionnaire requesting the following information for each method was developed. An "other methods" category was included so that respondents could identify any methods they used which were not already specified.

1. **The use of the method:** The respondents were given a range of five possibilities to indicate their level of knowledge about the method and its frequency of usage within their organization. The possible answers were:
   Never heard of it/Know of, never use/
   Use seldom/Use often/Use always

2. **Automation of the method** : The respondents were asked to indicate which of the methods that they used were automated, i.e were performed using a computer.

3. **Use of techniques as part of a methodology:** Each of the techniques included in the survey can be used on its own or as part of a standard methodology. The respondents were asked to identify which techniques that they used in systems analysis formed part of an overall methodology in their organisations.

A list of 152 companies throughout SA that had computer departments of more than twenty personnel was compiled from the Computer Users' Handbook. A key person in each organisation was requested to distribute the questionnaire to personnel who played leading roles in systems development (typically a project manager) since it was believed that these people would be able to identify methods used with a fair degree of accuracy.

It being possible that different project groups might use different methods within a single organisation, multiple returns from such organisations were aggregated. In total, 421 questionnaires were distributed, an average of 2.77 per organisation.

Four key indicators were derived from the questionnaire:

1. **Level of awareness:** This variable measures the percentage of all respondents indicating that they had heard of the method, whether or not they used it.
2. **Incidence of usage:** All responses stating that a specific method was used (whether seldom, often or always) were totalled and this amount was divided by the total number of responses.
3. **Degree of automation:** This measures the percentage of respondents using the method who employed a computer to do so.
4. **Part of a methodology:** This percentage, used as a measure for determining the level of formality in systems development, indicates which of the methods being used is used as part of an overall methodology.

5. Analysis of results

103 responses out of 421 sent (24.5%) were returned. 52 organizations (34.2%) responded to the survey. The actual size of the information systems departments surveyed ranged from 4 to 1050 personnel, while the average size was 77 personnel. On average, respondents indicated that 57.1% of work effort was spent in development. 67.3% made use of personal computers in systems analysis and design and 88.5% made use of fourth generation languages. 80% of systems were mainframe-based, with mini-computers (14%) and PC's (6%) accounting for the balance.

"Other" tools and methodologies

Respondents recorded usage of 13 other tools which were not specifically included in the questionnaire. Since none of these was duplicated, their level of usage was less than 1%. The implication is that the selection of tools and methodologies listed in figure 4 is comprehensive in South Africa.

The level of awareness

The level of awareness of the methods surveyed was generally high (see Appendix I). The methods listed in Figure 3 were known to all respondents. More than 80% of respondents had heard of 39 of the 56 techniques and only 8 techniques were unknown to the majority of the respondents. However, awareness of a technique does not necessarily imply that it is widely used in practice.

The incidence of usage

Appendix II portrays the incidence of usage of each method. The percentage of respondents who use the method all the time is also shown. 16 methods were used by more than 70% of respondents. Conspicuously low in usage are such well-documented methods as HIPO, Warnier-Orr, decision trees/tables, pseudocode and Jackson diagrams, which are used by less than a third of respondents. Many of the most-used methods are applicable at early stages of the SDLC for requirements analysis and specification.

Figure 4 compares these results against previous studies. For brevity, the most pertinent methods are displayed for comparison. Figures for Sumner and Sitek have been converted to a percentage, and those of Kievit and Martin aggregated.

The results confirm, in many cases, the findings of previous studies, with the exception of the study by Sumner et al, which records substantially lower usage of all methods than the other studies.

The fact-finding techniques we surveyed all recorded high usage. Feasibility studies were used by all respondents (25% of whom used them all the time). As in the study by Necco et al and Carey & McLeod, data flow diagrams, system flowcharts and structured walkthroughs are all widely used. The high incidence of usage of data dictionaries, Entity-Relationship modelling and normalisation indicate strong support for the Data Analysis approach to systems development [16].

The strong support shown for the methods normally applied at early stages of the SDLC is not as apparent in the methods used at the design stage, although the majority of respondents use structure charts, for example. Less support was shown for structured English, pseudocode and other traditional design-oriented methods.

Cost Benefit Analysis
Brainstorming
Feasibility Study
Presentation
Documentation
Review of Previous Documentation
Systems Flowcharts
Observations
Data Flow Diagram
Interviews

Figure 3: Methods known to all respondents

The degree of automation

The degree of automation of all the tools can be found in Appendix III. Some anomalies exist in the results. To what extent interviews can be automated is not clear for example. Other methods such as prototyping, report generators, and on-line development aids were expected to have higher percentages. Whether this is due to misunderstood terminology or inaccurate response requires further verification. Until such time, these figures should be treated with caution.

Nevertheless, several of the most commonly used methods show substantial automation, and this coupled with results discussed above has implications for CASE. A degree of correlation is noted to the study by Necco et al [10], who also measured the degree of automation of certain tools.

Figure 5 summarises the degree of automation of the most highly used methods, with comparative figures of the study by Necco et al.

The degree of formality

Many of the tools were used as part of a formalised methodology. 67% of the 12 methodologies included separately in the questionnaire were unknown to the majority of respondents; 5 (STRADIS, DSSD, SADT, HOS and BIAIT) were not used by any respondent. However, 52% of organizations use a methodology (see figure 6). Three organizations made use of an in-house developed methodology or had adapted a commercially available methodology. An awareness of the listed methodologies among respondents is largely limited to the methodology used by their own organisation.

METHOD	This Study 1989	Necco et al 1987	Carey& McLeod 1988	Sumner& Sitek 1986	Kievit& Martin 1989
Documentation	100				
Feasibility study	100			29	
Presentations	98				
Interviews	98				
Documentation Review	98				
Observations	96				
Data Flow Diagram	94	80	79	33	90
Brainstorming	92				
System Flowchart	92	96	75	58	93
Data Normalisation	88				
Prototyping	88		49	24	36
Structured Walkthrough	87	90	65	67	44
Cost benefit analysis	83				
Data Dictionary	81	97		58	45
Inspection	79				
Entity Relationship Model	71				
Structure Chart	60	75	43	20	53
Decision Tables	56		5	32	56
Questionnaire	54			22	
Structured English	37		61		
Pseudocode	29			38	
HIPO diagrams	13		25	10	35
Warnier Orr	10		25	20	20
Narrative Description		100			
File Layout form		93			
Screen Design Facility		91			
Input/Output Layout		88			
Top down Analysis			70		
Nassi- Shneiderman			7	5	12
SADT	0			7	
Program Flowcharts				49	

Figure 4: Methods used by more than 70% of respondents with comparative figures from other studies.

METHOD	Incidence of usage	Degree of Automation This study	Degree of Automation Necco et al
Documentation	100	79	48
Presentation	98	20	
Data Flow Diagrams	94	47	42
System Flowcharts	92	29	
Data Normalization	88	30	
Prototyping	88	30	
Data Dictionary	81	67	86
E-R diagrams	71	46	
On-line development aids	65	74	
Functional Decomp. charts	63	45	
Report Design Facility	62	72	75
Structure Charts	60	32	38
Code/Program generators	27	64	53

Figure 5: Degree of Automation of methods, with comparative figures.

An analysis of individual methods reveals that methods such as E-R diagrams (65%), data dictionaries (64%), data normalization (64%) and data flow diagrams (61%) among other methods are used most often as part of a formal methodology (see appendix IV). Other design-oriented methods, such as structure charts, pseudocode, decision tables, structured English and Warnier-Orr, if they are used, are not used as part of a methodology.

6. Discussion of results

It is clear from the results of this study that a number of different approaches to systems development are being used, some of them simultaneously. Key approaches are the Structured Systems Analysis and Data Analysis approaches [16], while the Traditional Approach is less common. The changing nature of the systems portfolio has reduced the utility of older, traditional methods used in systems development, a fact which has been suggested by other authors [16, 4].

The similarity between the results of this study and others suggests that the systems development environment in South Africa is very similar to that of the USA, at least in technical terms.

Respondents indicated a high level of awareness of all methods, yet comparatively few methods were very widely used. The conclusion to be drawn from this is that practitioners have made a choice of the most appropriate tool for the task, at the expense of alternatives. That there is congruence between empirical studies and a focus by practitioners on selected tools such as DFD's, system flowcharts, and data dictionaries implies that these methods are becoming standard for systems development, while other methods will eventually fall into disuse.

The implications of this standardisation are many. The mobility of staff may increase as skills become more portable from one organisation to another. The maintainability of systems may be affected as documentation styles converge, and productivity of staff may increase as learning curves reduce. Perhaps the most impact is likely to be in the area of automated systems development. Many methods have been automated over the years. Increasing emphasis has been placed on the notion of integration [1] suggesting that tools must be able to work together to provide a total support environment. Since automated tools are offered by a variety of vendors, any form of standardisation will enhance the capability for multi-vendor integration, as has happened in other software environments.

Methodology	No. of co's using	% of co's responding
Tetrarch/1	7	13.5
IEW	3	5.8
SSADM	4	7.7
Infomet	5	9.6
Method/1	5	9.6
In-house dev.	3	5.8
TOTAL	27	52

Figure 6: Percentage of companies using a Standard Methodology

None of the available methods completely supports the entire systems development process. The framework developed by Colter [4] provides an evaluative structure that can be used to compare alternative analysis techniques. His study evaluated several tools, methodologies and techniques, some of which had a very low usage in this study.

Figure 7 is derived from Colter's study [4]. It excludes methods which recorded a low incidence of usage, and includes a tentative placement of methods which he did not evaluate, but which recorded high incidence of usage or high automation in our study. It would seem that the portfolio of methods included has fairly comprehensive coverage of all dimensions of Colter's model. It is noted that the model covers systems analysis only. Given the wide usage of these tools in the industry and the extent of automation, it is postulated that this set of methods will form a "core set" of methods for systems analysis, both in manual and CASE environments.

7. Conclusion

Past research has often concentrated on systems development methods that have historical or academic value, not taking into account the possibility of techniques becoming outdated or impractical. The focus of future research needs to be adjusted if it is to be relevant to the practitioner.

This empirical study identifies those methods used in systems development in South Africa. Respondents reported a wide awareness of a comprehensive list of methods surveyed, and no other methods with significant usage were indicated. Most widely used of all techniques were the generic data-gathering techniques such as feasibility studies, interviews, observations and documentation review.

Also widely used were certain formal methods associated with the analysis stage of the SDLC, such as Data flow diagrams, data dictionaries, prototyping, and systems flowcharts. Strong support was also shown for the tools of the Data Analysis approach, data normalisation and Entity-Relationship modelling. Little usage was reported of many tools, such as Jackson and pseudocode, which are associated with the design stage of the SDLC, as well as many methods common in the literature, such as SADT, HOS, HIPO and Warnier-Orr.

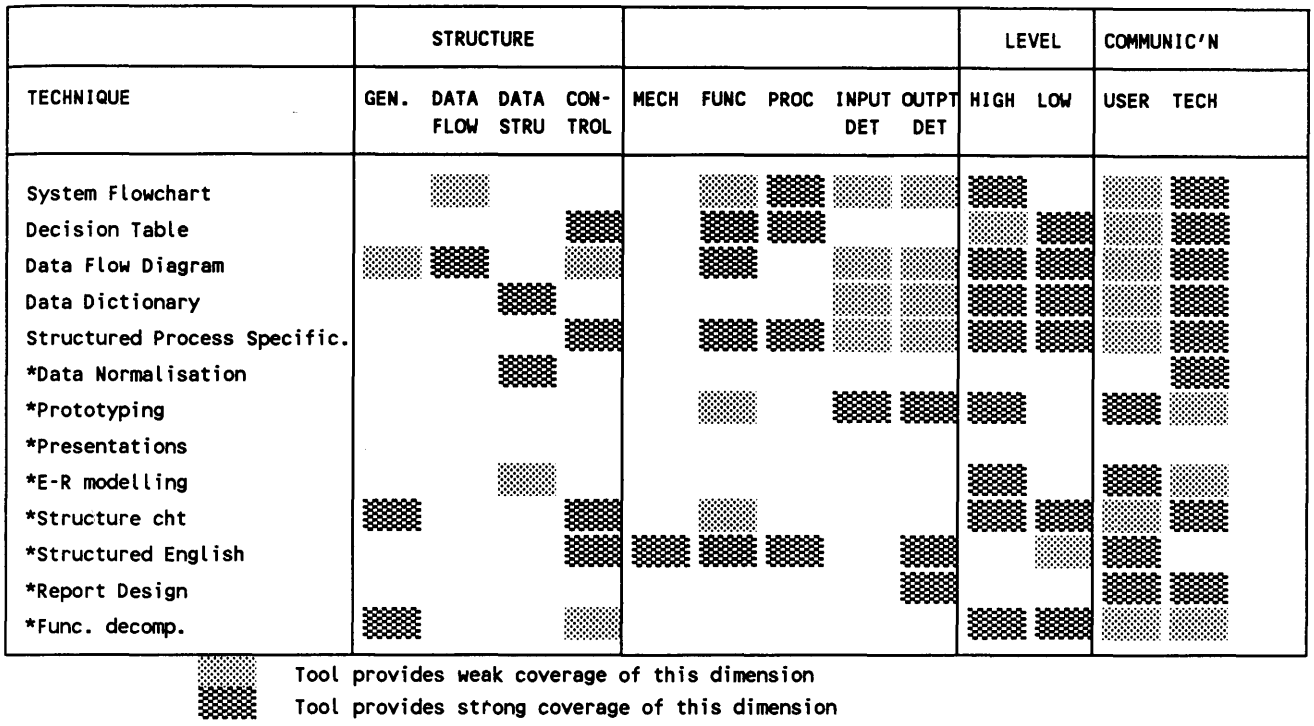

Figure 7. Adaptation of Colters model to include new methods (* indicates our insertion)

All of the formal methods used for analysis were strongly automated. Most were used as part of a formal methodology. Coupled with the fact that more than half of companies surveyed used a formal methodology, this has implications for the use of CASE, and the design of CASE tools. The application of the most-used methods to a theoretical model indicated that most dimensions of system analysis are well addressed by this toolset.

REFERENCES

[1] DL Burkhard & PV Jenster,[1989],Applications of computer-aided software engineering tools: Survey of current and prospective users,*Database*,Fall,1989,28-37.

[2] JM Carey & R McLeod, [1988], Use of System Development Methodology and Tools, *Journal of Systems Management*,**39**(4), 30-35.

[3] DI Cleland & WR King, [1968], *Systems Analysis and Project Management*, McGraw-Hill Inc., USA.

[4] MA Colter [1984], A Comparative Examination of Systems Analysis Techniques, *MIS Quarterly*,**8**(1),51-64.

[5] RD Galliers, [1985], An Approach to Information Needs Analysis, *in Information Analysis: Selected Readings*, edited by RD Galliers, Addison-Wesley Publishing Company Inc, Great Britain.

[6] RD Hackathorn & J Karimi, [1988], A Framework for Comparing Information Engineering Methods, *MIS Quarterly*, **12**(2), 203-220

[7] K Kievit & M Martin, [1989], Systems Analysis Tools - Who's using them?, *Journal of Systems Management*,**40**, 26-29

[8] K Lyytinen, [1989], New challenges of Systems Development, *Database*,Fall, 1989, 1-12

[9] MA Mahmood [1987], System Development Methods - A Comparative Investigation, *MIS Quarterly*, , **11**(3), 293-311

[10] CR Necco, CL Gordon & NW Tsai, [1987], Systems Analysis and Design: Current Practices, *MIS Quarterly*, **11**(4),461-475

[11] PA Nielson, [1987], "A Tentative Classification of Methodologies used in Information Systems Development", *Paper submitted for the 10th Information Systems Research Seminar in Scandanavia*, Vaskivesi, Finland, August 1987.

[12] I Shemer, [1987], Systems Analysis: A Systematic Analysis of a Conceptual Model, *Communications of the ACM*, **30**(6), June 1987.

[13] M Sumner & J Sitek, [1986], Are structured methods for Systems Analysis and Design being used?, *Journal of Systems Management*,**37**(6), 18-24

[14] P Tait & I Vessey, [1988], The Effect of User Involvement on System Success: A Contingency Approach, *MIS Quarterly*, **12**(1),90-107

[15] NP Vitalari, [1985], Knowledge as a basis for expertise in Systems Analysis: An Empirical Study, *MIS Quarterly*, **9**(3), 221-241

[16] AT Wood-Harper & G Fitzgerald, [1982], A Taxonomy of Current Approaches to Systems Analysis, *The Computer Journal*, **25**(1), 12-16

[17] E Yourdon, [1986], What ever happened to Structured Analysis?, *Datamation*,**32**(1), 133-138

APPENDIX I: Level of Awareness of Methods for Systems Development in South Africa

TOOLS	%	TOOLS	%
INTERVIEWS	100	CROSS-IMPACT ANALYSIS	35
OBSERVATIONS	100	HOS CHARTS	22
DOCUMENTATION REVIEW	100	PSEUDOCODE	90
PRESENTATION	100	INSPECTION	90
BRAINSTORMING	100	ENTITY-RELATION DIAGRAMS	90
DATA FLOW DIAGRAMS	100	JACKSON DIAGRAMS	87
SYSTEMS FLOWCHARTS	100	STRUCTURE CHARTS	87
DOCUMENTATION	100	INPUT-OUTPUT TABLES	87
FEASIBILITY STUDY	100	OPTIMIZATION TECHNIQUES	87
COST BENEFIT ANALYSIS	100	NON-PROC.LANGUAGES	86
DECISION TABLES	98	GANTT CHARTS	85
STATISTICAL ANALYSIS	98	STRUCTURED ENGLISH	83
QUESTIONNAIRES	98	PROGRAMMING WORKBENCH	83
CODE-GENERATORS	98	LINEAR PROGRAMMING	81
STRUCTURED WALKTHROUGH	98	FUNCTIONAL DECOMPCHARTS	81
DATA DICTIONARIES	98	BUBBLE CHARTS/DE MARCO	77
BENCHMARKS	96	HIPO DIAGRAMS	77
PROTOTYPING	96	ANALYSIS WORKSTATION	71
ON-LINE DEVELOPMENT AIDS	96	VALUE ANALYSIS	71
PERT/CPM	94	QUEUEING THEORY	62
DECISION TREES	94	SIMULATION LANGUAGES	60
REPORT GENERATORS	94	SENSITIVITY ANALYSIS	58
DOCUMENTATION TOOLS	94	WARNIER-ORR DIAGRAMS	52
SAMPLING TECHNIQUES	92	DELPHI	52
DATA NORMALIZATION	92	ENTITY LIFE HISTORY	48
DATABASE DESIGN AIDS	92	SCENARIO WRITING	42
SAPIENS	37	GAMING THEORY	42
GRIDCHARTING	35	MONTE CARLO	38

APPENDIX II. Incidence of Usage of Methods by Respondents in descending order

TECHNIQUES/TOOLS	%(1)	%(2)	TECHNIQUES/TOOLS	%(1)	%(2)
DOCUMENTATION	100	60	DOCUMENTATION TOOLS	54	25
FEASIBILITY STUDY	100	25	DATABASE DESIGN AIDS	44	10
PRESENTATION	98	31	OPTIMISATION METHODS	44	2
DOCUMENTATION REVIEW	98	29	VALUE ANALYSIS	37	8
INTERVIEWS	98	73	STRUCTURED ENGLISH	37	2
OBSERVATIONS	96	23	JACKSON DIAGS.	33	8
DATA FLOW DIAGRAMS	94	33	BUBBLE CHARTS/DEMARCO	29	0
SYSTEMS FLOWCHARTS	92	33	PSEUDOCODE	29	2
BRAINSTORMING	92	12	DECISION TREES	29	0
DATA NORMALIZATION	88	44	ANALYSIS WORKSTATION	27	10
PROTOTYPING	88	12	CODE-GENERATORS	27	8
STRUCTURED WALKTHROUGH	87	21	NON-PROC. LANGUAGES	25	8
COST BENEFIT ANALYSIS	83	25	ENTITY LIFE HISTORY	23	10
DATA DICTIONARIES	81	50	LINEAR PROGRAMMING	21	0
INSPECTION	79	8	SENSITIVITY ANALYSIS	19	0
ENTITY-RELATION DIAGS.	71	29	PROGRAMMING WORKBENCH	19	8
ON-LINE DEV. AIDS	65	25	QUEUEING THEORY	17	2
BENCHMARKS	65	2	SCENARIO WRITING	13	0
FUNC. DECOMP.CHARTS	63	25	HIPO DIAGS.	13	4
REPORT GENERATORS	62	13	DELPHI	12	0
STRUCTURE CHARTS	60	19	CROSS-IMPACT ANALYSIS	10	2
SAMPLING TECHNIQUES	58	2	GRIDCHARTING	10	2
PERT/CPM	58	13	WARNIER-ORR DIAGS.	10	2
INPUT-OUTPUT TABLES	56	4	SIMULATION LANGUAGES	8	0
GANTT CHARTS	56	13	SAPIENS	4	4
DECISION TABLES	56	2	MONTE CARLO	4	0
QUESTIONNAIRES	54	2	HOS CHARTS	2	2
STATISTICAL ANALYSIS	54	4	GAMING THEORY	2	0

(1) Usage measured a percentage of all respondents who sometimes used the tool
(2) Usage measured a percentage of all respondents who always used the tool

APPENDIX III. Degree of Automation of Systems Development Methods

TOOLS	%	TOOLS	%
SAPIENS	100	SENSITIVITY ANALYSIS	20
ANALYSIS WORKSTATION	86	PRESENTATION	20
DOCUMENTATION TOOLS	79	WARNIER-ORR DIAGS.	20
SIMULATION LANGUAGES	75	DELPHI	17
ON-LINE DEV. AIDS	74	VALUE ANALYSIS	16
REPORT GENERATORS	72	HIPO DIAGS.	14
PROGRAMMING WORKBENCH	70	COST BENEFIT ANALYSIS	14
DECISION TABLES	69	SCENARIO WRITING	14
DATA DICTIONARIES	67	STATISTICAL ANALYSIS	14
CODE-GENERATORS	64	DECISION TREES	13
DATABASE DESIGN AIDS	61	PSEUDOCODE	13
GANTT CHARTS	59	QUESTIONNAIRES	11
NON-PROC. LANGUAGES	54	STRUCTURED WALKTHROUGH	11
PERT/CPM	50	STRUCTURED ENGLISH	11
DATA FLOW DIAGS.	47	QUEUEING THEORY	11
ENTITY-RELATION DIAGS.	46	SAMPLING TECHNIQUES	10
LINEAR PROGRAMMING	45	BENCHMARKS	9
FUNC. DECOMP. CHARTS	45	INTERVIEWS	8
GRIDCHARTING	40	BUBBLE CHARTS/DEMARCO	7
STRUCTURE CHARTS	32	JACKSON DIAGS.	6
DOCUMENTATION	31	FEASIBILITY STUDY	6
DATA NORMALIZATION	30	OBSERVATIONS	2
PROTOTYPING	30	DOCUMENTATION REVIEW	2
SYSTEMS FLOWCHARTS	29	INSPECTION	2
MONTE CARLO	25	BRAINSTORMING	0
INPUT-OUTPUT TABLES	24	GAMING THEORY	0
ENTITY LIFE HISTORY	23	CROSS-IMPACT ANALYSIS	0
OPTIMISATION METHODS	22	HOS CHARTS	0

APPENDIX IV. Percentage of respondents using Method as part of a formal Methodology

TOOLS	%	TOOLS	%
ANALYSIS WORKSTATION	71	SENSITIVITY ANALYSIS	30
ENTITY-RELATION DIAGS.	65	COST BENEFIT ANALYSIS	30
FUNC. DECOMP. CHARTS	64	SCENARIO WRITING	29
DATA DICTIONARIES	64	HIPO DIAGS.	29
DATA NORMALIZATION	63	STATISTICAL ANALYSIS	28
NON-PROC. LANGUAGES	62	INPUT-OUTPUT TABLES	28
DATABASE DESIGN AIDS	61	SAMPLING TECHNIQUES	27
DATA FLOW DIAGS.	61	DOCUMENTATION REVIEW	27
CODE-GENERATORS	57	OPTIMISATION METHODS	26
DOCUMENTATION TOOLS	57	STRUCTURED ENGLISH	26
INTERVIEWS	55	REPORT GENERATORS	25
VALUE ANALYSIS	53	JACKSON DIAGS.	24
SIMULATION LANGUAGES	50	OBSERVATIONS	24
DOCUMENTATION	48	DECISION TABLES	24
GANTT CHARTS	48	BRAINSTORMING	23
STRUCTURED WALKTHROUGH	47	QUEUEING THEORY	22
PERT/CPM	47	QUESTIONNAIRES	21
PROTOTYPING	43	BUBBLE CHARTS/DEMARCO	20
PRESENTATION	41	DECISION TREES	20
ENTITY LIFE HISTORY	41	CROSS-IMPACT ANALYSIS	20
FEASIBILITY STUDY	40	WARNIER-ORR DIAGS.	20
PROGRAMMING WORKBENCH	40	GRIDCHARTING	20
SYSTEMS FLOWCHARTS	38	BENCHMARKS	18
INSPECTION	37	DELPHI	17
ON-LINE DEV. AIDS	35	LINEAR PROGRAMMING	9
STRUCTURE CHARTS	35	MONTE CARLO	0
PSEUDOCODE	33	GAMING THEORY	0

Book Reviews

Functional programming using Standard ML
by Åke Wikström, Prentice-Hall, 1987.
Reviewer: Mr L Conradie, University of Pretoria.

Over the past 10 years functional programming has progressed from being a small, curious branch of computer science to one of the foremost research areas in programming language design and theoretical computer science. In *Functional programming using Standard ML*, Åke Wikström discusses one such language, namely Standard ML (SML).

The book can be divided into two parts. The main text of the book provides an introduction to SML. The second part, in the appendixes, contains the standard reference manual to the SML language, written by Robin Milner and Robert Harper.

Functional programming in Standard ML is primarily aimed at the novice functional programmer, and therefore the mathematical theory of functional programming is not covered in as much detail as in most other books on the subject. The lambda calculus is given only a brief mention, and the formal aspects of type inference are completely left out. The book is a good introduction to some of the more advanced texts on the subject, such as those of Chris Reade and Anthony Field.

While SML is an impure functional language with many imperative features, Wikström examines only the purely functional subset of the language (with the exception of Input/Output). This results in topics such as exception handling and destructive assignment operators receiving scant coverage in the main part of the book. While purists may regard this as a plus point, this reviewer feels that at least a chapter should have been devoted to these subjects. Fortunately, these subjects are covered in the SML guide at the end of the book, so they are not completely forgotten.

A more serious shortcoming stems from SML being an eager language. This leads to the fact that lazy evaluation, which is one of the most important advantages of functional programming, is skimmed rather quickly.

The main section of the book is divided into 23 chapters, which can be grouped into seven sections. The first section, Chapters 1 and 2, is strictly for beginners. Chapter 1, entitled Computers, Data and Programs covers such basics as symbols, numbers, and what computers are. Chapter 2 deals with simple numeric functions.

The second section, Chapters 3 to 5, covers basic SML concepts such as naming, functions and running programs on the system. Chapter 5, entitled Systematic Program Construction, is about program design in a functional language. It also covers debugging and testing of programs.

Chapters 6 to 8 are about the basic data types of SML, namely truth values, characters and tuples, respectively. These chapters form the basis for the rest of the book. Sections on polymorphic types and lazy evaluation are to be found here.

The next three chapters present a more formal coverage of SML. Chapter 9 covers the syntax of the language. It explains concepts such as lexical structure, BNF and context dependence. Chapter 10 presents SML semantics from an operational point of view, while chapter 11, entitled Declarations, covers the language's scope rules and declaration syntax. These three chapters are the most technical in the book.

In the next section, consisting of chapters 12 to 18 more advanced functional programming concepts are covered, such as pattern matching, list handling, recursion and higher-order functions. Chapter 18 deserves mention for its coverage of the classification of functions into primitive, general and partial recursive functions. The relationship between iteration and recursion in the form of tail recursion is also covered in this chapter. The chapter is a very understandable introduction to these concepts.

Chapters 19 to 21 deal with concrete, recursive and abstract data types. ML's type inference system is also covered in this section, although only from the programmer's point of view. The theoretical aspects, such as the unification algorithm used, is left out. These chapters contain the most extensive examples, covering lists, binary trees, set graphs and a host of other data structures.

The book concludes with two chapters on I/O and programming methodology. The chapter on I/O is the only one covering the imperative aspects of SML.

The appendixes contain the standard report on The Standard ML Core language, written by Robin Milner, one of ML's chief designers. The exception is the section on I/O, which was written by Robert Harper. It also contains syntax diagrams for a subset of SML as well as a history of the development of the language. The reference guide is written in a much more formal style than the rest of the book and is therefore perhaps a bit advanced for beginners. This section is the standard reference to SML and should be used by all SML programmers and implementors alike.

Wikström has written a very good introduction to programming in the functional subset of SML. Although the intended audience of the book means that some important topics are not covered in any detail, the book should provide an adequate base for the more advanced texts in which these subjects are covered in more detail. Lazy evaluation and the lambda calculus are, in any

case, not subjects for beginners.

Functional programming using Standard ML is a well written book recommended for anyone interested in this exciting field of computer science. It is written in a very readable style and contains many examples which will aid the beginner.

Equations, Models and Programs: A Mathematical Introduction to Computer Science
by Thomas J Myers, Prentice-Hall, 1988.
Reviewer: Mr L Conradie, University of Pretoria

Since computers make up such a big part of our everyday lives, the basic concepts behind the science of computers are important. In this book, *Equations, Models and Programs* (*EMP*), Myers discusses these basic concepts from a mathematical point of view.

EMP is divided into three parts. In the first part, *Equations*, a language for explicit program description is presented. In the second, *Models*, examples of modelling and problem solving are covered. The third part, *Programs*, evaluates techniques for developing programs from specifications.

EMP is intended for beginners in the field of computer science with a little programming experience in BASIC or LOGO and a common high-school mathematics background in algebra, geometry and preferably trigonometry and analytic geometry. A chapter for instructors is included in the introduction, so that *EMP* can be used as textbook.

The book is written in an equational style so as to make it easier for the reader to understand the "programs" that are presented. The restriction is that, if the leader wants to implement the programs presented within the first seven chapters of the book, he must have at his disposal a programming system which allows an equational style of programming. This problem is resolved in the succeeding chapters where Myers presents these equations in MODULA-2. Consequently, anybody who has a similar compiler (say Pascal or C) at his disposal can implement the programs.

Part 1, *Equations*, is divided into three chapters. In chapter 1 the focus is on the computational process and the equations which are used for it. Chapter 2 covers the testable properties of programs that manipulate data such as the cardinal numbers, real numbers and logical values. Chapter 3 discusses structures of the data, such as sequences, bags and sets.

Part 2, *Models*, is divided into four chapters (4-7). In chapter 4, physical maps and space-coordinates are introduced - in other words, not just spaces but anything one can measure. Chapter 6 discusses systematic state-space search and chapter 7 concludes part 2 by tying the previous three chapters together and introducing formal reasoning about models and computations.

Part 3, Programs, consists of chapters 8 to 10. Chapter 8 is focused on program development in MODULA-2. Transliteration from equations to programs and the proper use of modules is shown. In chapter 9 the various methods of presenting data-structures are discussed. Some of these presentations require a knowledge of the lower levels of the programming language. These are dealt with in chapter 10 where every layer, from the bottom up, is described. This description is carried through to third and even fourth generation languages.

Equations, Models and Programs is a well written book that contains many examples. It teaches the reader the importance of equational programming and aids the reader in understanding computer science from a mathematical viewpoint.

Communications and Concurrency *by Robin Milner, Prentice-Hall, Hemel Hempstead, 1989.*
Reviewer: S Hazelhurst, University of the Witwatersrand

Milner's Communications and Concurrency is the most tantalising computer science book that I have read. It offers a profound insight into the specification and behaviour of concurrent systems. It promises a rigorous method of building concurrent systems which behave in a predictable and well-behaved manner.

Crudely, the practical application of the process calculus proposed is as follows. Using the process calculus, the desired behaviour of the system is specified. Next, the system is constructed using operators and combinators with well-known semantics, and finally, the implementation is shown equivalent to the specification.

It sounds fairly easy. Unfortunately it is not. The specification of even modest systems is difficult. To show that a specification is equivalent to an implementation is not difficult, but extremely tedious. This was my disappointment with the book - not so much a reflection on the book as on the difficulty and vastness of the area.

There are three key issues which Milner examines: the semantics of a process calculus which allows the specification of agents; the concept of equivalence of agents; and the concept of determinacy, which allows the well-defined behaviour of the system. The semantics of the process calculus are built using operational semantics to define the meaning of its basic combinators and operators. Near the end of the book, Milner examines alternative calculi, and in particular, shows that a synchronous calculus, which has greater simplicity, can be used to define the calculus. Milner also compares the process calculus to Hoare's logic and the work of others.

To get sensible answers from questions such as "is this implementation equivalent to this specification", the the meaning of equivalence is critical. Milner defines a range of equivalences. The first point that needs to be

dealt with is the treatment of internal actions: sub-components of systems may communicate with each other thereby changing the state of the system without any external intervention. Depending on whether internal actions should be ignored or not, the appropriate equivalence should be used.

Another concern is the degree of similarity we expect from the agents. For example, trace equivalence - one of the weakest forms of equivalence - states that two agents are equivalent if the ability of one agent to perform a sequence of actions implies that the other agent can (and vice-versa). This sounds adequate, but it is not. It is not able to characterise the ability of agents to make different choices. One consequence of this is that trace equivalence does not respect deadlock. The fact that P is trace equivalent to Q, and that agent P can perform actions *abcd* in sequence just means that Q can also perform actions *abcd*. However, sometimes (with the same external stimuli), Q might not be able to perform *abcd* but go into deadlock, even if P never goes into deadlock on those actions.

In the last part of the book, Milner attempts to show how practical systems can use the process calculus. One part of this is showing that the semantics of a simple, concurrent, imperative language can be defined using the calculus. The second is the exploration of determinacy and confluence - which are desirable properties, since they allow the construction of systems with predictable behaviour. He shows that if a system is built using restricted forms of the calculus, many of the undesirable effects of non-determinism can be avoided.

Milner sees this book as a beginning. Successful prosecution of this research will provide valuable advances both to the theory and practice of computer science. In the latter respect, mechanisation of the findings and of proofs of equivalence is essential if the theory is to be practicable

There are two criticisms in particular of the book. The first is that in some chapters there were minor typographical errors which cause confusion. The second is the notation. The notation needs to be simplified and rationalised. In some cases, notations are overloaded with potentially horrendous effect.

This is not a book for light reading. From the perspective of someone who is not a specialist in this area, the first chapter can be managed in a few hours and gives some useful insight. The rest of the book needs considerable effort to get through if value is to be derived from it. There are a number of exercises in each chapter. The completion of these exercises is essential to a proper understanding of the book. Unfortunately, they take time, and can be tedious.

This is a suitable text-book for a course on communication and concurrency, probably at the MSc rather than Honours level. As a contribution to understanding of communication and concurrency, this is well with the effort in reading.

Editorial Comment

For those who are interested in the health and status of SACJ I am pleased to report that, in general, things are going extremely well: there has been a steady flow of good contributions to the journal; long-suffering referees have been prompt and co-operative; the journal's finances are healthy (thanks largely to generous sponsorships); and there has been a small but steady growth in readership. I trust that you, as a reader, feel equally positive about the journal and that you have been finding something of interest in the various issues to date.

Inevitably, there are a few areas of concern, some of greater importance than others. Over the past two years a matter of particular concern to me has been the the recognition of SACJ contributions for state subsidy purposes. In April 1989, I submitted an application for recognition via the relevant authorities at my university. I will spare the reader all the gory details of the bureaucratic inanities that ensued. The bottom line is that, despite my best efforts, SACJ did not appear on the list of approved journals for 1990. Although I have not been able to contact anyone who can say precisely why approval was withheld, I have been told repeatedly that it would have been far easier to obtain approval had I indicated that SACJ is merely QI with a changed name, since QI was already on the approved list. That the evaluation of journals has been reduced to such system-beating heuristics disturbs me profoundly. I have conveyed my feelings in an angry letter to the director of research administration at my university, with a request that my views be passed on to the relevant authorities. Here is an extract from the letter:

"I must express my extreme displeasure about the hidden nature of the evaluation process, to the point where the committee (does it have a name?) is apparently answerable to no-one, and does not have to provide any reason whatsoever for its decisions. It appears that it does not even need to affirm that it has rejected an application. It seems to me ludicrous that I could possibly get by whatever their objection might be, merely by labelling SACJ a name-change from QI. This somehow makes what was previously unacceptable, now academically worthy. What a farce! [We] are being told that since we cannot hold our heads up high and go in the front door, we must surreptitiously cover our heads in shame and go through the back."

Unless I have blown it by not being sufficiently diplomatic about the matter, contributors to SACJ need not despair. By applying (or re-applying) for a subsidy during 1991, it seems that there is a good chance that the subsidy will be approved, provided that you clearly indicate that SACJ is a name-change from QI. I suggest that you do this, whether you believe it to be true or not!

Derrick Kourie
Editor

Notes for Contributors

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

Form of Manuscript

Manuscripts for review should be prepared according to the following guidelines.

- Use double-space typing on one side only of A4 paper, and provide wide margins.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.
- References should be listed at the end of the text **in alphabetic order** of the (first) author's surname, and should be cited in the text in square brackets. References should thus take the following form:

  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.

  [3] S Ginsburg, [1966], *Mathematical theory of context free languages,* McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may be provided in one of the following formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- as a **WordPerfect**, **TEX** or **LATEX** or file; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies. A styles file is available from the editor for Wordperfect, **TEX** or **LATEX** documents.

Charges

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

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

Proofs

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

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

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

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

Advertisement

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

South African Computer Journal

Number 4, March 1991
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 4, Maart 1991
ISSN 1015-7999

Contents