ISSN 0254–2757

QUÆSTIONES INFORMATICÆ

Volume 6 • Number 4 April 1989

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor

Professor D G Kourie
Department of Computer Science
University of the Pretoria
Hatfield
0083

Production
Mr Q H Gee
Department of Computer Science
University of the Witwatersrand
Johannesburg
Wits
2050

Subscriptions

The annual subscription is

	SA	US	UK
Individuals	R20	$ 7	£ 5
Institutions	R30	$14	£10

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

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Editorial

by

Derrick Kourie

It is my privilege to have been requested by the SAICS executive to take over the post of editor of QI from Professor Judy Bishop. I think it is in order to thank her on behalf of the readership for the fine job she has done in boosting the quality of the journal during her brief but effective term. It is also appropriate to thank the production editor, Quintin Gee, for his substantial role in producing the journal. I am grateful that he is still in the post, and for all the support and work that he continues to do.

My job as editor is directed towards the overall goal of serving the South African academic community in the various computer-related disciplines in particular, and the computer industry in general. A number of objectives which support this goal include

- ensuring that high quality papers are published, thereby providing a display window for computer-related research in South Africa
- boosting local and international circulation of the journal both within the academic community and in the computer industry at large, thereby promoting a fruitful interchange of ideas
- attempting to do this in a cost-effective fashion so that the limited financial resources of SAICS and the CSSA may be released (perhaps even modestly augmented) to promote their various other service-orientated activities.

A number of measures are planned which are intended to meet these objectives. I shall mention some of them below, while others will become manifest with the passage of time.

After much debate it has been decided to change the name of this journal from Quæstiones Informaticæ to The South African Computer Journal/Die Suid-Afrikaanse Rekenaartydskrif. It will be abbreviated to SACJ in English and SART in Afrikaans. Arguments against this name change include the conciseness and uniformity of reference in both official languages provided by QI, and a certain kind of catchiness to the name. Those in favour of the name change regard the new proposal as being more descriptive for ordinary mortals (i.e. non-Latin scholars), less pretentious, and therefore more inviting for a wider audience. The fact that the new title identifies the journal as South African is also regarded as important. Many readers would, I surmise, be fairly neutral about the name and adopt a philosophical "a rose by any name" position. Perhaps the divide is between those who opt for a high level of abstraction and information hiding, and those who feel that a measure of refinement is necessary.

Regarding the quality of papers, I shall continually strive to ensure that papers submitted are reviewed by at least two relevant and competent specialists. It is appropriate here to thank all those who have so enthusiastically reviewed papers to date. This is a time-consuming, altruistic, backroom task, with very little explicit reward. To ensure that the burden is spread more equitably, I would like to appeal to readers to suggest additional names of people who could be approached for reviewing. Names of overseas contacts would be particularly useful.

I should also like to invite as much reader-participation in the journal as possible. There are several levels at which this may be done. The most obvious is by way of letters to the editor. Many people out there have strong ideas about a variety of subjects. In the absence of a decent national network facility (perhaps someday!), please feel free to use SACJ as your soapbox.

However, it is also evident that many people read many books for a variety of purposes. Why not share these insights by submitting book reviews to the journal, particularly with respect to books which could be prescribed for courses? If there are any book publishers or distributors out there who perchance may read this editorial, perhaps you should make inspection copies to lecturers contingent on a review being provided to SACJ!

I would also encourage researchers to continue providing a steady stream of research papers to the journal. Clearly, SACJ is in competition with other international journals for your research results. However, this is not a head-on competition. While it would be sheer hubris to pretend that SACJ is precisely equivalent to one of the more prestigious overseas publications, there are considerations which argue in favour of submitting certain kinds of research to SACJ. First, SACJ will be dedicated to providing a quick turnaround in reviewing and publication. Hence, it is an ideal forum for presenting and testing interim research results, and even for quickly assuring your stamp on potentially important ideas which you hope to flesh out later. Secondly, SACJ is the obvious forum to use for locally relevant research. Finally, and quite candidly, the competition for publication in SACJ is obviously not as intense as in a more prestigious international journal. However, I need to be most

explicit on the implications of this latter point.

SACJ should not be seen as a soft option in the sense that quality will be sacrificed. By this I mean that on some arbitrary scale of quality measurement, if CACM contains papers above say the 95% percentile, then SACJ should fall into about a 60% percentile category. Put differently, there is clearly a gap to be filled that lies somewhere between poor, inferior drivel and outstanding research contributions – a gap which SACJ will seek to fill. Papers will therefore be rigorously reviewed, and every effort will be made to ensure that the journal is worthy of international recognition – even if such recognition does not come about immediately. This is not the impossible task that some might consider it to be. There are several South African scientific journals that already enjoy a measure of international recognition (the South African Statistical Journal – to name but one). Furthermore, it is my perception that many of our academics who travel overseas discover – perhaps slightly to their amazement – that they are well able to hold their own with academics at peer institutions. This suggests that there is probably sufficient brain power, research ability and research activity in the country to ensure that the goal of international recognition is attained.

As for the cost-effective functioning of SACJ, two points need to be made. First, SACJ will be available for a limited amount of advertising at R1000 per page and R500 per half-page. The computer industry and book publishers might wish to avail themselves of this offer, as might universities and employment agencies. Enquiries in this regard should be directed to Quintin Gee. Secondly, a modest charge per page (indicated elsewhere in this edition) will be levied on accepted research papers. This has become standard practice for most journals, the rationale being that the SACJ is one of the journals which counts for state subsidy purposes. However, the editor will have the right to waive such charges in deserving cases, as for example in the case of an author from industry whose company is unwilling to provide the financial support.

Ultimately then, SACJ will critically depend on your support. It will become what you, the reader, researcher and reviewer, make it. In a sense the South African Computer Journal will expose you, the South African Computer Academic, to the outside world without a single Latin phrase to hide behind.

Software Design to Meet Third World Requirements: An Experimental Software Engineering Approach

Philip Machanick
Department of Computer Science, University of the Witwatersrand, Johannesburg, 2050 Wits

Abstract

Appropriate technology refers to technology appropriate for use in less developed parts of the world, especially the Third World; this paper raises some problems in adapting a definition of appropriate technology to computer software. A partial solution, a strategy called experimental software engineering, is introduced. The potential of this solution is demonstrated by a case study, in which software for medical education is developed. The result is a clearer understanding of both appropriate technology and design of software for usability.
***Keywords:** appropriate technology, software engineering, human-computer interaction, medical education*

Received September 1988, Accepted January 1989

1. Introduction

Computers are becoming increasingly widely used in technologically developed countries, such as the USA. An important part of this trend is the migration of computers to contexts where users are technically naïve. This move of computers to the mass market has yet to impact less developed parts of the world. For example, Gambia has been reported as having no computer dealers at all in 1987 [5].

This paper examines some problems in fitting computers to needs of third world countries, from the starting point that dropping prices must eventually overcome the problems such as lack of foreign capital which inhibit the distribution of computers to poorer parts of the world. The fact that computer hardware is becoming more affordable does not mean computers will meet the needs of a third world country: the needs of the new society should be taken into account.

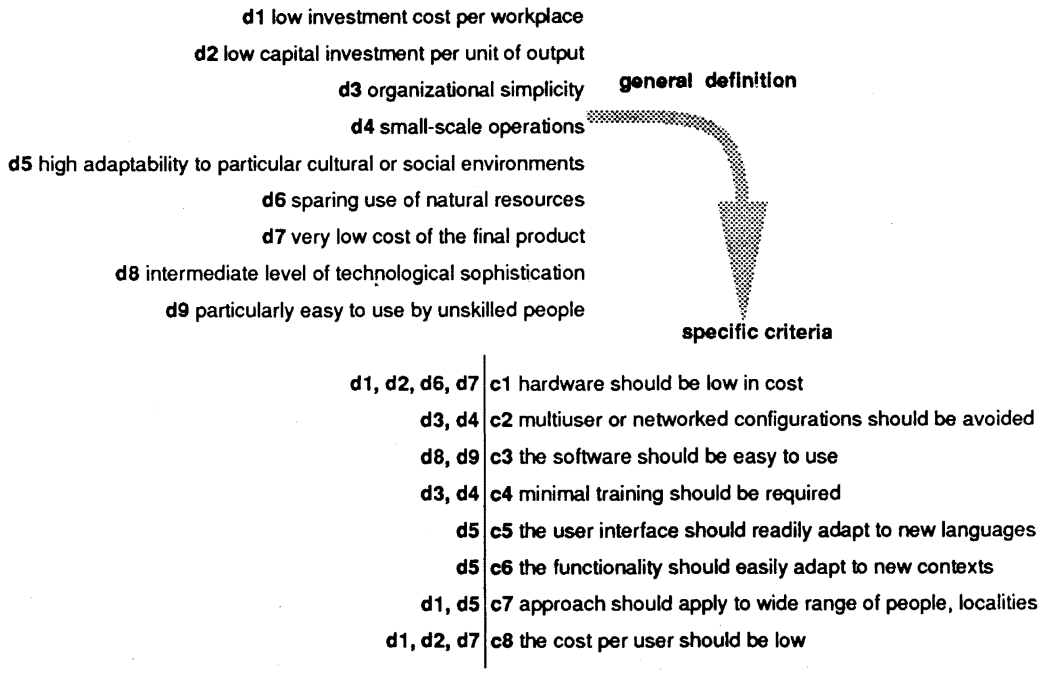
Much research into adapting other technologies to the third world has already been undertaken [9, 16]. Appropriate technology is the name broadly given to technologies considered suited to less developed societies. Mostly, "appropriateness" is measured by simplicity, and avoidance of high capital costs. Where computers are concerned, some thought needs to be given to the definition of appropriateness, since computers are a relatively advanced technology yet they are increasingly becoming affordable. In order to simplify the problem, the issue addressed here is how computer technology may be made to fit a specific, accepted definition of appropriate technology. This paper is further restricted in its scope in looking at a case study of a single application. In addition, the reasons for choosing the application domain are not presented.

The major focus of the paper is introducing and evaluating a strategy called experimental software engineering (ESE). ESE is a strategy for deriving the requirements when they are not clear, especially when the users have had little exposure to computers. ESE emphasizes usability, and placing the user in control of both the requirements and the finished product.

The next section looks in more detail at how a definition of appropriate technology may be applied to computers. In particular, criteria for deciding whether a computer approach is appropriate technology are presented. The following section introduces ESE, and explains how it relates to the problem of meeting the criteria for appropriate technology. From this start, a case study, in which the requirements for a tool for medical education are developed, is presented to illustrate how ESE may be applied. This case study is used to demonstrate how ESE brings out issues of usability, and gives the user control of the development process. In conclusion, ESE is evaluated in terms of appropriate technology concerns, and lessons which apply more generally to software engineering are considered.

2. A Definition of Appropriate Technology

Appropriate technology is technology specially adapted or designed for less-developed (especially third world) countries. Mostly, research into appropriate technology has revolved around low technology industries and agriculture. It is not obvious how experience of this kind may adapt to a relatively sophisticated technology such as the computer. For this reason, some attention is given to adapting an existing definition of appropriate technology to this new area.



A definition of appropriate technology [9] is used to derive criteria directly related to computer applications.

Figure 1: Criteria for computers as appropriate technology

A specific definition, presented in figure 1, is used as a starting point. This section derives criteria for evaluating computers as appropriate technology from this definition. Derivation of these criteria as illustrated in figure 1 is not presented in detail here (see [13]), since the major thrust of this paper is a description of the software development strategy employed. Instead, the implications of the criteria are considered here.

The first criterion (c1) – that the hardware should be low in cost – is derived from no fewer than four points of the definition, yet it is considered the least restrictive. Computer hardware is continually dropping in price; today's expensive workstation is tomorrow's personal computer. Even in an area as sophisticated as artificial intelligence (AI), tools for low-cost equipment such as IBM PCs and Apple Macintoshes are beginning to rival those on research machines of the last decade [12]. Low cost, then, should be seen as relating to a specific project, rather than as an inhibiting factor on research.

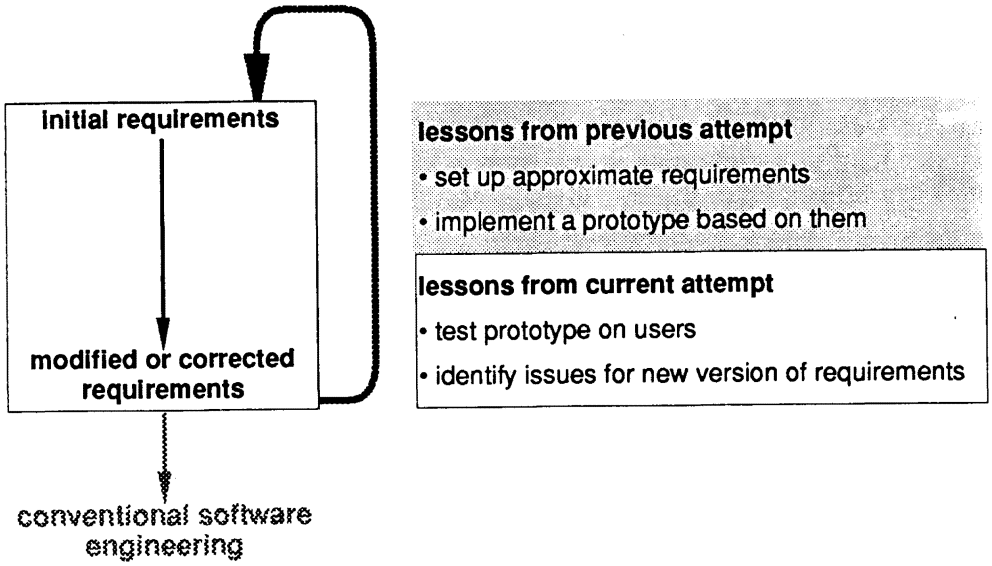
The next criterion (c2) relates specifically to reducing the complexity of the equipment. Networked or multiuser configurations will not always be complex – so exceptions should be allowed. For example, the AppleTalk network is a very cheap and simple way of sharing resources such as laser printers. However, even such a simple network can become complex to use when more sophisticated services such as file servers are added [11].

The next criterion (c3) – the software should be easy to use – requires some thought, since ease of use is not trivial to define. This point is further addressed in the rest of the paper. If the ease of use criterion can be met, the criterion of requiring minimal training (c4) should not present serious problems. Both criteria need to be specified concretely, in the context of a specific project. For example, when Apple launched the Lisa (the predecessor of the Macintosh), it was claimed that a novice could learn to use the machine in half an hour. An experiment has found that this claim is not literally true [6] – although some critics of the experiment have complained that its findings are unfair in some respects [10, 15].

The next three criteria (c5 to c7) relate to adaptability. This fact that three criteria are devoted to adaptability indicates how important this issue is. First world countries can generally rely on large markets to absorb development costs; third world countries often either have small populations or wide regional differences.

The final criterion (c8) – the cost per user should be low – also relates to spreading the usability of the software as widely as possible. A diskette costs a few cents, and hardware is becoming cheaper. Clearly, any measure of the cost per user must be heavily influenced by how widely the cost of the development of the software can be spread.

The criteria can be summarized in two major points: placing the user in control and spreading the usefulness of the software as widely as possible. Both of these points are captured (if not in full) in the concept of *usability*.



Each iteration leads to further clarification of the requirements, based on experience with a prototype for the current version of the requirements. Ultimately, should the requirements be sufficiently clear, a conventional software engineering strategy can take over. Ideally, issues elucidated should help in later projects as well.

Figure 2: Experimental software engineering

3. Experimental Software Engineering

One of the most difficult issues in software engineering is accurately capturing the user's requirements [24]. The user usually does not have a clear understanding of what a computer can do, while the software engineer may not have much knowledge about the application domain. In the context of appropriate technology, this problem is exacerbated in two significant ways.

Firstly, there is little experience in implementing software as appropriate technology. Such work as there has been has either been in the form of isolated projects [1], or of investigations into general policy issues [16]. No serious attention has been given to the problem of the issues in software engineering of meeting criteria for appropriate technology.

Secondly, potential users have had little exposure to computers, which increases the already serious difficulty of obtaining accurate requirements from the users.

In defining a strategy for dealing with these problems, two sources are drawn on in this research: other experience with software engineering in which usability has played a major role, and experience in an area in which software is created without a clear initial idea of the requirements. The example of usability is the 1984 Olympic Message System; the latter example is programming in artificial intelligence.

The Olympic Message System – which allowed athletes and other interested parties to leave messages for each other – was designed with usability as a central issue. The underlying philosophy was one of using behavioural measures to ascertain the acceptability of the system to the user. Although large numbers of people were used to test the system at later stages, the methodology was relatively informal. The strategy used deviated from the classical model of the software life cycle – sometimes known as the waterfall model [2]. The design and requirements analysis were carried through to relatively late stages. As features were implemented, they were tested on potential users for acceptability, and the system was changed as problems were identified [8]. This is in contrast to the waterfall model, in which software development is seen as consisting of largely non-overlapping stages.

The Olympic Message System is an important example, because it illustrates how far system design and construction can be driven by usability, especially in a nontrivial program. The users were from a wide range of backgrounds, and the software worked well and was accepted positively by the users [*ibid.*]. However, the potential of applying the experience of this project directly to other areas is limited in two respects. The initial functionality was reasonably clearly specified – the details were the real problem. Also, the implementers had access to a table-driven tool for generating user interfaces, which considerably increased their flexibility in changing the design. This tool was specifically intended for communications applications; such a tool may not necessarily be easy to construct for other applications.

AI is an interesting source of ideas for dealing with the issue of constructing software where the initial idea of the requirements is extremely vague. Some AI researchers view programming in AI as an experiment, in which the requirements are clarified.

r1 free switching between gathering evidence and forming, confirming and rejecting hypotheses should be supported
r2 no order should be imposed on the specific hypotheses and evidence which are considered
r3 association of hypotheses and evidence should be explicit
r4 an ordered record of the hypotheses and evidence considered should be kept
r5 the system should be non-judgmental
r6 solving of problems meaningful to the learner should be supported
r7 the student should supply medical knowledge for solving a given problem by some approximation to building an expert system – the system should not contain any medical knowledge
r8 ease of use should be emphasized; this implies that user interface may become a major issue
r9 inference should be avoided: the student, not the program, should find the solution

These requirements, which were the starting point for the experimental software engineering strategy, are based on the educational approach called knowledge engineering based learning (KEBL) as well as the criteria for appropriate technology of figure 1.

Figure 3: Initial requirements

AI is seen by some as an empirical science [17], in which programs produced by successive researchers form data points in a grand experiment [4]. The long-term outcome is an enhanced understanding of human intelligence and what computers can do. Programming in an experimental fashion requires powerful tools and techniques. Examples include the Interlisp programming environment [23] and structured growth, in which modules of a program may be rewritten as more sophisticated possibilities are considered feasible [21]. Some critics of AI have gone as far as to claim that the development of tools which support programming without a clear idea of the requirements is the major contribution AI has made [7]. Some lessons from AI research have found their way into software engineering environments, such as Cedar [22] and Pecan [20].

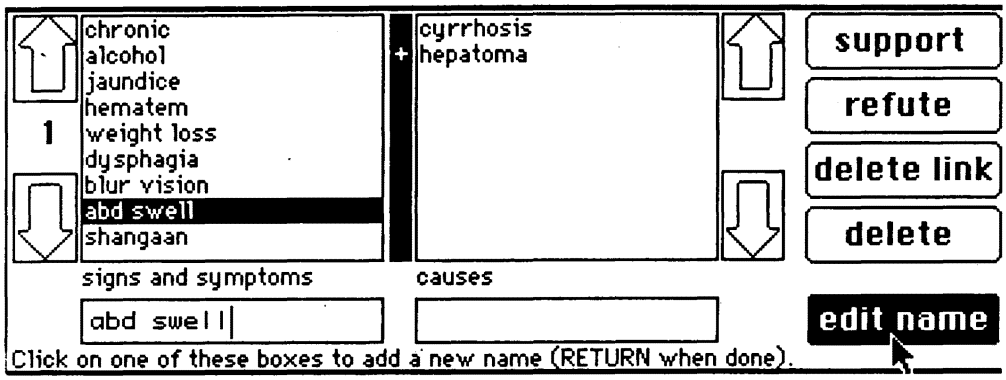
AI is however not always a suitable basis for constructing robust software, which is intended for widespread use. The lack of a precise specification of requirements makes testing difficult, and is likely to cause problems in the long term with maintenance. This claim is borne out by software engineering research which has measured the effects of replacing the requirements and design stages of the waterfall model by prototyping. The general finding is that the lack of formal documents cases problems with the later stages of the project, from integration onwards [3].

For developing software as appropriate technology, some of the ideas from the Olympic Message System and from AI can be put together. The first principle is to emphasize usability. To this end, behavioural measures of how the users relate to the software should be used. The second principle is that the requirements should be derived by a process of a succession of prototypes, each of which can be seen as a data point in an experiment to determine the correct functionality and user interface of the program. The earlier prototypes should be constructed using tools which allow maximum flexibility in changing the approach – in the AI tradition – while later prototypes should use increasingly rigorously specified languages in the spirit of software engineering. The outcome of this process, which is given the name *experimental software engineering* (ESE), is a precise specification of the requirements which may be used in a conventional software engineering project to implement the software.

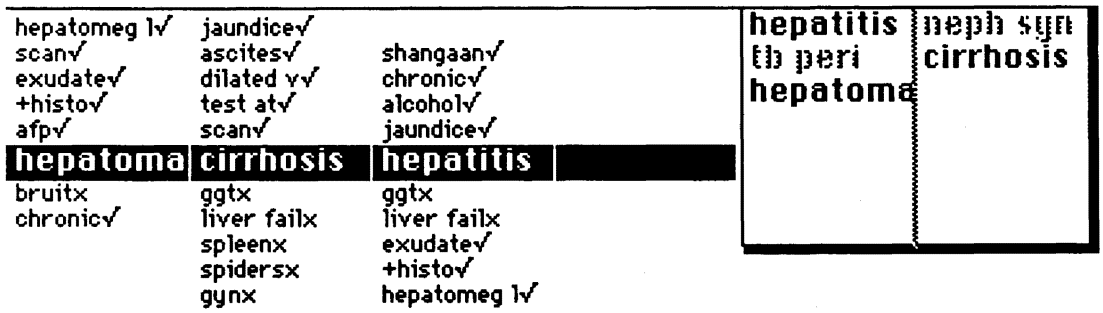
The ESE strategy is summarized in figure 2.

4. A Case Study: A Tool for Medical Education

The case study presented here illustrates how ESE may be applied in practice. The example used is the specification of requirements for a tool for medical education. An initial attempt at specifying requirements for this tool is based on the criteria for appropriate technology of figure 1. The tool is intended to be an approximation to an expert system shell, which will support learning medical problem solving in the same sense as Logo supports learning mathematical problem solving [18]. The philosophy is one of learning by doing, with the learner in control. The learner is placed in the role of a knowledge engineer, although in a much simplified version of an exercise in expert system construction. A full explanation of this step of the project is beyond the scope of this paper, which focuses on the application of ESE. This section presents a series of experi-



(a) The top half of the screen is used for adding, deleting and editing names – as well as making and breaking links. Here, a name is about to be edited.



(b) The bottom half of the screen is used for activating and deactivating causes (hypotheses), and marking signs or symptoms as present or absent. The section of the screen illustrated here contains a table of all causes on the right; those in grey are not currently active. The active ones appear in white on black writing in a report to the left of the table, with evidence for the above and evidence against below (based on the student's rules). A cause is activated or deactivated by selecting its name in the table.

Figure 4: The LISP prototype's user interface

ments which were carried out in developing the requirements, starting from the initial requirements in figure 3.

The first experiment was an initial attempt at investigating the implications of the requirements. The tool used was a program called CLASSIFY, a simple decision tree-like approximation to an expert system shell, which is supplied with Prolog-86 (an IBM PC implementation). The outcome of this experiment was used to validate the initial requirements; the requirements were then more fully tested using a program written in OPS5 on an Apple Macintosh (which is used for subsequent prototypes as well). The next version of requirements – in keeping with the philosophy of moving to tools which are increasingly useful for rigorously defining the behaviour of the program – was tested by an experiment with a prototype written in LISP. The final version of the requirements was prototyped in Pascal.

The CLASSIFY experiment mainly established communication with medical educators. A group of six people (three educators, two registrars and a medical student) which was presented with the KEBL idea was able to see merit in it, but agreed that CLASSIFY was not usable. The software proved to be limiting, and difficult to use. In particular, the fixed order in which the program asked questions based on its decision tree caused difficulty. Since this aspect of the program conflicted with the requirements, the finding was taken as confirmation that software conforming to the requirements should be constructed for another experiment.

The next prototype constructed was written in OPS5, in order to facilitate flexibility in changing the control strategy as new ideas were offered by medical educators. This prototype more fully implemented the requirements. It allowed the student to make supporting and refuting links between evidence and hypotheses, and to supply questions which the program could ask to find out if evidence was present. Once the student's rules had been constructed, the program would ask the student to give names of evidence. The student determined the order of inputs, in keeping with the learner in control philosophy. The program did not perform any inference – the idea was the student should learn to solve a problem in a systematic way by building a computer representation of the steps taken in finding the solution. The student could ask for a report of evidence for and against each hypothesis, and had to decide when to stop, as well as what the solution was. The option of switching to having the program ask the

r1 free switching between gathering evidence and forming, confirming and rejecting hypotheses should be supported – to this end, a permanent report on evidence for and against active hypotheses should be displayed, and hypotheses and evidence should be explicitly activated and deactivated
r2 no order should be imposed on the specific hypotheses and evidence which are considered
r3 rules should specify evidence as supporting or refuting hypotheses
r4 an ordered record of all actions taken by the student should be kept – including choice of hypotheses and evidence to consider, and the order of rule formation
r5 the system should be non-judgmental
r6 solving of problems meaningful to the learner should be supported
r7 the student should supply medical knowledge for solving a given problem by some approximation to building an expert system – the system should not contain any medical knowledge
r8 ease of use should be emphasized: the use of the keyboard should be eliminated as far as possible
r9 inference should be avoided: the student, not the program, should find the solution
r10 only one mode should be used for both problem-solving and knowledge acquisition
r11 signs and symptoms should be automatically considered present when brought into consideration
r12 allowance for a range of attributes in addition to *present* or *absent* should be considered

The experiments with LISP KEBL led to a further revision of the requirements, in the final application of the experimental software engineering methodology.

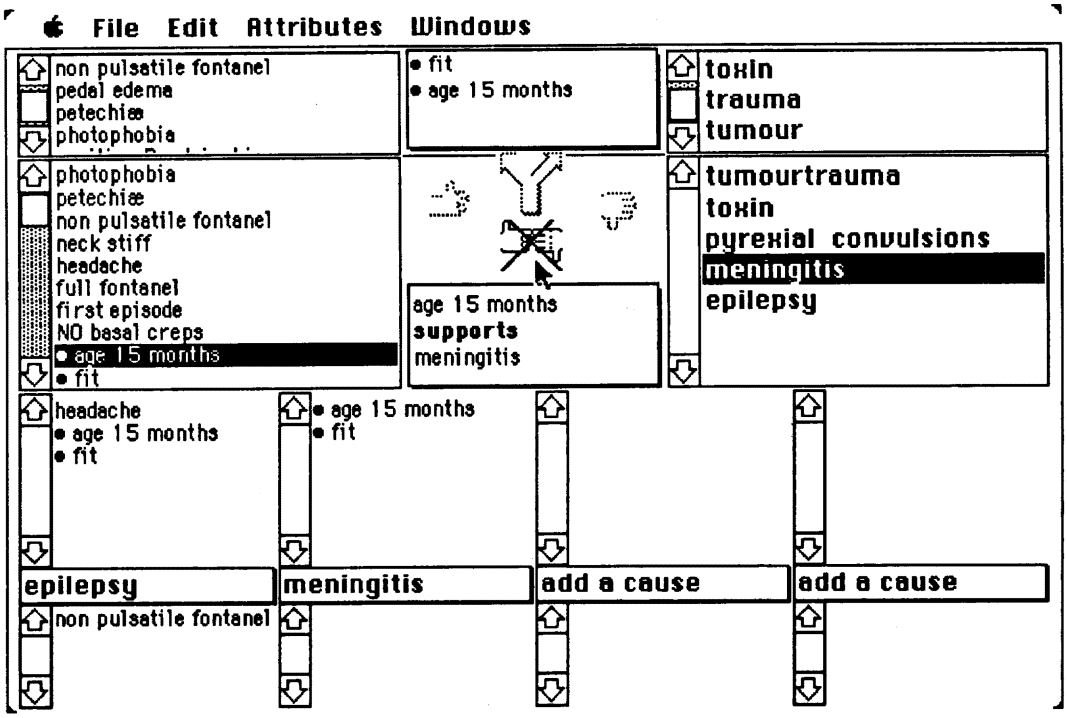
Figure 5: The final requirements

questions the student had supplied for evidence was allowed, to see whether the students would readily adopt the learner in control strategy. The program made use of the underlying LISP system's listener window to record the student's steps, and the student's rules could be saved to disk for reuse or evaluation.

The OPS5 prototype was tested with a group of 47 first-year students, who were given simple problems to solve. Informal observations were the main measure of the usability of the system. In addition, a survey of the students' attitudes was carried out at the end of the exercise, and they were given the opportunity to discuss their impressions with a group of medical educators. The survey results were generally positive; however, the survey was taken just before a long weekend, and only 51% of the students participated, so the results cannot be considered accurate. Of more significance are the observations of the students' use of the program. Typing turned out to be a major problem: the students were unable to make reasonable progress without intervention. In addition, they were only too willing to allow the computer to do the work. The "learner in control" mode of the program was generally avoided, and the mode in which the computer asked the questions was immediately used.

Based on these points – and other observations – the requirements were modified for the next prototype. The biggest innovation was the introduction of a mouse pointing device to make links; the only use of the keyboard according to these new requirements was to be in entering new names. The learner in control aspect was further emphasized, by removing the possibility of having the computer ask questions. Three other changes were matters of detail. A report indicating the evidence for and against each hypothesis under consideration was to be maintained at all times on the screen. In addition, a full record of the steps taken by the user was to be kept on disk, to allow retracing of the user's steps. This is in contrast to the OPS5 prototype, where windows on the screen needed to be explicitly saved, and the exact order of the user's steps was not recorded.

The next prototype was written in LISP. The version of LISP used allowed incremental compilation, which made for flexibility in accommodating the wishes of the three medical educators who tested early versions of the prototype. The essential functionality had ben fixed in the previous experiment, and the issue being investigated was the user interface. The LISP implementation had reasonable



The Pascal Program has only one mode, and the keyboard is not used. Names are loaded into signs and symptoms (top left) and cause (bottom right) dictionaries from disk, and are brought into consideration (into the middle part of the screen) by selecting them with the mouse. Causes are activated (i.e., become active hypotheses) by selecting their names in the "in consideration" section, and then selecting a cause position in a report (bottom). Names may be sent back to the dictionaries, and links are made or broken using the icons in the centre of the screen. Lists may be of indefinite length, and can be scrolled in the standard Macintosh style (using the mouse).

Figure 6: The Pascal prototype's user interface

access to the Macintosh graphics toolbox; other languages such as Pascal were better in this respect, but were considered less suitable for rapid prototyping. The user interface of the program is illustrated in figure 4. For this implementation, a change in terminology occurred: *signs and symptoms* replaced *evidence*, and *causes* replaced *hypotheses*. The reason for this change was to avoid a debate in medical education research as to whether medical problem solving was or was not hypothesis formation [14].

The program was initially tested in field studies at Hillbrow Hospital, Johannesburg, using a total of 14 fourth-year and sixth-year medical students. Subsequently, an experiment was conducted with 16 nursing sisters from the Soweto Community Health Centres. In both cases, findings were based on informal observations. Despite the considerably reduced reliance on the keyboard, typing remained a problem. In neither the Hillbrow nor the Soweto exercises was there time to train the users sufficiently in the use of the keyboard; all typing was done for them. In other respects, the program proved to be clumsy in detail, though the overall strategy was acceptable to the users.

It is interesting to contrast the attitudes of the medical students with those of the sisters, who had less exposure to technology. The medical students were keen on setting up a computerized guru which would find all the answers for them. One went as far as to suggest that the computer could be linked to laboratory equipment to save time in entering information. The sisters on the other hand were more sceptical. One expressed concern that using a computer in a clinical setting would cause the patient to feel neglected. Another argued strongly that the computer strategy could just as easily be carried out on paper. On the whole, though, most participants were positive and indicated interest in seeing further research.

The major weakness of the program was its use of two modes: one for making and breaking links (called *knowledge acquisition*) and one for activating and deactivating causes (called *problem solving*). Although these two modes were considered natural in terms of the original intention to emulate an expert system-building exercise, they were confusing to the users. This finding is considered important for its confirmation of earlier similar findings about the difficulty of using moded software [19]. In addition, modes can be seen as contrary to the learner in control philosophy, in that they restrict the options open to the user.

5. Evaluation of Experimental Software Engineering

The research has only gone as far as a final version of the requirements, with a matching prototype – this time written in Pascal – which, it is hypothesized, could form the basis for continuing with a conventional software engineering exercise. These final requirements are presented in figure 5; key aspects of the Pascal program are illustrated in figure 6. The Pascal program has not been fully implemented. However, the key elements of the user interface are in place, and sufficient detail has been implemented to make a detailed design reasonable straightforward.

Further research is needed into how ESE interfaces to the rest of the software life cycle. In particular, consideration should be given to including ESE tools in a software engineering environment designed to support later stages of the software life cycle. Also, work on examining the cost implications of ESE is needed. Since the ESE aims to increase the accuracy of the requirements specification, it is reasonable to suppose that the later stages – especially maintenance – should be facilitated. However, this claim needs to be justified by further research.

Nonetheless, the ESE case study has illustrated how relatively computer-naïve users can make a meaningful contribution to specifying non-trivial software, aimed at meeting criteria for appropriate technology. In addition, some issues clarified by this research are of wider application. Use of a keyboard is considered to be an inhibiting factor for users who are unfamiliar with technology. A modeless user interface, in which the user is free to choose the order of events, is found to be more intuitive than a heavily moded one. Furthermore, a learner in control strategy is found to be natural, and the learner should not have the option of allowing the computer to control the order of events. An important finding is the usefulness of AI tools on relatively cheap computers, especially as tools for ESE.

The extent to which these lessons apply generally needs further research. It would be particularly interesting to investigate whether the process of software production could become appropriate technology, so that third world countries could develop their own software industries. The case study presented here illustrates what can be achieved using relatively low-cost equipment; further research should be possible without losing sight of the intention of making the results accessible to poorer parts of the world.

Acknowledgements

I would like to thank Conrad Mueller and Prof. A.M. Starfield, who supervised the research for a Masters degree on which this paper is based, for their helpfulness and encouragement. In addition, Dr Andrew Truscott was very helpful in setting up the experiment in Soweto. Ian McNairn, Prof Graham Mitchell and Prof Pat MacPhail assisted in designing and testing the LISP prototype. The students who were prepared to be subjects of the experiments, as well as those who assisted in setting up and running the experiments, made the whole thing possible. Scott Hazelhurst assisted in proofreading this paper, and made helpful comments.

References

[1] B Auvert, P Aegerter, V Gilbos, E Benillouche, P Boutin, G Desvé, M-F Landre and D Bos, [1986], Tropicaid: Un système expert sur ordinateur portatif pour l'aide à la décision médicale dans les pays en développement, *6th International Workshop on Expert Systems and Their Applications*, 28–30, Avignon, France.

[2] B W Boehm, [1976], Software Engineering, *IEEE Transactions on Computers*, **25** (12), 1226–1241.

[3] B W Boehm, T E Gray and T Seewaldt, [1984], Prototyping Versus Specifying: A Multiproject Experiment, *IEEE Transactions on Software Engineering*, **10** (3), 290–303.

[4] B G Buchanan, [1982], New Research on Expert Systems, *Machine Intelligence* **10** (ed. J E Hayes, D Michie and Y-H Pao), Ellis Horwood, Chichester, 269–299.

[5] P Byass, [1987], Computers in Africa: Appropriate Technology? *Computer Bulletin*, **3** (2), 1987, 17.

[6] J M Carroll and S A Mazur, [1986], LisaLearning, *Computer*, **19** (11), 35–49.

[7] J Doyle, [1985], Expert Systems and the "Myth" of Symbolic Reasoning, *IEEE Transactions on Software Engineering*, **11** (11), 1361–1374.

[8] J D Gould, S J Boies, S Levy, J T Richards and J Schoonard, [1987], The 1984 Olympic Message System: A Test of Behavioral Design Principles of System Design, *CACM*, **30** (9), 758–769.

[9] N Jéquier and G Blanc, [1979], *Appropriate Technology Directory*, OECD, Paris.

[10] G Kiliany, [1987], Response to "LisaLearning" Article (letter to editor), *Computer*, **20** (3) March, 4.

[11] *MACazine*, [1987], Business Report, *MACazine*, **4** (12), 41–63.

[12] P Machanick, [1986], Low-Cost Artificial Intelligence Tools, *Quæstiones Informaticæ*, **4** (3), 27–32.

[13] P Machanick, [1988], *Design of Medical Education Software as Appropriate Technology Using Artifical Intelligence and Software Engineering* (masters dissertation), Computer Science Department Technical Report 1988–01, University of the Witwatersrand, Johannesburg.

[14] C H McGuire, [1985], Medical Problem-Solving: A Critique of the Literature, *Journal of Medical Education*, **60** (8), 587–595.
[15] D L Metzger, [1987], "LisaLearning" Called Apple-Bashing Session (letter to editor), *Computer*, **20** (3), 4.
[16] M Munashinghe, M Dow and J Fritz, [1985], *Microcomputers for Development*, CINTEC-NAS, Sri Lanka.
[17] A Newell and H A Simon, [1976], Computer Science as Empirical Enquiry: Symbols and search, *CACM* **19** (3), 113–126.
[18] S Papert, [1980], *Mindstorms*, Harvester Press, Brighton.
[19] T S Perry and P Wallich, [1985], Inside PARC: The 'Information Architects', *IEEE Spectrum*, **22** (10), 62–75.
[20] S P Reiss, [1985], PECAN: Program Development Systems that Support Multiple Views, *IEEE Transactions on Software Engineering*, **11** (3), 276–285.
[21] E Sandewall, [1978], Programming in an Interactive Environment: the 'LISP' Experience, *Computing Surveys*, **10** (1), 35–71.
[22] W Teitelman, [1984], A Tour through Cedar, *IEEE Software*, **1** (2), 44–73.
[23] W Teitelman and L Masinter, [1981], The INTERLISP Programming Environment, *Computer*, **14** (4), 25–33.
[24] P Wegner, [1984], Capital-Intensive Software Development, *IEEE Software*, **1** (3), 7–45.

This paper was received in camera-ready form.

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. These may include research, review and exploratory articles of interest to the journal's readers. The preferred language of the journal will be English, although papers in Afrikaans or other congress languages of IFIP will not be precluded. Typed manuscripts for review should be submitted in triplicate to:

Professor D G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
Pretoria

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 Computing Review Categories.
- Tables should be typed on separate sheets of A4 paper, and should be numbered and titled.
- Figures should also be supplied on separate sheets of A4 paper, and should be identified on the back in pencil with the author's name and the figure number. Original line drawings, and not photocopies, should be submitted.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin. Distinguish clearly between:
  - upper and lower case letters
  - the letter O and figure zero
  - the letter I and the number one
  - the letter K and kappa.

  References should be listed after the text in alphabetical order of the (first) author's surname, cited in the text in square brackets. References should 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 (3), 366-371.

Manuscripts accepted for publication should comply with the above guidelines, but may be in one of the following three formats:

- typewritten and suitable for scanning
- provided as an ASCII file on diskette
- camera-ready.

Authors wishing to provide camera-ready copy may obtain a page specification from the production editor.

Charges

A page charge, scaled to reflect production costs, will be levied on papers accepted for publication. The costs per final page are as follows:

Typed format, not camera-ready R60
Disk in ASCII format R40
Camera-ready format R20

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

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.

Copyright

Copyright in published papers will be vested in the publisher.

Letters and Communications

Letters to the editor will be welcomed and will provide a forum for discussion on topical issues. They should be signed, and should be limited to about 500 words.

Communications reflecting minor research contributions will be considered for publication in a separate section of the journal. Such communications will, however, not be regarded as a fully-fledged publication for FRD 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.

Advertising

Placement of advertisements at R1000 per full page per issue and R500 per half page per issue will be considered. Enquiries should be directed to the production editor.