
This item was submitted to [Loughborough's Research Repository](#) by the author.
Items in Figshare are protected by copyright, with all rights reserved, unless otherwise indicated.

Authoring educational software: theory and practice

PLEASE CITE THE PUBLISHED VERSION

PUBLISHER

Loughborough University of Technology

LICENCE

CC BY-NC 4.0

REPOSITORY RECORD

Newton, John C.. 2021. "Authoring Educational Software: Theory and Practice". Loughborough University.
<https://doi.org/10.26174/thesis.lboro.14413697.v1>.

Authoring Educational Software: Theory and Practice.

by John Christopher Newton.

A Master's by Course Dissertation submitted
in partial fulfilment of the requirements
for the award of the degree of M.Sc. in
Computer Education of Loughborough
University of Technology,
January 1990.

Supervisor: Dr. D. R. Green, M.Sc. M.Ed. Ph.D.

(C) by JOHN CHRISTOPHER NEWTON 1990.

Abstract.

The production of educational software requires the skills and knowledge from a number of fields to be drawn together in order to meet the needs of an increasingly discerning audience of users. Thus, a collection of perspectives for those involved in the authoring of this software is presented.

Chapter 1 provides a historical exposition of the development of computer assisted learning (CAL) from its pre-computer beginnings to the present day.

Next, Chapter 2 considers the CAL production system. Three specific issues are dealt with here: selecting and using authoring packages, an investigation of two commonly used packages, and a survey of commercially available packages within the U.K.

This is followed by three essential perspectives for those involved in CAL software production: Chapter 3 presents an examination of the relevant education issues; Chapter 4 is concerned with designing for effective human-computer interaction; and Chapter 5 looks specifically at the application of software engineering techniques to the CAL authoring process.

To complete this dissertation, a look to the future is presented, which identifies the developments that are likely to affect those involved in the development of educational software.

Acknowledgements.

I would like to acknowledge the contribution made by a number of people to the content of this dissertation. My thanks are extended to the following:

- firstly, and most importantly, my supervisor David Green, who has supported my efforts with unfailing advice and encouragement.
- Bill Lynch of Systems Interactive, who taught me everything I know about the TenCORE language.
- Karen Smith of Format P.C. for freely giving technical information about the Top Class system.
- the Staff at Loughborough University who made the Computer Education course so interesting and stimulating.
- lastly, but by no means least, my wife Sandra who has put up with me for the last 2½ years.

Declaration.

I declare that this dissertation is entirely my own work.

Contents.

Abstract	ii
Acknowledgements	iii
Chapter 1.	
Introduction: A Historical Perspective	1
<i>1920-1950: The Teaching Machines</i>	3
<i>The Arrival of Computers</i>	6
<i>CAL in the U.K.</i>	10
<i>Summary</i>	12
Chapter 2.	
Authoring Educational Software:	
The Production System.	14
<i>Modelling Authoring Packages</i>	15
<i>Two Authoring Packages</i>	25
<i>An Authoring Package Survey</i>	39
Chapter 3.	
Educational Issues for CAL Authors	44
<i>The Theories of Learning</i>	46
<i>Modelling the Learner</i>	55
<i>Learning and Computers</i>	60
Chapter 4.	
Human Factors for CAL Authors	69
<i>The Nature of HCI</i>	70
<i>Authoring System: The Interface</i>	
<i>Between Author and Computer</i>	73
<i>Courseware: Interface Between</i>	
<i>Learner and Computer</i>	86

Chapter 5.

Courseware Production:

A Software Engineering Approach	97
<i>The Software Life Cycle</i>	97
<i>The Production Team</i>	102
<i>Documentation</i>	103
<i>Coding Conventions</i>	106
<i>Software Testing</i>	107

Chapter 6.

Conclusion: A Look to the Future	114
<i>Hardware Developments</i>	114
<i>Software Developments</i>	122
<i>Developments Within Education</i>	126
<i>An Agenda for the Future</i>	129

Bibliography	132
---------------------	------------

Appendix A.

Authoring Language Suppliers	142
-------------------------------------	------------

Appendix B.

The TenCORE Instruction Set	144
------------------------------------	------------

Appendix C.

The Top Class Instruction Set	146
--------------------------------------	------------

CHAPTER 1.

Introduction: A Historical Perspective.

Overview.

Computer based learning is a relatively new field. Because of the links with (or dependence on) new technology, it is a field that is constantly developing. In order to analyse the issues that affect present and future educational software design, it is appropriate to begin by considering the historical development of computer based education.

This chapter describes the significant past developments of CBE and examines the evolutionary process that has led to the present state of educational software in the U.K. The view presented is selective, and highlights the events pertinent to the major theme of this dissertation: the theory and practice of courseware authoring.

Terminology.

Before going any further, it is important to consider the range of acronyms that plague the field of educational computing. The following definitions are presented in order to offer an explanation of those in common usage. Many of these terms seem to be inter-changeable and, to some extent, their use is influenced by what is currently fashionable.

The use of computers to support the learning process, perhaps by employing a tutorial, simulation or game program is logically referred to as Computer Assisted

Learning, or simply CAL. More widely seen in publications from the U.S. is the term Computer Assisted Instruction (CAI) which, to readers in the U.K., suggests overtones of drill-and-practice type software. Nevertheless, when comparing texts of U.S. and British origin the terms CAL and CAI appear to be synonymous.

Other acronyms, which suggest a more exclusive use of the computer in the teaching environment, are Computer Based Learning (CBL), Computer Based Education (CBE) and Computer Based Training (CBT). The latter is often used in an industrial training context. Occasionally, the following acronyms are encountered: CAT -Computer Assisted Training; CEL -Computer Enhanced Learning; CSL - Computer Supported Learning.

Finally, it is useful to distinguish between the use of the computer to support learning and the computer used to manage the learning process. The terms Computer Managed Learning (CML) and Computer Managed Instruction (CMI) reflect this difference.

The Development of Computer Based Education.

From out of the maelstrom of past activity in the field of CAL, three distinct phases of activity can be identified that have played an important part in shaping the present state of educational software. These are:

- The development of teaching machines and associated theories of instruction during the period 1920 to 1950.
- The use of computers since 1960 and the influence of the PLATO and TICCIT projects.
- The major CAL initiatives that have shaped the present state of affairs in the United Kingdom: NDPCAL and the MEP.

1920-1950: The Teaching Machines.

Arguably, the prelude to the use of computers as an aid to teaching was in the development and use of the so-called "teaching machines." These machines were devised to automate the process of delivering material of an instructional nature and to test the subsequent learning outcome.

Richmond (1965, p37) indicates the logical starting point for a historical exposition of computer assisted learning in stating that "...Sidney Pressey of Ohio State University is the man who is usually credited with inventing the first teaching machines..."

The Work of Pressey.

Pressey's work, which began in 1926, was concerned with automating the process of administering objective-type tests by using a mechanical apparatus to produce a test giving machine. Of simple construction, such a machine would display typewritten questions through an aperture, to which the student would respond by selecting an answer and pressing the corresponding key. Correct answers would result in a counting device being incremented at the rear of the machine. The major drawback with this original design was that the student was not provided with any feedback to indicate which questions were answered correctly.

Recognising this limitation, Pressey modified his design so that the machine would wait until a correct response was made before presenting another question to the student. This modification produced a machine that gave the student some feedback, albeit minimal, which at least let the student know whether the answer given was right or wrong. A third (and apparently final) change to the design was made, which produced an early example of machine generated positive feedback. This was achieved by

arranging for the machine to reveal the correct answer to the student whenever an error was made.

According to Richmond (1965 p.47) Pressey's work came to an end by 1932, as a result of Pressey becoming discouraged by the lack of interest shown by potential manufacturers of his machine. As a result, most of his work was subsequently confined to the laboratory.

The Work of Skinner.

Perhaps the most significant landmark in the history of CAL is the work of B.F. Skinner, who documents the use of a teaching machine in an article published in "Science" (Skinner, 1958). However it is his work in behavioural psychology and learning that is his legacy to modern CAL.

Skinner's approach to understanding the mechanism of the mind was mainly based on his earlier investigation into animal learning. He came to believe that effective learning was based on a process of mastering a subject in small steps in a linear fashion with each step building on its predecessor. In fact, the underlying theme of Skinner's work was a complete theory of instruction based upon operant conditioning and exemplifies the basic tenets of the Behaviourist school of psychology.

Skinner's early work on programmed instruction used text books as a delivery medium. These did not use the usual format of paragraphs but consisted of page after page of short sentences, each conveying one piece of information, and each building on the previously presented facts. Hundreds of these pieces of information made up a course of instruction.

Given the theories that the behaviourists put forward regarding the learning process, then the logical step for the teaching process was for it to become similarly mechanistic in nature.

Skinner (1954) recognised this in stating that "Mechanical and electrical devices must be used..." when he described the applicability of the teaching machine to his methods of programmed instruction. Retrospectively analysing the use of computers, Kearsley (1985) states that Skinner's work "... provided the conceptual framework for the initial efforts towards computer assisted instruction".

Not surprisingly, as Criswell (1989) points out, Skinner's teaching machines were not widely received with open arms in observing that "Many educators resisted the machines and their programs because they felt the machines might displace teachers or impart instruction in an undesirable, mechanistic fashion." Indeed, with programmed instruction providing the framework for much of today's CAL software is is not surprising that the objections levelled at Skinner's teaching machines are equally valid today.

The Work of Crowder.

During the late 1950's another figure, Norman Crowder, emerged to be a significant influence in the development of Computer Based Learning. Crowder based his approach and subsequent training solution on the real life problems he encountered as an instructor in the U.S. Air Force. Crowder describes his methodology, the so-called "intrinsic programme," as:

"An individually used, instructor-less method of teaching which represents an automation of the classical process of individual tutoring. The student is given the material to be learned in small logical units and is tested on each unit immediately. The test result is used automatically to control the material that the student sees next [...] The test questions are multiple choice questions and there is a

separate set of correctional material for each wrong answer that is included in the multiple choice alternatives." (Crowder 1959).

It is this branching capability of Crowder's "intrinsic programme" that provides an advancement of the philosophy of automated instruction. Prior to this, no provision appears to have been made for the remedial treatment of wrong responses in programmed instruction.

In allowing learners to find out where they went wrong, Crowder realised that constructive treatment of errors was an important part of the learning process, and that computers were the ideal machines to cope with the delivery of this type of individualised instruction.

The Arrival of Computers: PLATO and TICCIT.

The development of the digital computer was not directly associated with educational activities but to provide a method of dealing with complex mathematical calculations. However, the development of an easy to learn programming language, i.e. BASIC, at a major U.S. University (Dartmouth) provided many educators with the means to access the new technology and hence to be able to write computer assisted instruction programs.

Along with the rapid growth of computer technology, the BASIC language spread into many schools and colleges, providing practising teachers with the means to program their own CAL material. Although the educational and technical quality of much of this home-grown software was (and still is) questionable, the BASIC language has become so widely available that it is a favourite choice for the production of educational software by those with little or no computer programming experience.

In fact, the BASIC language has undergone many modifications and changes to keep pace with the microprocessor systems that it is now used on. Today, virtually every home/personal computer is supplied complete with BASIC. With such proliferation, it is no surprise that it continues to be a widely used language for CAL courseware production.

The early notable efforts in the U.S. were aimed specifically at the design and production of CAI material, and took place in the 1960's as educational research projects: namely the PLATO and TICCIT systems. Significantly, each had its own programming language designed specifically for the production of educational software. The main purpose of each project was to design and implement cost effective computer based instructional systems.

PLATO

The PLATO project was developed at the University of Illinois, and was supported by the National Science Foundation and the Control Data Corporation. The main goal of the project was "... to develop an automatic teaching system sufficiently flexible to permit experimental evaluation of a large variety of ideas on automatic instruction." (Bitzer et al. 1962.)

The PLATO project started out as a single student system. However, as the logic of the system was developed, the project staff worked towards the goal of achieving multiple student instruction. Tenczar (1981) reports that the PLATO system became operative in the early 1970's, by which time "...over 100 man years of system software effort..." had gone into the development of operating system software and an authoring language (called "TUTOR") to control the PLATO hardware.

The resulting PLATO IV system is described by Alpert (1975) as a computer based network system based around a central computer that could serve up to 1,000 student consoles. The consoles, or terminals, comprised a keyboard and graphics screen. Additionally, the system could accommodate touch screen based student interaction, a micro-fiche image selector and an audio unit. Of particular interest, is the TUTOR authoring language, which comprised of over 400 commands to allow programming of CAI related tasks such as graphics generation, response analysis and student data keeping.

Tenczar goes on to explain that as general purpose microprocessor based computers became available during the 1970's the researchers on the PLATO project began the task of investigating the possibilities of implementing mainframe type CAI on micros.

The PLATO project was evaluated in 1977 by the Educational Testing Service. PLATO material was used in community colleges in Chicago and at Urbana, Illinois. Although the evaluators found that a significant improvement was found when comparing PLATO to traditional teaching in mathematics, no improvements were noted for other subjects.

So, the overall effect of the PLATO project was essentially to advance the hardware and production aspects of CAI material, but offered no positive outcome with respect to addressing the problem of using computers to help bring about effective learning. It is important to note that the development of the TUTOR programming language demonstrates the recognition of the need for a purpose made authoring language for the production of educational software.

PLATO is now marketed by the Control Data Corporation, and has been used internationally. The

Coventry PLATO project (Bell 1985) and a project undertaken at the University of Witwatersrand, Johannesburg (Freer, 1986) are two documented examples of the PLATO system in use.

The TICCIT Project

The TICCIT (Time-Shared Interactive Computer Controlled Information Television) project was carried out as a joint venture by the MITRE Corporation and Brigham Young University. Like the PLATO project, it was also heavily funded by the National Science Foundation. The project aimed to develop college level courses in Mathematics and English, with the MITRE Corporation designing and developing the hardware and system software and Brigham Young developing courseware.

Describing the use of TICCIT at Phoenix College, Arizona, Morrison (1975) explains that mini-computers were linked to 128 computer terminals, each consisting of a high resolution colour television receiver and a keyboard. With regard to the development of courseware, Morrison states that "TICCIT accomplished what no other CAL system has done to date." He clarifies this bold claim by saying that "...course content and computer programming have been completely separated. Authors need not learn computer programming."

The project leader, Victor Bunderson, explains how TICCIT aimed to produce learner controlled courseware (Bunderson 1974). He characterises this learner controlled courseware as embracing a modular approach to courseware structure and being related to a taxonomy of instructional variables.

Bunderson also advocates a team based approach to courseware production but does not elaborate on the authoring mechanism. In a later paper (Bunderson 1981) he states that the distributors of TICCIT, (Hazeltine Inc.)

have developed a TICCIT Authoring Language (TAL) in response to a call for greater flexibility in courseware authoring.

The TICCIT project was evaluated by the Educational Testing service. Chambers and Sprecher (1983) report that the evaluation provided some evidence that CAI could be an effective instructional tool, but "...the evidence was far from clear-cut."

Computer Assisted Learning in the U.K.

The National Development Programme in Computer Assisted Learning.

The beginning of co-ordinated computer assisted learning (CAL) activity in the U.K. is usually associated with the National Development Programme in Computer Assisted Learning (NDPCAL). This was funded by the Government from 1973 to 1977, at a cost of £2.5 million. According to MacDonald (1977), the NDPCAL represented

"...a departure from a monolithic tradition of computer based curriculum development, largely American, which has given rise to a stereotyped view of what computer assisted learning means."

The stereotype that he refers to here is, of course, that of Skinnerian programmed instruction.

The Project Director, Richard Hooper (1977) saw the major aim of the project as primarily, "...to develop and secure the assimilation of computer assisted and computer managed learning on a regular institutional basis at a reasonable cost." This innovative aim was tackled by stimulating CAL activity through the development of new

courseware. Essentially, this appears to have been based on work already under way at Leeds University (for projects in Chemistry and Statistics), Queen Mary College (for the Engineering Sciences project) and at the University of Surrey to develop materials for undergraduate science education. This later became known as CUSC - Computers in the Undergraduate Science Curriculum.

Hooper boasts that the scope of the project in the year 1975/76 included "...17 universities, 10 polytechnics, 20 colleges (including military colleges) and 31 schools." He states "...some 10,000 students/pupils were experiencing CAL and CML in NDPCAL projects," and reports that the project resulted in the production of some 200 CAL packages. It seems, however, that little formal or objective evaluation was implemented.

Although NDPCAL may have given more people an exposure to CAL than would otherwise have been the case, I fear that the result may well have been to generate caution rather than enthusiasm concerning the value of CAL in the classroom.

The Micro-electronics Education Programme.

Starting in 1981, the Government funded a five year program as a successor to the NDPCAL project. This was called the Micro-electronics Education Program (MEP). This scheme, aimed at education in schools, had three main aims:

- to provide in-service training in micro-electronics.
- to stimulate curriculum developments using computer technology.
- to disseminate programmed materials and information.

At about the same time, the Department of Industry funded a scheme to put at least one microcomputer into

every school. However, the computers that were made available to the local authorities were limited to approved British micros only. These were sold at half the full cost, leaving the local authority to find the other half. Although financially advantageous, this scheme imposed a major limitation in terms of the paucity of suitable educational software: indeed, at the time, no software existed for the approved computers !

According to Hartley (1987) this software vacuum would be filled by "...commissioning large and experienced units..." to generate the required materials to "...commercial standards." The production of software appears to have been a vigorous and frenzied activity, with some 300 programs developed by the end of 1982 (Educational Computing 1982,3(7), p.8).

Taking note of the views of two recognised commentators on the educational computing scene, it is clear that commercial standards were not reached:

"A mountain of poor-quality educational software seems to be accumulating with only a molehill of quality". (Maddison 1983);

"...the courseware available so far for use in the new machines is inadequate in quality, quantity and variety." (Hawkridge 1983);

Summary.

The advances made in technology, particularly in the development and refinement of micro-electronics has had a profound effect on the development of low-cost yet powerful computers. As a result, educationalists now have suitable hardware resources to implement effective CAL programs. Unfortunately, educational software of the

required quality or quantity is not yet available.

If a remedy for this situation is to be found, then an analysis of past events is important. In doing so, the following issues arise as being worthy of further consideration:

- Courseware production methods. Clearly, the majority of software is written with unsuitable programming languages, as demonstrated by the proliferation of courseware written in BASIC.

As the PLATO and TICCIT projects have demonstrated there is a case for using a programming language that has been designed expressly for the purpose of producing educational software - the so-called "authoring language." This issue is discussed fully in Chapter 2.

- Software authoring skills. Software quality is directly affected by the skills of the author. To produce good quality educational software, the author must be equipped with knowledge of the fields of educational psychology and effective human-computer interaction. If the mistakes of the MEP-produced software are to be avoided then suitable training in these fields must be provided. Chapters 3 and 4 examine the educational and human-computer interaction issues more closely.

- Managing the authoring process. The widely accepted principles of software engineering may well provide courseware authors with the basis for a courseware design and development methodology. Thus, the production process, is considered further in Chapter 5.

Chapter 2.

Authoring Educational Software: The Production System.

Introduction.

Historically, the production of educational software has been the province of the computer programmer. However, this situation is slowly being changed as educators become able to produce their own courseware using a new generation of sophisticated programming tools: the so-called authoring packages.

Thus, the new breed of teacher-cum-programmer is faced with a daunting task: to select an authoring package that is suitable for his/her own use. Often, the selection process requires the individual to sift through the claims made by the manufacturers of these complex software tools, necessitating many feature comparisons to be made before a decision can be reached regarding which package to purchase.

In an attempt to assist the prospective authoring package user, this chapter describes a framework which may be used to evaluate and compare different authoring packages. The chapter is divided into three sections:

Section One presents a model which may be employed to evaluate and compare any software package used for the production of educational courseware.

Section Two uses the model presented in Section One as the basis for describing in some detail two significantly different Authoring Systems, namely TopClass and TenCORE.

Section Three presents a survey of the authoring languages and systems that are commercially available in the U.K.

Section 1: Modelling Authoring Packages.

The term 'Authoring Package' is a broad description that embraces two different types of software package that may be used to produce CAL material. The two types of package are a) Authoring Languages, and b) Authoring Systems.

Authoring Languages are similar in construction to traditional programming languages. They consist, at basic level, of an instruction set which is used by the author to write the CAL program. Unlike a 'traditional' programming language, the instruction set of an authoring language is in a form more suited to the task of producing CAL type programs.

Authoring Systems on the other hand, tend to be more restrictive, typically offering an inflexible menu/prompt driven program creation environment. Such systems have acquired the inevitable reputation of being responsible for reducing the author's overall control of the design and outcome of the resultant software, both in a technical and pedagogic sense.

The Need to Evaluate.

With the wider availability and use of authoring packages, the CAL author is faced with an increasingly difficult task: to evaluate and select an authoring package that is suitable for his/her own use.

The purpose of evaluating authoring packages then, is simple: to provide information, either for oneself or others about the strengths, weaknesses and suitability of a particular package, either specifically, or for comparative purposes. Unfortunately, what actually constitutes a 'good' or 'bad' authoring package will depend on the requirements of the individual, who will have his own set of criteria. The process of evaluation (and hence comparison) is, to quote Wellington (1985) "...ultimately a personal and subjective activity."

A Framework for Making Comparisons.

How should the task of evaluating and comparing authoring packages be performed ? One of the most commonly adopted methods is to produce a table, listing all the feature comparisons that may be made. While such an approach presents a lot of information concisely, it often is not in a form that is easily understood, particularly those who do not have a technical background.

In order to allow meaningful judgements to be made, it is desirable to establish a framework that may be used to model any programming system that could be used for producing educational software. The use of a model will introduce uniformity to the evaluation process, thus making the resulting information more meaningful to a wider audience of prospective users.

Fairweather and O'Neal (1984), have put forward a suitable model for comparing authoring packages. It considers three basic criteria, relevant to authoring package performance. They are :-

- Power.
- Productivity.
- Ease of Use.

This model, shown in Figure 2.1, represents these parameters graphically. It forms the basis for evaluating any software system used for producing educational material. For the purposes of further discussion it will be referred to as the P.P.E. model (Power - Productivity - Ease of use). Each parameter is considered in detail below.

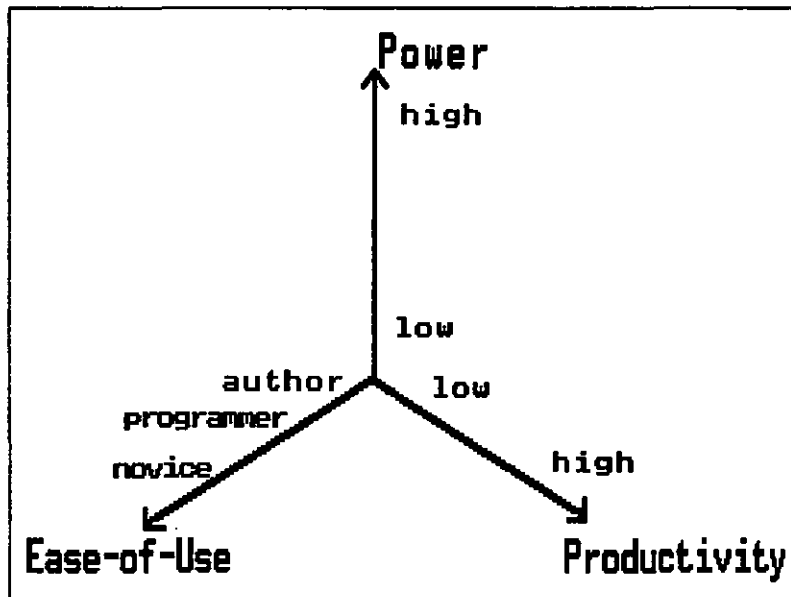


Figure 2.1

Power. Essentially, this axis of the model provides an indication of how easily the capabilities of the computer hardware may be accessed, and utilised, by the author/programmer. Thus, the power rating provides an indication of functionality. The power rating for a given package will be directly related to the range of instructions that the programming language provides.

Many authoring packages currently available provide the programmer with instructions specifically to handle the following types of programming task:

- generate text and graphics displays.
- generate questions.
- perform answer judging.

- control the flow of program execution.
- perform mathematical operations.
- provide data storage facilities.
- control peripheral devices.

The authoring package under investigation should, therefore, be tested to determine whether adequate mechanisms are provided to allow the programming of these fundamental CAL specific tasks.

Productivity. This axis indicates the potential productivity of the authoring/programming package. It provides a measure of the quantity and sophistication of CAL material that may be produced per man-hour of effort. Potential productivity is directly influenced by the availability of certain CAL production 'tools' that enhance the authoring process. Typical productivity tools include:

- drawing utilities.
- graphics editors.
- font editors.
- 'test-and-revise' utilities.
- on-line help and debugging facilities.
- support documentation.

Ease of use. This axis provides an indication of how easily the authoring/programming system is learned and used. Ease of Use is essentially a human-computer interface issue. Considering the issue of software usability, Goodwin (1987) highlights that different "sets" of users (or in this context, authors) will have different needs:

"First time, casual, and expert users may all have different requirements, and their requirements may change as they move from one level of expertise to another."

With this point in mind, the Ease of Use axis of the model will be considered in terms of three distinct regions, in order to indicate if the system under evaluation is:

- only suitable for those with previous authoring experience (a low Ease of Use rating).
- only suitable for those with programming experience.
- suitable for use by novice programmers (a high Ease of Use rating).

On the issue of usability, we would be well advised to remember the words of Pogue (1980): " Any authoring system must be easy to learn and simple to use regardless of the computer expertise of the author." This statement embraces the *raison d'etre* of authoring packages: to put CAL software production into the hands of teachers rather than programmers. Hence, the assessment of Ease of Use must be extended to consider the availability of features such as automatic error checking, on-line debugging and good support documentation.

Using the Model.

For illustrative purposes, consider the following examples, which show how the model may be used to represent graphically three different types of CAL authoring method.

Example 1: A "General Purpose" Programming Language

It is not uncommon for many CAL authors to produce courseware using a 'general purpose' programming language such as BASIC or Pascal, for example. Figure 2.2 shows how this type of authoring method is represented using the P.P.E. model.

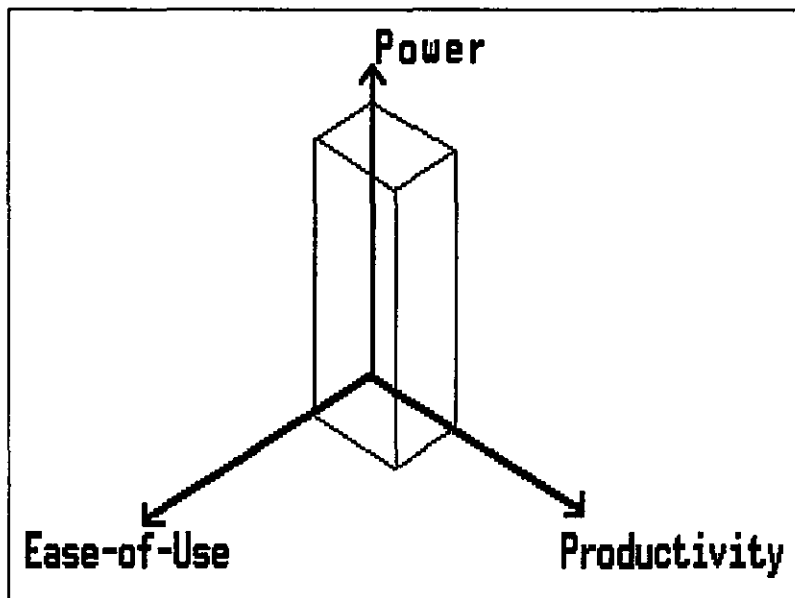


Figure 2.2

Broadly speaking, a general purpose programming language offers a wide range of instructions, thus allowing the author/programmer to exercise a high degree of control over how the computer hardware is utilised. However, because many CAL type programming activities cannot be directly written using general purpose instructions, (question and answer judging or importing graphic images, for example) this method does not offer a very productive means of authoring.

Furthermore, a general purpose language is not easy for novice authors to use because it typically requires the author to: a) master the use of a complicated instruction set, and b) learn to use an unfriendly command-line style interface. To write educational material, an experienced programmer's skills are required to produce some of the 'ingenious' programming routines needed !

Example 2: A Typical Authoring Language.

Unlike general purpose programming languages, many of the authoring languages that are currently available offer a comprehensive range of instructions designed specifically to handle CAL programming tasks. A 'typical' authoring language is modelled in Figure 2.3.

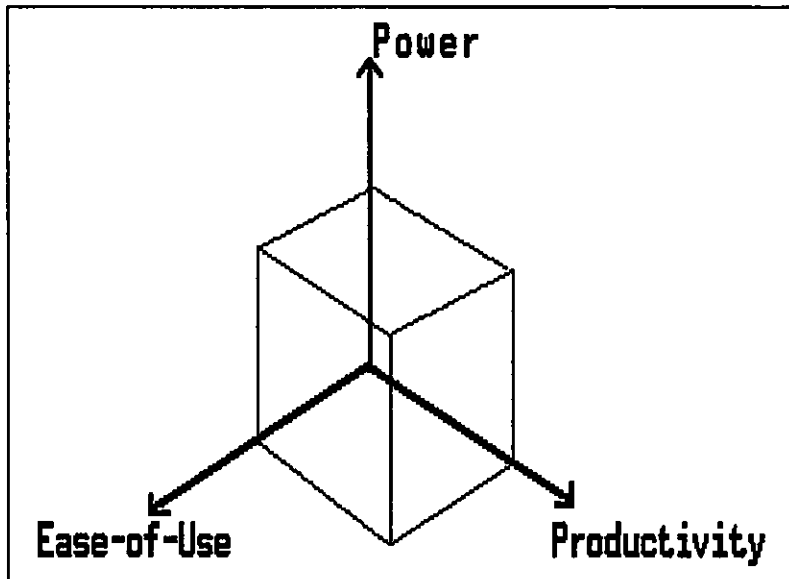


Figure 2.3

Because authoring languages are designed for the production of CAL software, they tend to provide a CAL biased instruction set. Consequently, the overall range of instructions provided is often reduced, compared to a general purpose language. This results in less hardware control being available to the author. Productivity however, is greatly enhanced by the availability of CAL specific instructions.

Authoring languages tend to offer the same type of programming environment as the general purpose language, requiring the use of an editor to produce source code, and a compiler or interpreter to generate the executable program. Thus, it is often necessary for the author to have some previous programming experience in order to make the best use of authoring languages.

Example 3: A Typical Authoring System.

A growing number of authoring systems are available that aim to provide subject experts, rather than programming experts, with the tools to produce CAL material. A 'typical' authoring system is modelled in Figure 2.4.

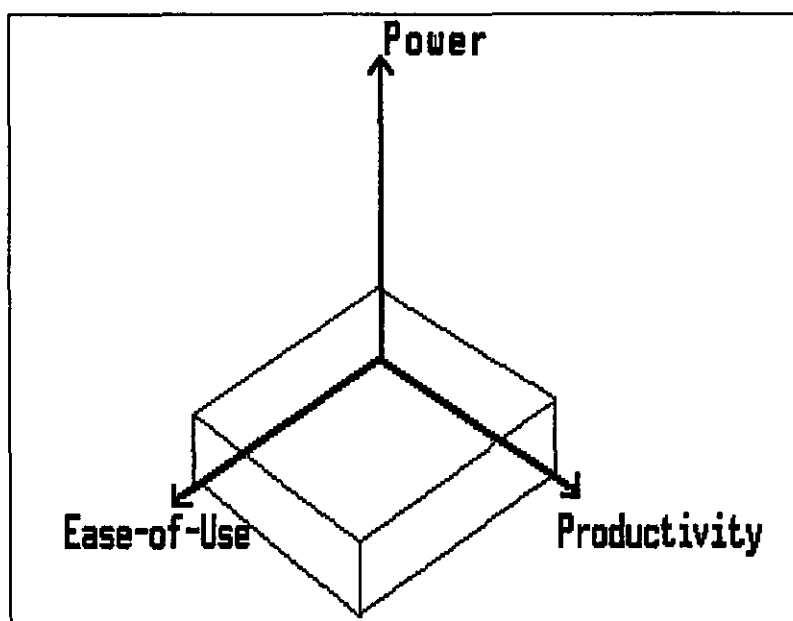


Figure 2.4

Authoring systems are, by their nature, very restrictive. Typically offering a menu and prompt driven interface, they do allow novice authors to use the system, but their 'easy-to-use' philosophy is also responsible for severely restricting the amount of control in design that may be exercised by the author.

Within the constraints imposed by the authoring system, the novice author can be very productive, but at what cost ? The resulting courseware usually consists of monotonous, uniform presentations of text based material. Unless used with care, authoring systems simply allow poor quality software to be produced quickly.

Observations on the P.P.E. Model.

As described, the P.P.E. model only deals with the technical aspects of the authoring package being considered and fails to recognise the importance of factors such as cost, availability and the type of computer needed, for example. Accepting these limitations, and using the model as the framework for a technical evaluation of CAL authoring packages, it has two main failings;

1. It provides no indication of the quality of the resulting software.
2. It does not indicate how free the author is to implement a chosen teaching strategy in his courseware design.

Clearly, the issues of quality and teaching strategy are worthy of further examination, and are considered below.

Quality.

Bork (1984), in considering the production of computer based learning material, states that " The final test of any production system is the quality of the product." Clearly, the issue of quality is an important one, but Bork appears to imply that the quality of the authoring package may be judged largely on the courseware that it has produced. To do so is quite wrong. It is the skill and ability of author that determines quality.

Indeed, to consider the issue of quality solely in terms of the production system is dangerous: it implies that a given authoring package will produce high quality CAL material regardless of the author's ability.

This must be avoided, to prevent inexperienced authors believing that by using a certain authoring package, good quality CAL material will automatically result.

Teaching Strategy.

As it stands, the model does not provide any information regarding the pedagogical aspects of the authoring package being modelled. The only indication of scope for individual design is implied by the power axis rating, but this only indicates the degree of freedom from a technical view point.

Clearly, it is important that the production system should be able to accommodate the design of software using a variety of different teaching strategies, as preferred by an individual author. Thus, an authoring package should be educationally neutral, and not favour the production of one particular strategy, e.g. drill and practice, games or simulations.

So, in order for the P.P.E. model is to be used for the evaluation of authoring systems, two options arise: the model must either be modified in some way, to indicate pedagogical freedom, or pedagogical freedom must be considered as a separate issue. For the purposes of Section Two of this chapter, the second option will be adopted.

Section Two: Two Authoring Packages.

In this section, the previously described model and evaluation criteria will be applied to two markedly different authoring packages: TopClass, an authoring system used widely in education; and TenCORE, an authoring language widely used in industrial training.

The approach taken is to present a description of each package which may then be used for comparative purposes. The strengths and weaknesses of each package will be noted under the appropriate headings. The information provided is given so that the reader may draw his/her own conclusions about the appropriateness of each package in the context of their individual requirements.

The TenCore Package.

Overview.

TenCore is essentially a microcomputer based implementation of the TUTOR Authoring Language used on the American PLATO project. It is claimed by the manufacturers, the Computer Teaching Corporation, that TenCORE provides "...a complete computer based instruction development system for the IBM Personal Computer." The version reviewed here (Version 3.1) supports IBM PC/XT/AT and System/2 computers and all of the commonly used IBM display adapters.

TenCORE provides an integrated graphics editor and character editor to complement the command based authoring language. The attributes of TenCORE are considered below, in terms of Power, Productivity and Ease of Use. Pedagogic freedom will be discussed as a separate issue. The P.P.E. model of TenCORE is shown in Figure 2.5.

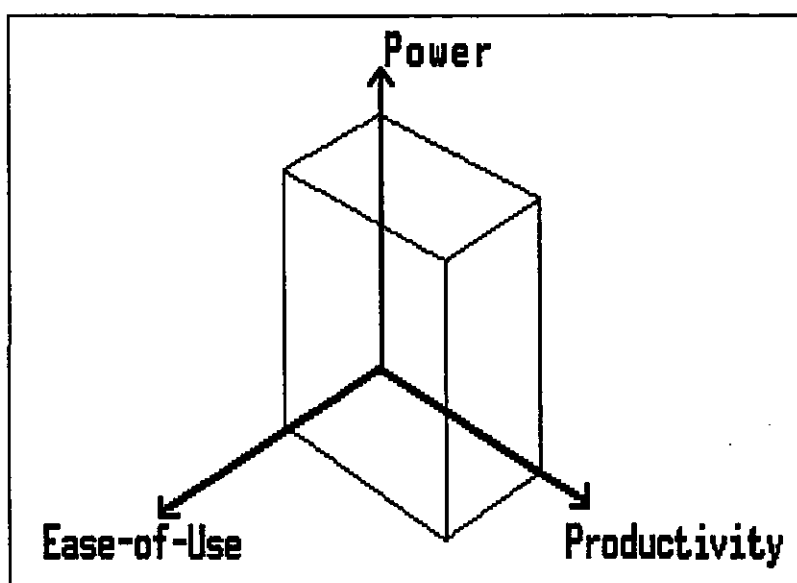


Figure 2.5

Power.

The TenCore language contains over 150 English-like instructions (see Appendix B). Whilst these are primarily concerned with supporting instructional type functions, many of the commands provided are characteristic of a general purpose programming language. The TenCORE instruction set was written in assembly language. Klass (1984) claims that the TenCORE instruction set "...is also powerful enough to be used for all the TenCORE editors and utilities", a fact that testifies to the power of the language.

The Text and Graphics Display capabilities of TenCore are impressive. The author is able to address and write characters to any location on the screen using an 80 x 25 text grid or a 640 x 350 (in EGA mode) pixel grid. Characters and their backgrounds may be displayed in as many colours as the display supports. Text can be presented in any of 4 standard sizes, with fixed or proportional spacing. Text displays can include superscripts, subscripts and accent marks. User-defined characters are also possible.

Graphics commands are provided to handle dots, lines, boxes, circles and ellipses. Windows and text overlays are supported, allowing easy programming of complex screen designs. Additionally, TenCORE can directly interface with the P.C. Paintbrush drawing program, and can call drawings and diagrams created in this way directly into a program as a bit-mapped graphic image.

Response to student input is extremely flexible. When dealing with student inputted text, say in response to a question, the author can choose to:

- specify word(s) that must be present
- specify words that may be ignored
- accept incorrect spelling of key words
- ignore the case of text input
- accept extra words
- ignore word order

Any number of responses may be specified, along with the action to be taken for each anticipated response.

Program Flow & Execution may be controlled by student input or program generated conditions. Simple control of program flow is by means of NEXT and BACK commands, initiated by the user as single key presses, allowing a previous display or the next display in a sequence to be seen. Branching strategies are easily implemented, allowing a degree of individualisation in terms of feedback to responses, or as a student initiated branch by designing a menu selection interface as part of the program.

By making full use of the 'block' structure that the TenCORE language imposes, it is possible to produce a very modular and structured program.

Stand-alone versions of a TenCORE program may be run by using the 'student executor' utility. This allows programs to be run from a floppy disk away from the authoring environment in a 'stand alone' form. To use this facility for courseware publishing, a separate licence must be purchased.

Mathematical Operations are fully supported in TenCORE. Arithmetic operations of addition, subtraction, multiplication and division (in real and integer form), raising to a power and degree to radian conversions are all easily used. Built in functions are available for Sin, Cos, Tan (and Arcsin, Arccos, Arctan) and logarithmic (natural and base ten) operations. Numerical data may be in 1 to 8 bit integer, or 32/64 bit floating point format.

Data Storage. TenCORE allows the use of both local and global variables in a program. Variables must be defined as being either integer, real or buffer (string) types. In addition, contiguous sets of variables may be defined to form arrays. Provision is also made for the storage of data on disk by using 'Namesets,' which provide a random access file of user-defined sets of records. Basic operations on these Nameset files include file creation, data entry, data manipulation and use of data from a disk file by the main program.

Peripheral Devices. TenCore supports an extensive range of external devices, a library of device driver files are supplied with the system. The appropriate files must be installed before authoring Language is used.

- **Display Drivers** are supplied for all IBM standard graphics cards, i.e. CGA, EGA (low and high resolution) MCGA and VGA (used on IBM PS/2 computers). Additionally, the following graphics cards are supported: VEGA, Tecmar, Hercules.

- **Input Devices.** In addition to the standard keyboard, TenCORE supports mouse and touch screen control. TenCORE allows tutorials to be designed that make use of interactive video disk (I.V.) technology. I.V. Systems that are supported include: Visage, Micro Key, MIC and IBM Info-Window.

Productivity

The first impression gained when writing a lesson with TenCORE is that TenCORE programs are produced in a similar way to the process of programming with a general purpose language: source code is entered using a line editor. Whilst most of the code for a program *could* be entered in this way, a number of productivity tools are integrated into the system to enhance the authoring process:

- A graphics editor allows objects to be drawn on the screen using the cursor, a mouse or a light pen. When objects are drawn, the appropriate TenCORE code is then automatically "inserted" into the source code.
- A character set editor is provided. This allows the author to define alternate characters for each keyboard character. The alternate characters can be used at any time in a program, simply by prefixing the character with a special code (ESC and F).

Whilst entering a program into the computer, the author can choose to observe how a program runs and incorporate modifications using a 'test and revise' method of working. This process is very fast, and does not require the code to be compiled each time it is tested. By toggling between 'programmer' and 'student' modes, the authoring process is considerably enhanced. An on-line reference is provided that allows the syntax and use of any TenCORE instruction to be checked whilst using the

line editor, thus avoiding the need to frequently refer to the manual.

When program development is complete, it is a simple matter to produce a stand alone version of the lesson (called a 'binary'), which allows lessons to be used away from the authoring program.

Ease of Use.

As with most programming systems that use a command-line interface, the authoring method employed by TenCORE is not easily mastered by novices. However, those with some previous programming experience should find that TenCORE makes the production of educational software significantly easier than is possible when using a general purpose language.

The reward for perseverance with TenCORE is that once mastered, programs of significant complexity can be written quickly, employing a wide range of advanced techniques (animation, windows , mouse input etc.).

Pedagogic Freedom.

Many of the commands in the TenCore language are reminiscent of instructional learning strategies. However, as the production method provided by TenCORE is open, i.e. not guided by a pre-determined menu/prompt dialogue, the author is not 'locked' into using a pre-determined teaching strategy. The sample programs that are provided with TenCore demonstrate the ability to program simulations and laboratory experiments, as well as drill and practice material, although to be fair, a high degree of mastery of TenCORE programming is needed to produce complex simulation programs.

The TopClass Package.

Overview.

TopClass is probably the most widely used authoring package by educational establishments in the United Kingdom. Like TencORE, the TopClass package is available for the IBM PC/XT/AT family of personal computers. The version of TopClass reviewed here is release 2.3.

Rather than being a single authoring program, TopClass provides an authoring "toolkit" of nine different programs that may be used to aid the courseware production process.

TopClass permits the use of two distinctly different authoring strategies. Firstly, by using the 'CREATE' program, TopClass operates as an authoring system, providing an automatic program generator. Secondly, by using 'EDITOR', the author can write a program directly, using TopClass as an authoring language. The availability of these two modes of authoring is reflected in figures 2.6a and 2.6b.

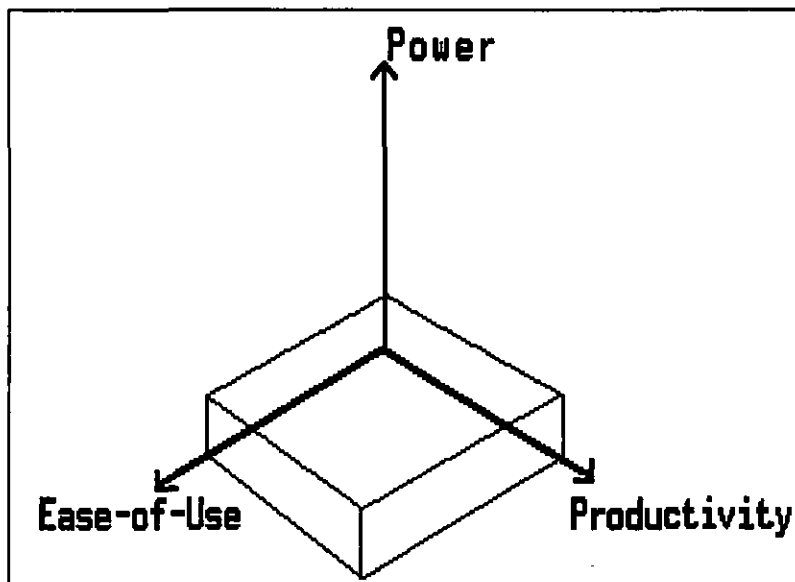


Figure 2.6a

Figure 2.6a: TopClass as an Authoring System.

CREATE is a menu/prompt driven program that allows courseware to be created simply by filling in blank screens, allowing the author to generate a series of text and graphics frames on a What You See Is What You Get (WYSIWYG) basis. The CREATE program automatically converts the screens produced into TopClass language source code.

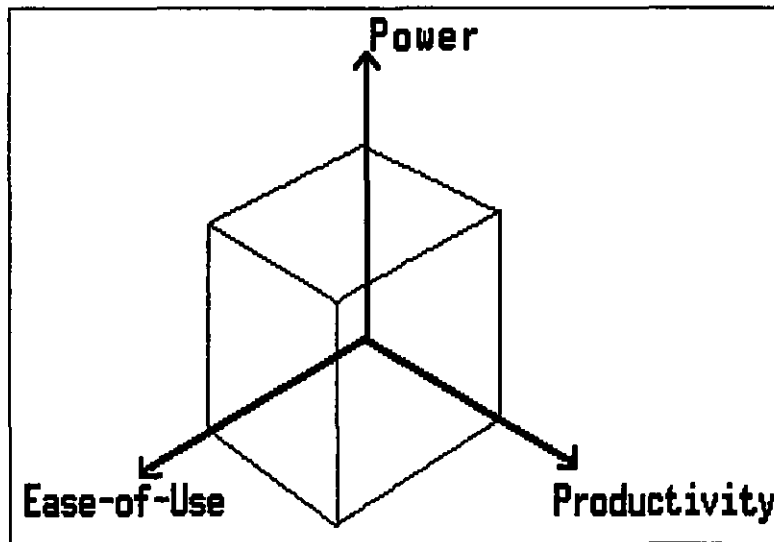


Figure 2.6b

Figure 2.6b: TopClass as an Authoring Language.

EDITOR is essentially a text editing program, that allows more advanced users to make modifications to the code produced by CREATE. Alternatively, EDITOR allows the more advanced author to produce courseware by directly entering Topclass code in an authoring language style of working.

Power.

The TopClass instruction set comprises of around 60 commands, that are primarily geared to producing instructional type CAL material (See Appendix C). The TopClass instruction set is written in compiled BASIC. On first inspection, the range of instructions available are rather limiting when compared with the range of

instructions to be found in a general purpose programming language. However, a closer look reveals a number of powerful commands that would normally have to be manually written into the code by the author have been provided.

Text and Graphics Displays. Screen presentation must be defined as being either text only or graphics mode (which may include text). Text screens are limited to the standard ASCII characters which may only be in one of two standard sizes corresponding to 40 or 80 column display mode. Different colours may be used, depending on the capability of the display device and graphics adapter fitted.

Graphics displays may be drawn using a limited range of commands that allow lines, boxes and circles to be generated. Graphics screens can also include standard text. At the time of writing TopClass only supports 640x200 line (low resolution) EGA displays, but the manufacturers claim that high resolution (640 x 350) graphics will be supported in a future release.

A graphics editor called 'IMAGE' is provided in the suite of TopClass programs that allows the author to:

- generate and edit images at pixel level
- combine several images into one large image
- enlarge and reduce images

To allow images created by other drawing programs (e.g. P.C. Paintbrush) to be incorporated into a TopClass program a utility called SNAPSHOT is provided. This is a memory resident program that allows text or graphics from another program to be 'captured' for later use in a TopClass program via the IMAGE program.

Response to Student Input is dealt with when authoring in CREATE mode by selecting pre-defined strategies from a menu. Strategies may be chosen to test student inputted text for: exact matching with a prescribed response; key word matching; specifying a list of acceptable key words; or using a 'soundex' facility that tests student input phonetically.

TopClass limits the number of pre-defined responses that can be specified to a maximum of 16 correct and 16 incorrect answers. Each specified response can be dealt with individually, allowing some scope for making a CAL program adaptive to different students needs using a branching program.

Program Flow & Execution. When using CREATE to produce programs, only two types of program flow can be used: linear or branching. Obviously this severely restricts the teaching strategy that may be implemented (this is discussed under 'Pedagogic Freedom', below).

Stand alone program disks for student use can be made with the RUNTIME utility. Because TopClass stores its source code in ASCII format, a utility called 'ENCODE' is supplied to render the program source code unreadable to anything but the RUNTIME program, ensuring program security.

Mathematical Operations are available by using the information stores to be manipulated. The functions supported include: Sin, Cos, Tan (and their inverses), logarithmic operations (natural and base ten), random number generation and factorials.

Data Storage is limited to the use of up to 120 so-called information stores, which can hold data in text or numerical form. The TopClass manual does not provide

specific details about the amount of data that an information store can contain or if both real and integer numbers can be dealt with.

Peripheral Devices. When used as an authoring language TopClass can support the following devices as part of a CAL program:

- Interactive video disk, if the computer is fitted with a MIC controller card.
- Interactive video tape, if the tape player is able to 'communicate' with the computer. This implies that a professional specification tape player must be used.
- Interactive audio tape -a separate system supplied by Format P.C. Ltd.
- Digitised speech is available when the "TopTalker" system is purchased with TopClass.

Unfortunately, the display driving capability of TopClass is limited. At the present time, TopClass can only use MDA, CGA and low resolution EGA displays. The lack of support for 350 line EGA, and more importantly, the VGA display standard is a definite limitation of TopClass's graphics power.

Productivity.

As already discussed, TopClass is not a single program, but a collection of program creation utilities. New users of TopClass find that the automatic program generation provided by CREATE allows courseware to be produced very quickly with very little effort. This fact is the key to a major weakness of TopClass: novice users fall into the trap of producing low quality software. It is important to plan any program very carefully to avoid this happening.

In terms of its productivity, CREATE suffers from another significant draw-back: it is not possible to test and revise material whilst programming. When programming using CREATE, the process is one way. If a mistake is made during the authoring process, only two courses of action are available to the author:

- a) start over again from the beginning, or
- b) use EDITOR to correct mistakes at source code level.

Clearly option b) is of no value to a novice user, although it might provide an incentive to learn how to use the EDITOR. Option a) on the other hand, represents very un-productive way of working.

Many TopClass users report that the most productive way of authoring CAL courseware is to write the basic program using CREATE, and then use EDITOR to deal with the more involved programming tasks, and so 'fine-tune' the program. Unfortunately, relying on the CREATE utility as a program 'outliner', or design aid, tends to produce a bland, un-imaginative program.

To assist in the preparation of graphics images TopClass has a powerful image editor that allows the creation and manipulation of graphic images at pixel level. A memory resident program called SNAPSHOT can be used to capture images produced by other programs, and used as part of a TopClass presentation.

As aids to the authoring process, other utilities provided are:

- TOPCOPY: a program that allows novices to use MS-DOS via a menu driven interface.
- RUNTIME: which executes programs that are in TopClass code.

QBANK: a utility for holding a bank of questions on disk, which may be used in any TopClass program.

TCFONT: a font generating/editing tool.

READFILE: provides a datafile on disk of the students responses to questions asked as part of a tutorial.

Ease of Use.

The menu/prompt driven interface employed by ALL of the programs in the TopClass suite allows those with little or no previous computer experience to use TopClass quickly and easily. However, over a length of time over which experience is gradually acquired, users tend to find that working via the menu interface is a slow and tedious process. Generally speaking it is not possible to take short cuts and by-pass the menus. This serves to frustrate the experienced author.

The solution to this frustration is to program a lesson directly, using the TopClass language and the EDITOR utility. Fortunately, learning to author using the TopClass language can be an easy process: once a program has been written, it is possible to run it in single step mode AND see the source code on the screen as the program runs: this makes greatly assists the process of learning TopClass instruction set.

The overall accessibility of the TopClass package is severely restricted by a poorly written manual, which is seen as un-friendly by most new users. The total lack of a tutorial or introductory booklet compounds the problem. As a result, I suspect that many of the TopClass packages sold remain on the bookshelf rather than in use on a computer.

Pedagogic Freedom.

Users of the TopClass CREATE program are locked into a programming environment that is biased towards the production of either linear or branching instructional software. To exercise pedagogic freedom, authors must use EDITOR to program in TopClass source code. Even with this approach, the instruction set available is limiting enough to make any strategy other than instructional difficult to program.

Contrasting TopClass and TenCORE.

Although the TopClass and TenCORE packages both aim to provide a production system for those intending to design and program CBT software, there are marked differences between the two. These differences may be summarised as follows:

- for the novice author, TopClass is a more accessible package than TenCORE. Its build-a-screen approach allows a simple program to be produced very easily. Unfortunately, this easy-to-use philosophy is responsible for the pedagogic restrictions imposed by the TopClass system: with only linear or branching programmed learning strategies directly catered for, the limits of the package are all too quickly reached.
- TenCORE, on the other hand, is not an easy package to get started with, particularly for those who have no previous computer programming experience. However, once mastered, TenCORE offers a powerful CBT software production system that does not limit pedagogic freedom. Furthermore, its programming language-like features make it an excellent system for large CBT projects, particularly if programming is undertaken by a team rather than individual author.

Section Three: An Authoring Package Survey.

To conclude this chapter on authoring packages, this section presents the results of a survey carried out to determine the availability of commercially produced authoring systems within the U.K.

The intention here is not to present a comprehensive description of each package, but merely to indicate availability and provide an overview of each. The full name and address of the supplier of each system surveyed is given in Appendix A for those who wish to find out more information about a particular system.

Crystal.

Unlike the other packages reviewed here, Crystal is really a rule-based expert system creation package rather than a true authoring package. However, as an information handling system it is admirably suited to the task of producing computer based training materials.

By making extensive use of menus, the program creation interface of Crystal is suitable for novices to use. Options are provided to allow graphics to be produced with Crystals own "Screen Painter", or to import a bit-mapped image created by an external package. Furthermore, Crystal allows the use of all of the M.I.C. commands, thereby supporting the production of interactive video based material.

Keen to gain a foothold in the educational sector the producers, Intelligent Environments Ltd., offer an educational package which includes a full development system, nine student licences and sample projects for £1995. Crystal is suitable for use on all IBM PC and compatible machines.

Mentor II.

The Mentor II authoring system is a completely menu and prompt driven package. As such, it is a production system that is easy to use for those with little or no previous computer or programming experience.

The system requires that text and graphics screens are created separately using the appropriate part of the system (with "Textedit" or "Grafedit" as appropriate). Completed screens are then integrated into a complete program using the "Builder" facility. Because of this *modus operandi* a certain amount of planning is advised before programming commences: it is difficult to write a program as-you-go-along with Mentor II !

As with many other systems that are menu and prompt driven, Mentor's ease of use would seem to be at the expense of flexibility. Programming mistakes are not readily rectified without getting involved in programming at code level.

Mentor II is produced for the IBM PC and PS/2 range of computers by Mentor Interactive Training Ltd., and costs £2500.

Microtext.

Microtext is an authoring language for producing "frame based" instructional material. Versions are available for both the IBM P.C. and BBC computer systems.

The production of a Microtext program is centred around the generation of a series of screen based modules. Because the Microtext instruction set resembles the BASIC programming language, it imposes no structure on the program produces, and hence is considered to be more suited to smaller CBT programming projects.

The instructions for generating graphics based screens are minimal, however additional utilities are available from the supplier that allow graphic images to be produced externally and then imported into Microtext. Similarly, a utility program is available to allow interactive video images to be incorporated into a Microtext program.

Microtext was designed at the National Physics Laboratory, and later developed and produced by Transdata of London. The basic system costs £570.

ProCAL.

This package is essentially an authoring language, based on two main programs: 1) a line editor for writing programs in source code form; and 2) a command interpreter for translating programs written in PROCAL command language into an executable CAL program. Schofield (in Strawford) reports that the range of commands provided are comprehensive, and allows control of video tape and laser disk players.

In addition to the ProCAL Editor and Interpreter, a range of other utilities are provided to aid the authoring process. "Test" and "Check" are two programs that are used for testing a program module and checking command language syntax. "Draw" is a graphics program to allow the production of graphic based presentations, although these are limited to CGA and low resolution EGA standards.

Written mainly in compiled BASIC, the speed of PROCAL produced CAL software can be rather slow at times, especially when graphics are used.

ProCAL is produced by VPS Interactive Ltd., the system costs £1350.

TenCORE.

TenCORE is an authoring language based on an enhancement of the instruction set developed for the PLATO project authoring language, called "TUTOR".

The TenCORE instruction set is comprehensive, and offers a range of over 150 English-like commands. Programs are written by entering commands into a line editor or by designing screens using a graphics editor. Graphics images may also be produced using the PC Paintbrush program and then imported into a TenCORE program at a later stage.

The open programming language nature of TenCORE means that courseware design is not limited to the style of 'traditional' CBT. In fact, TenCORE is widely used in industry for producing complex simulation programs.

TenCORE is designed to run on the I.B.M. PC and PS/2 range of computers. It is distributed in the U.K. by Systems Interactive Ltd. of London and costs £2050.

Top Class.

Top Class can be, depending on its mode of operation, both an authoring language *and* an authoring system. As an authoring system it offers an easy to use screen based tutorial production system. Authoring is completely menu and prompt driven, and as such does not require the author to be familiar with computer programming in order to produce simple presentations.

As an authoring language, Top Class provides an easy to learn and understand instruction set. Programs may be written by using the text editor provided, and executed using the "Runtime" interpreter. Utilities for generating graphics, text fonts, and storing question banks are all included in the package.

The manufacturers also produce a range of hardware add-ons: a digitised speech system; CD-ROM facility and video disk capability.

Versions are available for the BBC Micro and the I.B.M. PC family of computers. TopClass is produced and distributed by Format P.C. Ltd. of Belper and costs £495 for the IBM version and £64 for the BBC version.

Unison.

Produced by an ex-member of the Plato project team from the University of Illinois, Unison, like TenCORE, is based on an implementation of the TUTOR language. Indeed, at language level, Unison and TenCORE are almost identical in appearance.

The Unison package is minimal - a language compiler, graphics editor and font editor. A text editor is not included, and so a separate editor or word processor must be used for producing programs. Unison source code must first be compiled before a program can be seen working. This makes a test-and-revise method of working time consuming and difficult.

Unlike most other authoring packages, Unison does not require an additional licence to be purchased for courseware publication. Despite its cumbersome nature, the package is an attractive proposition, it is competitively priced at only £400. Unison is available in the U.K. from Castle Learning Systems, of Chelmsford.

CHAPTER 3.

Educational Issues for CAL Authors.

To be seen as educationally worthwhile, CAL software must not only be technically sound but must also employ appropriate teaching strategies. Clearly, with this latter point in mind, it is vitally important that those responsible for the design of courseware are acquainted with the issues that influence educational quality. Hence, the purpose of this chapter is to identify and examine the important factors that should influence the designers of educational software.

The CAL Environment.

In order to identify the educational issues that are relevant to CAL authors, consider the model shown in Figure 3.1.

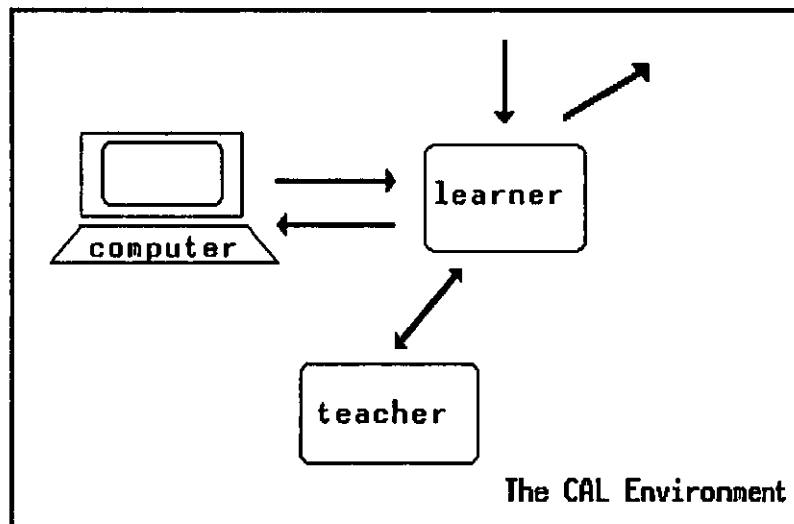


Figure 3.1

The model illustrates the principal components found in the CAL environment, namely:

The computer. Depending on the application, this element of the environment may range from the "standard" visual display/ keyboard combination through to a multi-media system equipped with videodisk player, touch-screen and other specialised peripheral devices.

The learner, who may, at any given time, be interacting with the computer, the teacher or perhaps other learning resources, as the learning situation (or the learners motivation) dictates.

The teacher, whose activities may encompass interacting with a group of students, offering advice to individuals, and ensuring the overall effectiveness of the learner's experiences.

It must be recognised that this simple model has limitations. There are many other influences, not accounted for, that affect the dynamics of the learning environment: the presence of other learners and the intrusion of distracting noises, for example. Nevertheless, this simple model does allow the important educational issues to be identified. Basically, CAL authors will need:

- a) a knowledge of the relevant theories of learning, in order to be equipped to design courseware that utilises an appropriate educational strategy.
- b) an awareness of the characteristics and needs of the learner for whom the courseware is intended.
- c) an understanding of the processes and range of activities that may occur in a computer orientated learning environment.

Thus, in order to provide an educational perspective for CAL authors, the following themes will be considered further:

- The Theories of Learning,
- Modelling the Learner, and
- Learning and Computers.

It must be stressed that although these three themes are examined separately, they are closely inter-related in the CAL design process. Each has an important part to play in influencing the design of educational software.

The Theories of Learning.

Understanding the nature of learning is a central issue in education, yet identifying a single, adequate theory of learning is difficult, as Hilgard points out:

"... no one has succeeded in providing a system invulnerable to criticism. The construction of a fully satisfactory theory of learning is likely to remain for a long time an uncompleted task."

(Hilgard 1958, p.14)

Although written some thirty years ago, Hilgard's words are just as valid today. There is no single theory of learning that can confidently claim to provide the definitive explanation.

The historical development of learning theories, as Romiszowski(1981, p 165) points out, "...has been eventful and colourful, marked by a series of feuds between partisan groups fighting under different names in different epochs." He likens the evolution of the various theories to a "pendulum" effect, as opposing groups in each "epoch" contest each others views: associationists against humanists, the connectionists against the

"gestalt" school, and most recently, behaviourists against cognitive psychologists.

To represent the contribution made by the various theories of learning, only the recognised main-stream contemporary views will be considered. Specifically:

- the Behaviourist viewpoint, and the work of Skinner.
- the Neo-Behaviourist views of Gagné.
- the views of the Cognitive School: Ausubel and Bruner.

Before going further, we must recognise that the theories examined below are not specifically intended to be applied to the computer based learning situation. Working under this caveat, and by applying a measure of interpretation, the theories considered provide CAL authors with an informative perspective.

The Behaviourist Viewpoint.

This viewpoint is closely associated with the work of B.F. Skinner, whose approach to instruction treats the learner as a black-box. Skinner simplistically defines learning in terms of an observable change in behaviour that is not caused by physical maturation or growth (Skinner 1961).

Skinner's views are based on a refinement of Thorndike's "Law of effect," which was in turn based on a belief that human behaviour could be analysed in terms of simple stimulus-response (S-R) sequences. Thorndike's "Law of Effect" stated that if a S-R sequence was immediately followed by a pleasurable experience then it would be increasingly likely that the same response would occur again upon presenting the same stimulus. Conversely, if a S-R sequence was immediately followed by an unpleasant

experience, then it would follow that upon presenting the same stimulus the same response would be less likely to result.

Instruction is described by Skinner as the process of conditioning the learner to produce a desired behaviour, a process that he refers to as "operant conditioning." Operant conditioning can be distinguished from the Pavlovian "classical conditioning" thus: in operant conditioning the learner acquires entirely new behaviour patterns as a result of prolonged and repetitive S-R training. Classical conditioning however, simply causes the learner to exhibit natural (in-born) behaviour patterns.

To facilitate the learning of complex behaviours, Skinner stipulates that the desired end result should be reached by reducing the task to a number of smaller intermediate stages. According to Skinner it is essential that the instructor identifies a specific existing behaviour pattern in the learner before commencing instruction. Having established a starting point, the learner's behaviour can then be modified step by step towards a desired final state.

In developing his theory of instruction, Skinner emphasised the importance of positive feedback following a learner's response to a stimulus (he calls this a "reinforcer"). The job of the instructor, essentially, is to arrange a sequence of stimuli, observe the learner's response to each, and administer positive feedback to reinforce the correct responses. Skinner considered that his approach to instruction was admirably suited to automation, as his development of teaching machines in the 1950's demonstrated.

Even today, Skinner's views on instruction may be seen at work in many so-called "drill and practice" type

CAL programs. More often than not, these examples of the Skinnerian legacy are nothing more than computerised versions of their machine based programmed learning counterparts. As such, these drill and practice programs do little to demonstrate the true capabilities of the computer as a learning tool.

The Neo-Behaviourist Viewpoint.

This is best exemplified by the work of R.M. Gagné, presented in "The Conditions of Learning and Theory of Instruction." (Gagné, 1985). This book has, over a succession of four editions from 1965 through to 1985, revealed a gradual shift in Gagné's stance, moving from a behavioural base to take on board some of the central beliefs of the cognitive school.

Two significant aspects of Gagné's work reveal how his views may be distinguished from those of true behaviourism:

- he recognises a number of different "learning-categories", which he associates with different instructional strategies. This point is elaborated on below.
- in contrast to the behaviourists, he places some importance on the internal mental processes of the learner.

Gagné defines learning as "...a change in human disposition or capability that persists over a period of time and is not simply ascribable to processes of growth." (Gagné 1985, p.2). As already mentioned, he identifies and describes eight different types of learning, which he presents as an ascending hierarchical sequence. The eight types of learning may be summarised as:

1. **Signal Learning.** This term is used to describe a Pavlovian type of conditioned response.

2. **Stimulus-Response Learning.** Unlike the general responses that the learner would exhibit in signal learning, here a more exact response is acquired to certain stimuli.

3. **Chaining,** which describes the learner being able to link a number of S-R events to perform more complex actions.

4. **Verbal association,** which is really a form of chaining S-R associations to form verbal chains from previously acquired S-R associations.

5. **Discrimination Learning.** This describes the learner being able to distinguish between similar stimuli and make the correct response.

6. **Concept Learning.** Here, a learner is able to classify groups of stimuli that, despite their differences, may receive a general response. For example, being able to identify a range of different vehicles all as motor cars, despite their different shapes and colours.

7. **Rule Learning.** By chaining two or more concepts, the learner is able to infer rules like "if X, then Y". Gagné distinguishes this inference process from simple verbal rule memorisation by pointing out that the learner will be able to apply the rule in all relevant situations.

8. **Problem Solving.** Having acquired a range of rules, the learner should naturally be able to use them to perform problem solving. This is done by recalling existing rules and re-combining them to develop a new rule which can be tested. This mode of working is characteristic of discovery learning.

Gagné's approach to learning and instruction places great importance on the teacher as a designer and manager of the learning process. The hierarchical organisation of the eight types of learning demonstrates Gagné's belief that lower order skills must be mastered before the learner can progress to higher order learning.

Gagné describes two alternative strategies that the instructor may use to move the learner from lower to higher learning levels. An exposition tactic requires the instructor to recall lower order rules that have previously been learned, and then provide an example of how these rules might then be employed to solve a new problem. Alternatively, a guided discovery approach requires the instructor to present the learner with a problem to be solved - the learner must then recall relevant rules and hence discover the higher order rule that provides a solution to the problem. In this latter strategy, the instructor would normally guide the process of discovery by offering hints and suggestions to lead the learner towards the solution.

Comparing the expository & discovery strategies, Gagné expresses a preference for guided discovery on the grounds of long term effectiveness. However, he concedes that the exposition strategy may be seen as more attractive because it is less time consuming.

The Cognitive School.

Whereas the behaviourist school perceives learning essentially as a simple stimulus-response event, the cognitive view places great importance on taking into account the mental processes that occur within the learner in order to interpret the process of learning. The cognitive viewpoint is best illustrated by considering the work of two prominent workers in the field: Jerome Bruner and David Ausubel.

Bruner: the Developmental Viewpoint.

Bruner considers learning as a process rather than a product, and in so doing provides an illuminating perspective of what instruction should be:

"Instruction consists of leading the learner through a sequence of statements and restatements of a problem or body of knowledge that increase the learner's ability to grasp, transform, and transfer what he is learning."

(Bruner, 1988).

He stresses the need for the learner to be recognised as an individual, and emphasises that there is no one ideal programme of instruction that will satisfy the needs of every member of a group. He asserts that successful instruction may only be achieved by first considering the individual's requirements in terms of past learning, stage of development, nature of the instructional material and individual differences.

Bruner is a recognised advocate of "discovery learning," which he describes in terms of a developmental process, characterised by three levels of activity through which the learner may progress:

- an enactive level, where the learner manipulates objects directly. Learning to ride a bicycle is an example of this type of activity.
- an iconic level, where the learner operates on mental images of objects rather than manipulate objects directly.
- a symbolic level, where the learner manipulates symbols instead of mental images.

As Romiszowski (1981, p.171) correctly points out, Bruner's views on learning are based on an interpretation of Piaget's work in developmental psychology.

On the issue of feedback, Bruner makes two important observations that CAL authors would do well to take note of. Firstly, if learning or problem solving is taking place in a given mode (i.e. iconic or symbolic) then corrective feedback "...must be provided in that same mode or in one that translates into it." Secondly, he states that the tutor (or in this context CAL author) must ensure that the learner is corrected in a manner that will eventually allow the learner to perform corrective actions himself. Bruner warns that not to do so will produce a learner who is dependent on the constant presence of a tutor.

Ausubel: the subject matter viewpoint.

In contrast to the developmental approach advocated by Bruner, David Ausubel (1983) argues that learning can be performed successfully by what he refers to as "meaningful reception learning." To understand the meaning of this statement, first consider the model shown in Figure 3.2. It represents Ausubel's interpretation of learning, in terms of strategies and outcomes.

meaningful learning	Classification of relationships between concepts	Well designed audio tutorial instruction	Scientific research
intermediate learning	Lectures or textbook presentations	School type laboratory work	Most routine research or intellectual production
rote learning	Times tables	Applying formulas to solve problems	Trial & error "puzzle" solutions
	reception learning	guided discovery	discovery learning

Figure 3.2.

The horizontal axis of the model represents the style of learning to be adopted, and is divided into three bands which range from reception, through guided discovery, to discovery learning. Moving from left to right on this axis, we observe that the control of the learning process shifts from teacher centred through to student centred learning.

Reception and discovery learning may be differentiated between thus: In "reception" learning, the student is given the material to be learned in its final form. In discovery learning however, the learner is required to discover what is to be learned before "internalisation" can take place.

The vertical axis of the model represents the outcome of the learning process, which may range from meaningless (rote) learning to meaningful learning, with a region in between of intermediate (not rote, yet not entirely meaningful) learning. What actually constitutes meaningful or meaningless learning is, according to Ausubel, largely dependent on the learner's existing knowledge, or concept structures, and hence the means to "anchor" new information to these structures in a meaningful way. This is the mechanism by which the learner acquires "new meanings." Rote learning, on the other hand, does not result in the acquisition of new meanings: no logical perception or comprehension is required on behalf of the learner.

Ausubel stresses that reception learning can only be meaningful if the material to be taught is properly structured. Essentially, the teacher must organise topics into a sequence of learning concepts: each should be a link in a chain and provide the basis for the following topic.

Although this approach appears reminiscent of that used by Skinner, there is an important difference. Skinner regards learning in terms of output from the learner. Ausubel, on the other hand, describes learning in terms that relate to providing input to the student.

Ausubel recognises that motivation is of key importance in reception learning. He stresses that meaningful-reception learning must be an active process, with the learner deciding how to categorise the new information and perform recognition with existing knowledge. Such activity must, according to Ausubel, "...stop short of actual discovery or problem solving."

Modelling the Learner.

In the light of the importance placed by cognitive psychologists on the internal processes of learning, it follows that the CAL author should be aware, in some detail, of processes that occur within the learner. In so doing, he/she would then be better equipped to design more effective educational software.

A suitable model for this purpose is put forward by Gagné (1985) and is shown in Figure 3.3 below. The model is not unique, but is used here because it is typical of the many "information processing" models of human cognition that abound.

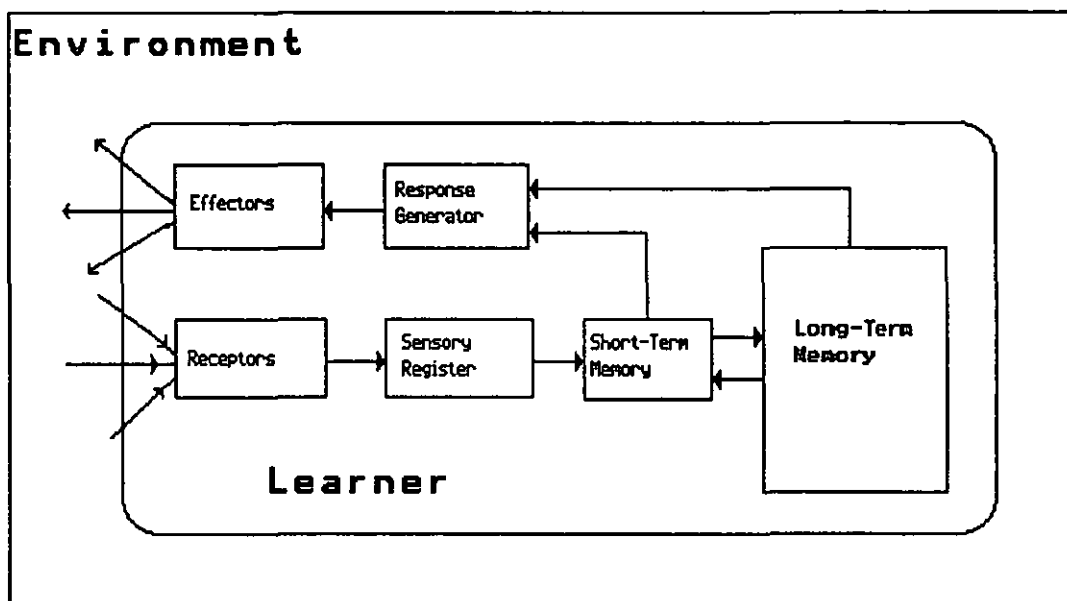


Figure 3.3

Source: Gagné, R.M. (1985)

The component elements of the information processing model are discussed in some detail below.

Sensory Input and Attention

The model shown in Figure 3.3 illustrates the flow of information which is collected from the learners environment via the sensory input from the five senses. This stimulation results in the generation of nervous signals which are registered in a sensory register - a temporary store.

The information in the sensory register does not persist, but is processed into what Gagné refers to as "patterns of stimulation." These patterns may be either processed further if required, or ignored if they are not of interest. It is this selection of information which explains the process of attention. We are continuously receiving information from all of our sensory inputs, yet if all of this information were to be completely processed, the system would become overloaded. To prevent such an overload, some form of filtering must take place,

where only the signals that are to receive attention are selected. Winfield (1986) puts forward a useful two-state model of attention to illustrate the filtering process.

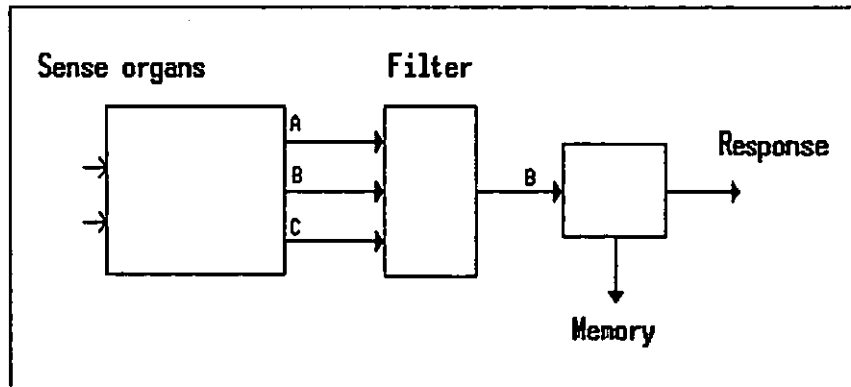


Figure 3.4.

Source: Winfield, I. (1986)

An understanding of the process of attention is important. CAL material will have to compete with many other stimuli in the learners environment. The CAL designer must produce computer software that demands the learners attention.

The Memory Stages

Information received from the selected sensory channels is passed to the first true memory stage, the iconic memory. It is so called because its function is to maintain a mental picture of the sensory signals. Although the image retained by the iconic stage only persists for around 0.5 - 2 seconds, it is invaluable in processes involving reading, for example. An illustration of the action of iconic memory is experienced when we perceive the apparent smooth movement of images displayed on television or cinema screens, which are actually rapid sequences of differing static pictures.

Another example of iconic memory in action may be seen by giving a subject a brief glimpse of an image (a map, for example). When asked questions about the map the subject will search and find information from their iconic image. Iconic memory holds more than just visual data, it also stores 'images' of auditory stimulation, for example.

Having captured an image, it is then passed to short term memory (STM) where it may persist for up to 20 seconds. If this information is of immediate interest, it may be kept alive by performing a process known as a "rehearsal loop" whereby it is regularly refreshed to maintain it. An example of this process in action is in remembering a telephone number for a few minutes, by repeating it over and over to oneself until able to write it on a note-pad. Some people vocally perform the rehearsal process to keep information temporarily alive. STM has a limited capacity, only 6 or 7 separate items can be simultaneously stored, hence human information processing is subject to a limited "band-width." The implication for the CAL designer is obvious: do not give the learner too much information to process at any given time (see Figure 3.5).

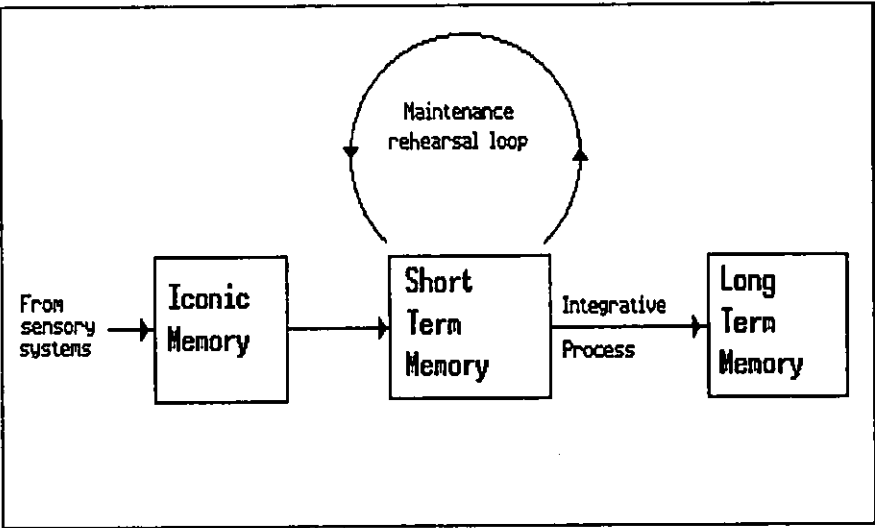


Figure 3.5.
Source: Winfield, I. (1986)

If information is received which is important and must be retained, it is converted into a form which can be held in long term memory (LTM). The process of passing information from STM to LTM is referred to by Gagné as "encoding." He describes this information storage "...not as sounds or shapes, but as concepts, whose meaning is known and can be correctly referenced in the learners environment."

Once the information destined for LTM has been encoded, it undergoes a storage process. In such a process the information, if successful and meaningful storage is to take place, must be linked to existing LTM structures or concepts. Ausubel refers to this linking as the formation of "anchorages" where new LTM information is attached to, or associated with, existing LTM structures.

The final process associated with LTM is that of information retrieval. The key to successful memory recall is in the recognition of the required information when a search pattern is invoked. Assuming that the storage capacity of LTM is infinite then the apparent loss of information is assumed to be a failure to successfully recognise and hence recall that information.

Conscious Memory

In addition to processing incoming signals, STM also functions as a working memory, and in performing recall operations on LTM, it is STM acting as a working memory which holds the recalled information for conscious use. Once recalled into STM, information can be reprocessed for assimilation with other mental structures: this provides an explanation for the act of consciously thinking things through.

Response Generation

Responses to information in the system are generated by driving, or stimulating, the motor-effectors. In so doing this will result in interaction with sensory inputs and thus providing feedback of our action. Such deliberate action is referred to as the purposeful interaction of the learner with the environment.

This last point, in fact, epitomises the need for the CAL author to use a model of the cognitive processes performed by a learner. After all, the purpose of any piece of educational software is to stimulate the purposeful interaction of the learner with his/her environment which, in the CAL context, centres around the computer.

Learning and Computers.

There are many different applications of the computer in the learning situation: tutorial, drill and practice, educational games and simulations are all examples. Although these terms loosely describe the role of a given piece of software, their meanings are ambiguous and open to a wide degree of interpretation: Just what is a "tutorial" program, and how does it differ from "drill-and-practice" software ?

Kemmis et al. (1977) put forward a set of clearly defined categories, which may be used to describe a piece of software in terms of its educational paradigm. Four such paradigms are identified: instructional, revelatory, conjectural and emancipatory. To the educator they offer a useful classification system for CAL software. Each paradigm is presented below.

The Instructional Paradigm.

This mode of CAL embraces Skinner's theory of operant conditioning, where subject matter is broken down into smaller units to be individually mastered. The instructional approach is not new. A similar system was used with mechanical teaching machines, but these lacked the sophistication that the computer offers.

The computer provides the means for a number of important features to be designed-in to instructional CAL programs. For instance, branching, as Crowder demonstrated, will help to inject some degree of flexibility into the learning material, allowing for appropriate responses to be generated, thus accommodating different students abilities.

To be worthwhile, however, instructional programs must involve the user interactively with the learning process. Providing the student with feedback enables this to happen. As Bruner points out, "Learning depends upon knowledge of results at a time and at a place where the knowledge can be used for correction." Clearly, giving such information in a way that is appropriate to the individual learner demands great skill from the CAL designer.

The Revelatory Paradigm

Here, the computer is used to present subject matter and its underlying theory in an ordered manner. This paradigm may be equated to the "guided discovery" dimension of Ausubel's model. The CAL designer is responsible for setting the framework in which discovery can take place, but unlike the instructional paradigm, more control of the learning process is given to learner.

The important difference between this and the previously described paradigm is that some degree of

problem solving activity is required from the learner, in order to discover the material to be learned which must in turn be processed (Ausubel referred to this process as "internalisation").

One of the strengths claimed by advocates of this paradigm is that in using a computer to model or simulate a situation or system, the learner is freed from unnecessary distractions or details which are not needed for learning to take place. While this may be generally true, a decision must be made about what information is or is not necessary for inclusion in a simulation. Indeed, unless the cognitive processing ability of all users is anticipated, some vital information may be denied to the learner.

As all learners are individuals, and should ideally be treated as such, many programs in this category of CAL are far from perfect. However, it should be stated that the method can be usefully employed, particularly if a teacher is available in the learning environment, to provide the additional information and guidance that learners may require that is not available from the computer.

The Conjectural Paradigm

The conjectural paradigm is heavily dependent upon the student being an agent of his/her own learning. Probably given no more than an objective or an outline, the learner must engage in problem solving activity, and therefore must already possess the appropriate conceptual structures for meaningful learning to result. Furthermore, the learner must be suitably motivated to operate in this way in the first place. This approach requires a certain level of maturity (cognitively speaking) on the behalf of the learner.

As with the revelatory paradigm, the presence of a teacher in the learning environment will be useful to provide any information or facilities that are not available via the computer.

The Emancipatory Paradigm

Although perhaps not strictly of relevance to the design of CAL courseware, this paradigm embraces the use of a computer as a tool, allowing the learner to be free of work which has no educational value. The often cited examples of software in this category are databases, graphics packages, spread sheets and the like.

The table shown in Figure 3.6 summarises how a range of types of software equate to these paradigms:

Textbook mode	}	Instructional
Drill and Practice		
Programmed learning (linear)		
Programmed Learning (branching)		
Educational games	}	Revelatory
Simulations		
Problem solving	}	Conjectural (exploratory)
Creative Activities		
Spreadsheets	}	Utility (emancipatory)
Wordprocessors		
Databases		

Figure 3.6

Although the above paradigms offer a useful means with which to view types of software, they all neglect an important fact. They take no direct account of how meaningful the learning process can be. This absence may imply that the meaningfulness of CAL software is not, as yet, taken as being a design consideration by many software designers.

A CAL Model.

Perhaps the simplest model of CAL is one that adopts the approach taken by Chandler (1984), who categorises a range of CAL activities according to the degree of control afforded to the learner by the computer. This "locus of control" is illustrated in Figure 3.7.

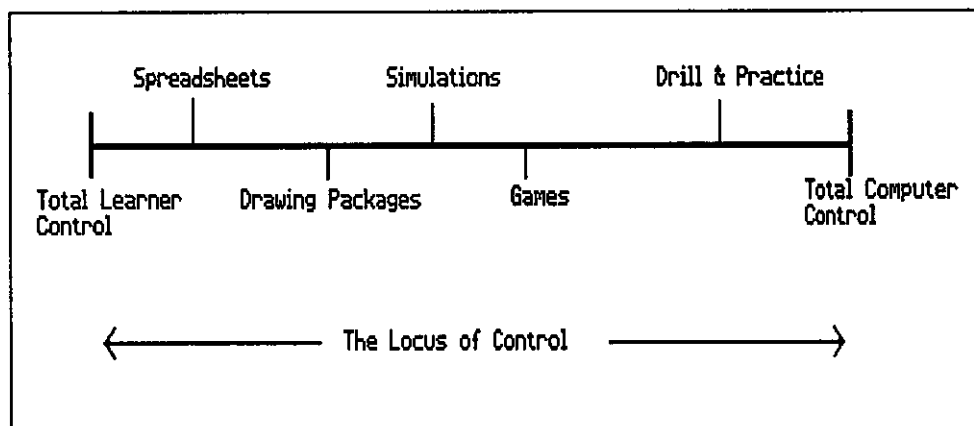


Figure 3.7

Chandler's model is, however, only of limited use. Although it considers the interaction of learner and computer, it fails to recognise the presence of other influences that are part of the CAL environment. To be of value, the model must be extended to reflect the meaningfulness of the learning that may occur.

Extending the Model.

As already discussed, Ausubel emphasises the importance of recognising learning outcome when modelling learning. Taking this point on board, a rote-meaningful dimension can be added to the "locus of control" model, as Figure 3.8 shows.

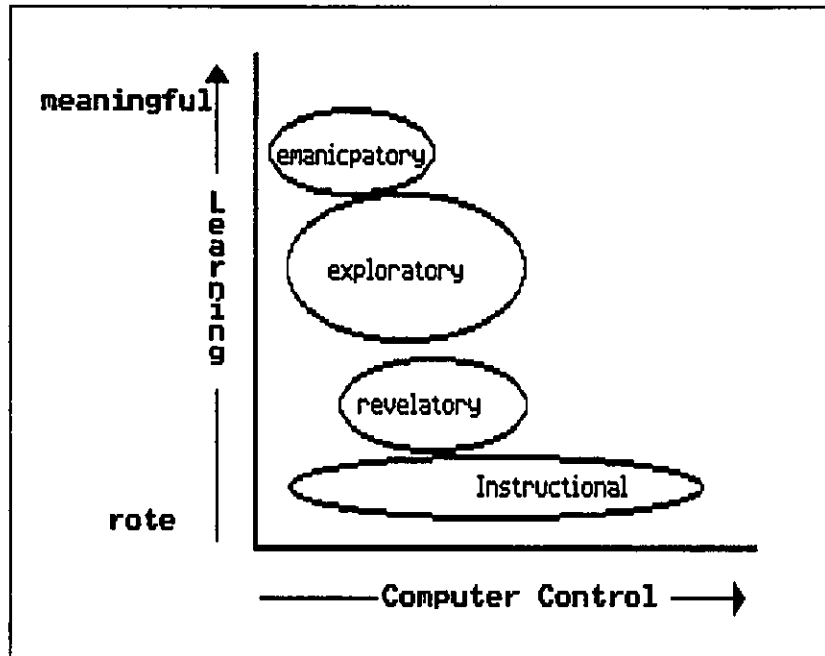


Figure 3.8

The four CAL paradigms have been added for illustrative purposes. It is interesting to note the similarity between the model shown in Figure 3.8 and the two dimensional model that Ausubel uses to describe learning in terms of activities and outcomes.

To allow the model to reflect the influence of the teacher in the learning environment a third axis must be added. This indicates the degree of control exercised in the learning environment by the teacher. As a result the model is shown in its new 'three dimensional' form in Figure 3.9. For illustrative purposes, two of the CAL paradigms identified by Kemmis have been included.

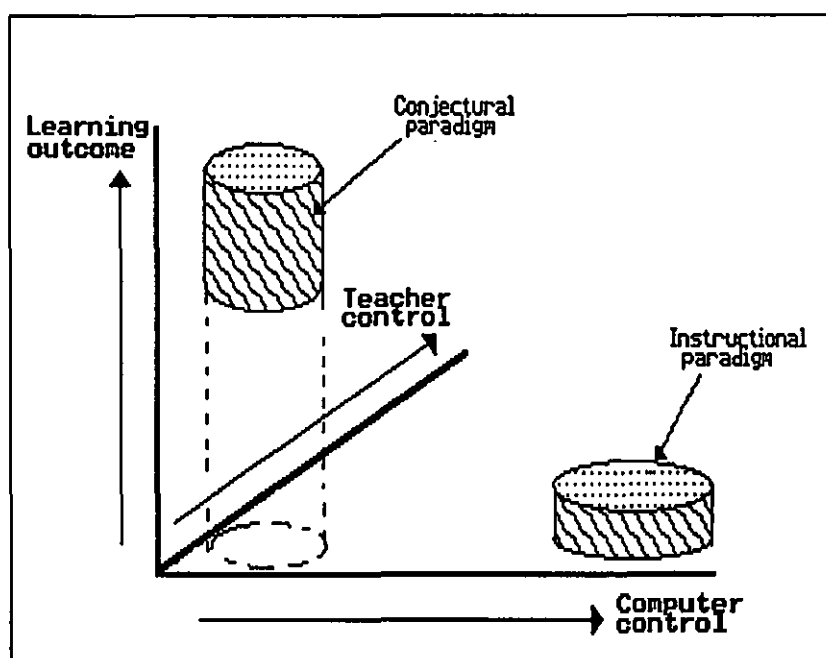


Figure 3.9

It would appear from the model that learning appears to be influenced by two agents: teacher and computer. However, a third agent exists that influences the point of control, i.e. the learner. As Armstrong (1987) demonstrates, accepting the simplifications of the model the balance of control, shared between learner, teacher and can be expressed thus:

$$L + T + C = 1$$

Where L is the proportion of learner control, T is the proportion of teacher control and C is the proportion of computer control. Thus, the amount of learner control may be expressed in terms of teacher and computer control, as shown graphically in Figure 3.10.

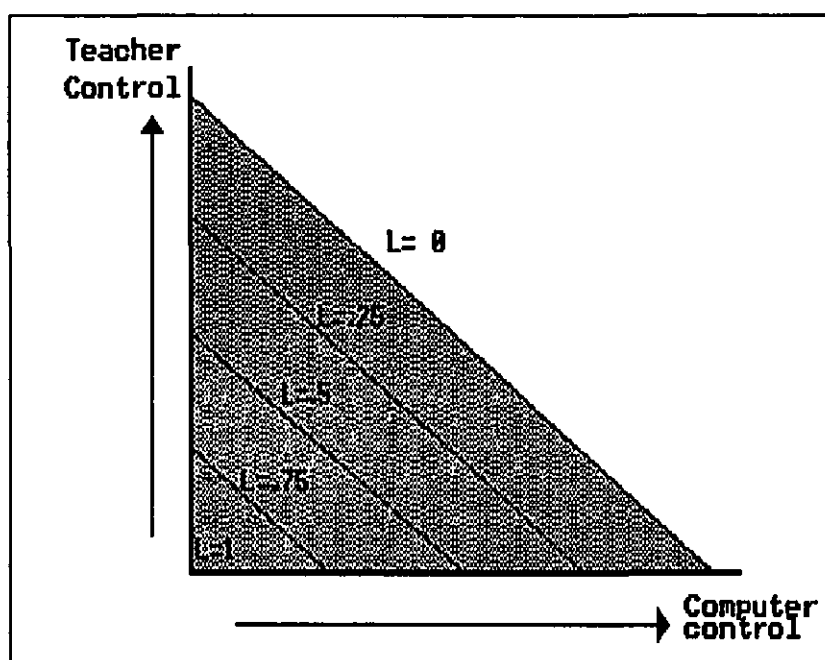


Figure 3.10
Source: Armstrong(1987)

Observations on the CAL Model.

As with any model, the one illustrated in Figure 3.9 has limitations since it can only represent a simplification of the actual learning environment.

The CAL environment is dynamic: the balance of learner-teacher-computer control, and the resultant learning outcome may well change with time as a student uses a piece of software. At one point the learner may be passive, reading instructions from the screen. At other times the learner will be active, perhaps making a menu selection, responding to a question or reading a paragraph of text from the screen.

Similarly, the amount of teacher control experienced by a learner will, over a period time, vary. This simple

model only recognises the presence of one learner, but in reality the teacher's attention may well have to be shared between several learners who, in all probability, will progress at different rates and hence be at different operating points on the model, even though they are all using the same piece of software.

As it stands, then, the model makes the following assumptions:

- that the teacher is only concerned with one learner.
- that the learner has the exclusive use of a computer.
- that the learner's attention is only influenced by the computer and the teacher.

Nevertheless, the model does make an important contribution. It serves to draw attention to the main elements that influence the learning environment. Armed with such an awareness, the author is better equipped to design software that is educationally sound, and relevant to the needs of both teachers and students.

Chapter 4.

Human Factors for CAL Authors.

The Human-Computer Interface.

It is widely accepted that poor interface design can greatly contribute to the failure of a computer system to operate effectively. CAL systems are not immune in this respect:

" Computer assisted learning (CAL) programs, like almost all applications programs written for human use, have been criticised for the design of their user interfaces..."
(Whiting, 1989).

"...the principles of computer ergonomics, dialogue design and user support systems are also of great importance and relevance in the field of Computer Assisted Learning (CAL)."
(Lay, 1981).

Clearly, the understanding, and engineering of the fragile link between man and machine is fundamental to the design of both effective authoring systems and successful CAL software. The purpose of this chapter is to present an exposition of the essential HCI issues that relate to the design of effective learning systems. To this end, the topics discussed here are:

1. The nature of Human-Computer Interaction (HCI);
2. Authoring System: The interface between author and computer;
3. CAL software: The interface between learner and computer.

The Nature of HCI.

The approach taken, in introducing this overview of the HCI process, has been to turn to the work of Donald A. Norman, editor of "User Centred System Design" (Norman, 1986). Norman coins the phrase "cognitive engineering" to embrace a blend of cognitive psychology, science and human factors to apply what is known from these fields to the design and construction of computer interfaces.

From Psychological to Physical.

Assuming that a user (author or student) wants to interact with a computer, then he/she will have personal goals which will stimulate an interaction process. A mismatch immediately arises: The computer user has goals that are psychological in nature, yet the computer's mechanism is physical. This mis-match represents what Norman refers to as a "gulf" between the user's goals and the computer system. If the user's goals are to be satisfied, then the gulf must be bridged.

For an interactive system, Norman points out that two such gulfs exist, i.e. the gulf of execution, across which the user inputs to the computer, and the gulf of evaluation, whereby the user interprets system action. These are illustrated in Figure 4.1.

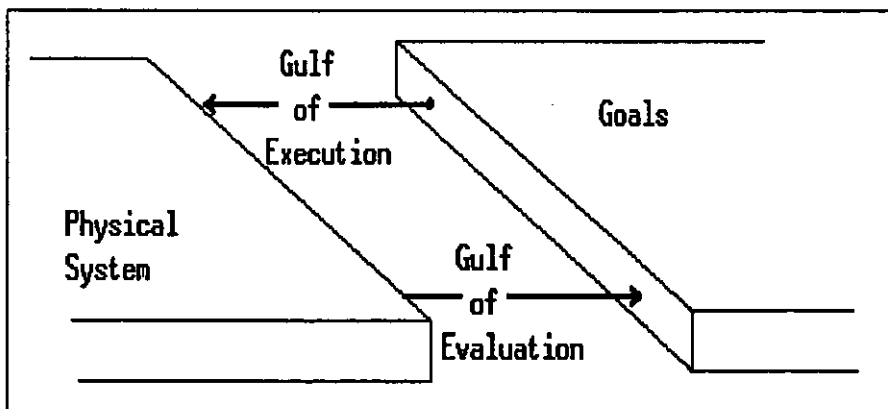


Fig. 4.1

Source: Norman, D.A. (1986).

Norman goes on to suggest that in order to bridge the gulf of execution, a four stage process must occur. This is described below:

1. Intention formation, where interaction with the computer modifies the thoughts of the user, producing goal formation.
2. Specifying the action, where the psychological intentions are translated into the changes that are necessary to produce physical action.
3. Executing the action, which is the first physical stage of the process and results in a psycho-motor sequence.
4. Contact, i.e. making contact with the input mechanism.

Similarly, Norman resolves the gulf of evaluation into a sequence, whereby the computer system is interpreted by the user, and compared with the original goals and intentions. Norman describes the sequence thus:

1. Interface perception, where the display system of the computer stimulates the user's sensory (visual) input, resulting in patterns held in the users iconic memory.
2. Interpretation of these patterns into meaningful conceptual structures which can then be assimilated with existing structures. This is the short term to long term memory process referred to by Gagné (see Chapter 3).
3. Evaluating this information in terms of a comparison with the original intentions.

Figure 4.2 provides an illustration of the step-wise bridging of the gulfs of execution and evaluation for computer interaction to occur.

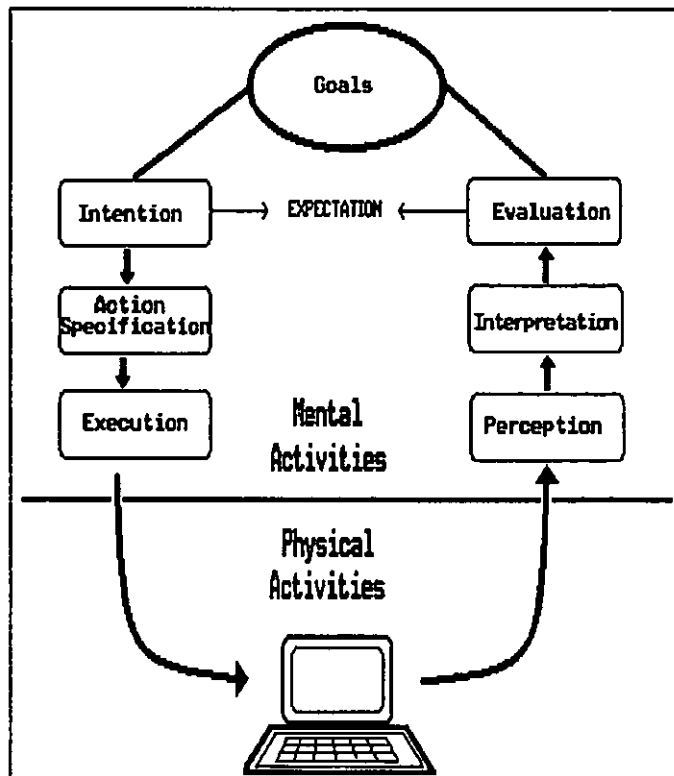


Fig. 4.2

Source: Norman, D.A.(1986).

The implications for the field of CAL in considering the psychological-physical processes may be stated, in simple terms, thus:

- CAL authoring software should employ an interface that allows the author to design and develop software in an easy and productive manner.
- CAL courseware should employ interface designs that minimise interaction effort within the context of the learning task.
- Courseware authors should strive to design systems in which the computer is transparent in the learning process.

Authoring System:

The Interface between Author and Computer.

In accepting the basic premise that an authoring package exists to allow teaching/training experts rather than computer programming experts to produce computer based training software, it follows that the success of an authoring system will, in the eyes of the user, be judged largely on the quality of the interface provided. This quality will, as Lindquist (1985) points out, usually be interpreted in terms of two basic parameters: "effort to learn" and "effort to use." These two parameters provide the basis for investigating the authoring interface further.

The Novice-Expert Problem.

Authoring system users who have a background in training or teaching rather than in computer programming are, at least initially, novice computer users. However, most of these users will, after a period of time during which experience is acquired, become more familiar with the computer. Indeed, over an extended period, some users will go on to accumulate considerable knowledge and skill, becoming experts in the use of a particular system.

The needs of this range of authoring system users, from novices through to experts, differ considerably. Novice users will require the system to present a dialogue style that is predictable and consistent. They will also need to be equipped with a model of the system (a so-called "virtual machine") that is easily conceptualised. Expert users, however, will prefer a fast and versatile dialogue style, allowing abbreviated versions of frequently used instruction sequences: the virtual machine must be geared to performance rather than easy usage.

Thus a conflict arises for interface designers: to devise an interface that can serve the needs of expert and novice users alike.

The findings of Bosser (1987), who has researched the effects of user experience and system performance, are of particular interest. Bosser notes that if an inappropriate interface design is employed, the novice may be penalised. Thus, for the novice authoring system user, difficulties encountered in learning to use the system may interfere with the task of producing courseware due to the user initially lacking the required experience. On the other hand, if an expert user is faced with a system designed to accommodate novices, the fluency of the authoring process will be disrupted by the use of an apparently clumsy interface.

It would appear that designing an interface that accommodates the needs of novice users will inevitably lead to a sacrifice of system functionality and usability at the expense of learning to use the system. However evidence can be found demonstrating that, with careful attention, the interface can be designed to support all three activities. Whiteside et al.(1985), in comparing different user interfaces, found that some systems that were best for novices were also best for experts.

Nevertheless, many new users of authoring systems, particularly those with little or no previous computer experience, find authoring systems to be slow to learn and clumsy to use. Indeed, from the survey carried out in Chapter 2, it is clear that all of the IBM PC based systems reviewed left a lot to be desired in terms of their interface design.

Moving out of the narrow field of authoring system software, good examples of interface design can be found: The Apple Macintosh exemplifies how to design a system that is easy to use and accommodate a range of different user needs. Indeed, the Macintosh interface along with that of the Digitalk Smalltalk V system stimulated the following design exercise.

Designing the Authoring Interface: An Example.

The following study uses a hypothetical authoring system (referred to as "CONCEPT") to provide the basis for discussing the application of cognitive interface design principles to the authoring interface.

In doing so, an interface is described which, it is hoped, will indicate a possible direction for making authoring systems more easily learned and readily used.

The Traditional Authoring Interface.

Typically, it is with a menu and prompt driven system that many new users take their first steps in authoring educational software. This is when the all important first impressions are made. Even when a system that has clear and hierarchically organised menus is used, the newcomer finds the task of converting their goals into outcomes a difficult and frustrating experience. Many users all too easily become disorientated as they are forced to wade through an apparently endless series of menus in order to produce the simplest of tutorial programs.

Essentially the problem is this: none of the current generation of IBM PC based authoring systems appear to have been designed to deliberately provide the user with a mental model, or "virtual machine" which can be employed as a framework for learning and using the system.

Navigating through menus to eventually perform the desired action is a compartmentalised process. Often, the user is given no indication of where he has come from or where he is going. As a result, only those with previous computer experience, or high motivation, or who are already committed to implementing CAL/CBT will persist in learning (or tolerating) the intricacies of a badly designed authoring system and so go on to use the package to its full potential.

Providing a "Virtual Machine."

Iconic interfaces are becoming increasingly common, and for good reason: they represent a deliberate attempt to provide the user with a carefully designed virtual machine. Because computer users often rely on using physical analogies to construct a mental model, icons can be appropriately used to provide the user with a ready made metaphor.

Consider the user interface of the Apple Macintosh, which provides the user with a virtual machine based on a "desk-top" metaphor. Generally, new users can relate to and purposefully use the Macintosh in a far shorter time than would otherwise be possible if a command language interface, such as MS-DOS or UNIX, was employed. This is because of one fundamental fact: the iconic interface requires the user to simply recognise symbols and menu choices, rather than remember commands. In this icon and menu based environment the user can interact with the computer almost as he/she would interact with real world objects.

In addition to achieving the primary objective of making the computer use a natural and easily learned process, there is another benefit in designing-in the virtual machine. The provision of a built-in model ensures that all users of the machine share the same virtual

machine concept, thus allowing a wider discussion of the machine's operation and dissemination of product information in virtual machine terms rather than on a more unfriendly technical basis.

It would, then, seem appropriate to reconsider the design of the authoring interface with the intention of building-in a virtual machine suitable for the needs of courseware authors. The advantages of doing so are clear:

- first time users would see the process of authoring as more friendly, and hence be encouraged to explore and make fuller use of its potential.
- experienced users would find authoring to be a fluent and productive process.
- a good interface design would motivate users to use the package again and again, rather than simply leaving it on the shelf.

CONCEPT: A New Authoring Interface.

The CONCEPT authoring environment represents a possible direction for interface design if cognitive strategies are to be used to improve the interface between author and computer. Because the CONCEPT approach is based on a system with "standard" authoring functions it is possible to apply its philosophy to many authoring systems currently available. It is felt that Top Class, a system widely used in education by those with little or no previous computer experience, could greatly benefit by adopting an interface similar to the design presented below.

The metaphor used for CONCEPT is that of an "author's work-bench" in which the "standard" authoring tools may be found. These tools are grouped thus:

- a lesson building and modifying program
- image generating and editing utilities
- a character font making program
- a data storage facility

Figure 4.3 illustrates how these "standard" authoring utilities may be organised into the structure of the author's work-bench.

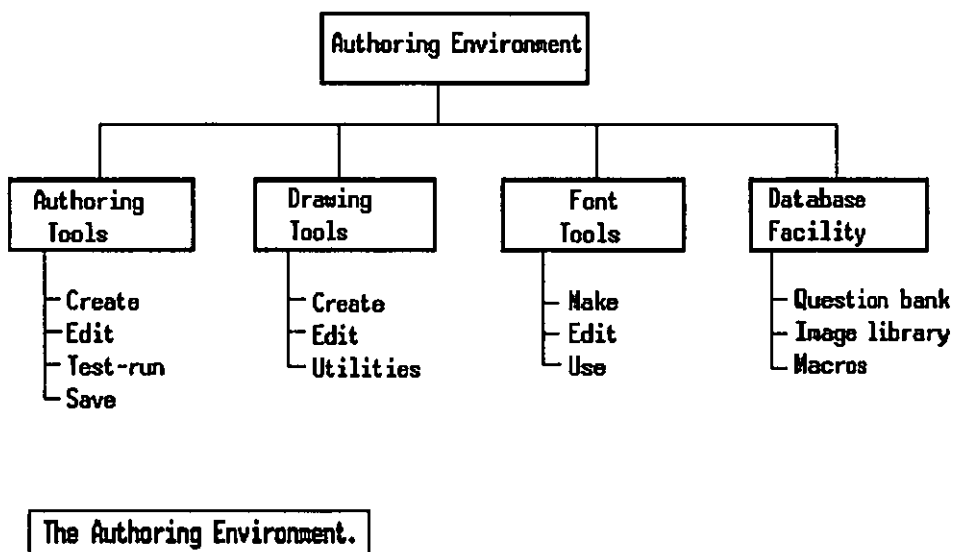


Figure 4.3

Each of the four major authoring functions within the system are presented to the user as icons. Selecting an authoring function is performed by using a mouse and pointer to select an icon. Upon selection, a menu would be presented that allows a further choice as appropriate.

Unlike many menu based systems, such as Top Class, which uses a one-menu-per-screen design, the CONCEPT icons and pop-up menus all appear on the same screen. This way, users can see the path taken through the available choices, and navigate through the authoring system easily and purposefully to achieve their goals.

Using the CONCEPT approach, the user is merely required to recognise and choose the appropriate icon in order to author, and is not burdened with the need to commit command names, menu choice sequences and key presses to memory.

Implementing CONCEPT.

The diagrams on the following pages illustrate the appearance of the CONCEPT interface, as seen by the author. The underlying principles are to:

- replace the "traditional" opening menu (as exemplified by Top Class and TenCORE) with icons to represent the basic authoring processes.
- present concise, relevant menus in a way that reveals the internal structure of the system to the user.
- allow the user to navigate through, and maintain a sense of position within, the system.
- reduce the need for the user to memorise names, codes and rules.
- simplify the interaction process by using a mouse and pointer.

The opening screen of the CONCEPT interface (see Figure 4.4) presents the user with a simple four choice selection task, depending on whether the author wishes to:

- Author, or work on a piece of software.
- Draw, or produce a graphics image to be used in a piece of software.
- create a text Font.
- work with information held in a Database.

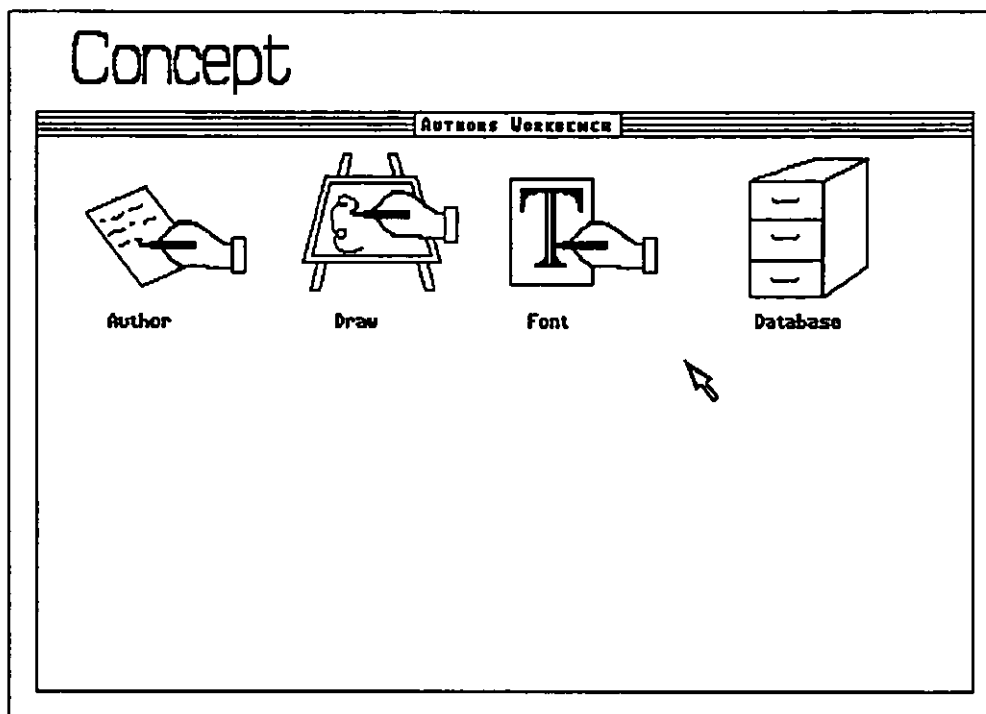


Figure 4.4

Each authoring activity, as the structure chart in Figure 4.3 shows, allows a number of different processes to be performed. Thus, once an activity is chosen, a pop-up menu is presented to the user for the appropriate next choice to be made. An "exit" option is always provided so that the user can close a menu and reverse the last decision made.

To leave the authoring environment and return to the computer operating system, the author simply moves the pointer outside of the work-bench window area and presses a mouse button.

Figures 4.5 through to 4.8 on the following pages illustrate the main screen designs.

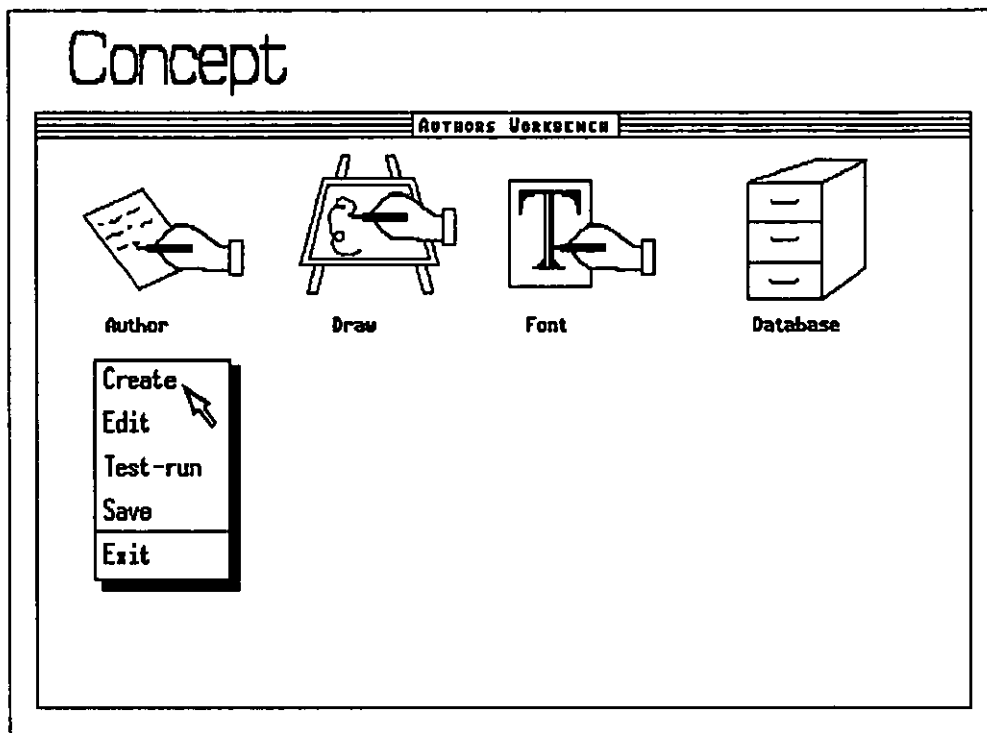


Figure 4.5

Author. (Figure 4.5)

When 'author' is chosen, the user may choose to:

- create a new lesson.
- edit an existing lesson.
- test-run a lesson.
- save a lesson.

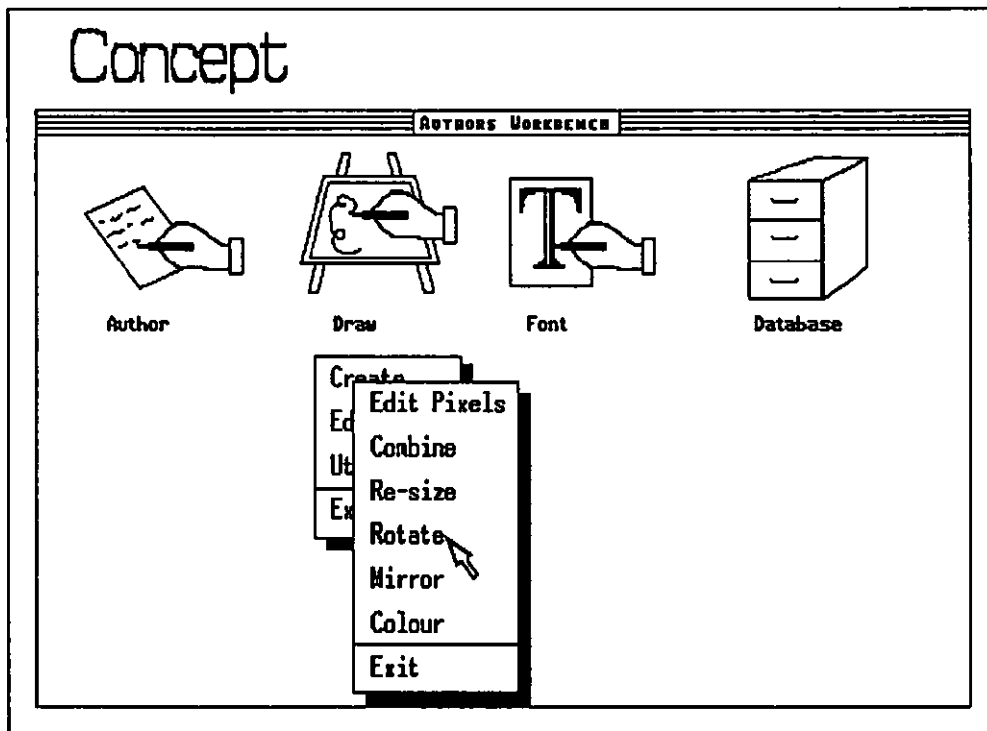


Figure 4.6

Draw. (Figure 4.6)

When 'draw' is selected, the user may choose to:

- create a graphic image.
- edit or work on an existing image.
- operate on an existing image using a set of utilities, as the second overlapping menu shows.

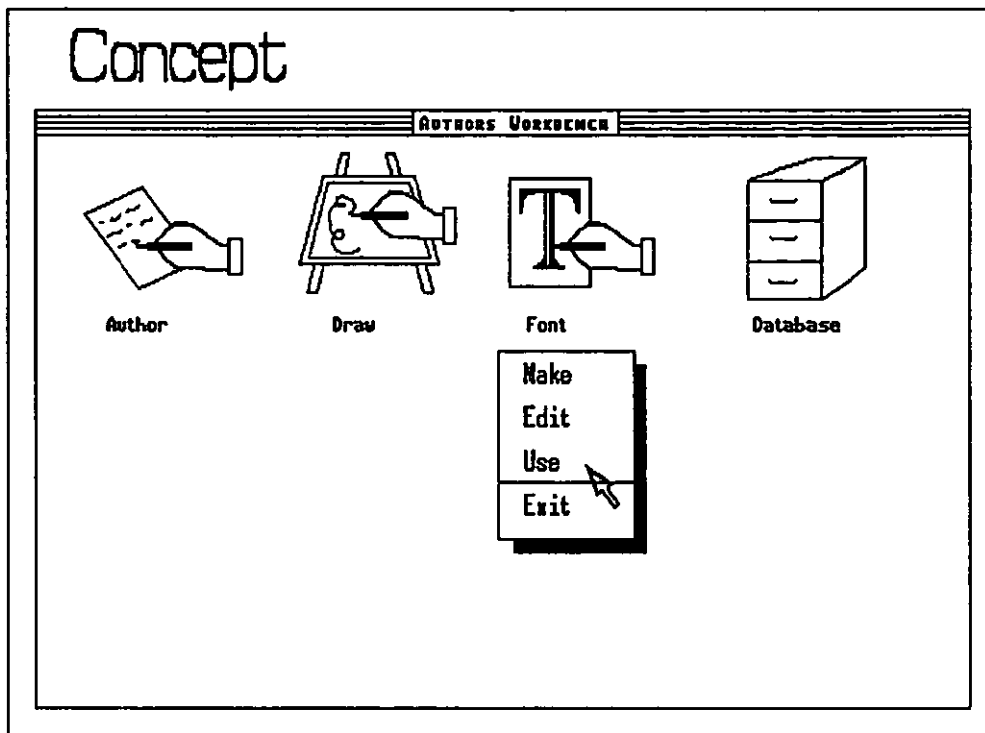


Figure 4.7

Font. (Figure 4.7)

Selecting "font" allows the author to:

- create a new style of text font.
- edit an existing text font.
- build a string of text using a specified font.

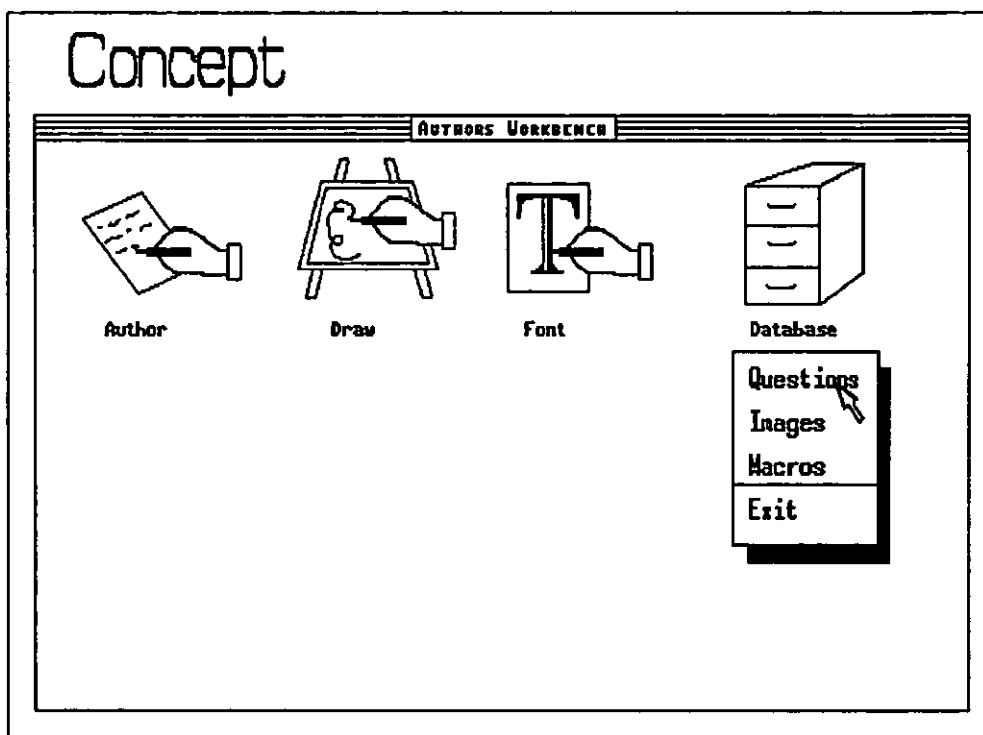


Figure 4.8

Database. (Figure 4.8)

By choosing "database" the user can store or retrieve:

- questions from a question bank.
- images from an image library.
- macros (program code combinations).

CONCEPT: Hardware Considerations.

To implement the CONCEPT interface on an IBM PC/AT or PS/2 compatible computer, authoring stations must be equipped with:

1. An EGA or VGA standard monitor and graphics card in order to display the high resolution graphics based icons used by CONCEPT.
2. A mouse, to operate the point-and-select dialogue style of CONCEPT.

The first requirement is not unreasonable: Most new computers are now supplied with VGA graphics as the standard colour display: CGA and EGA are rapidly disappearing as configuration options. The second requirement identified is also easily accommodated. For users who do not already have a mouse, this inexpensive device could be supplied as an optional item when the system is purchased.

CONCEPT: Where Next ?

The CONCEPT interface provides an indication of a possible direction that might be taken to improve the human-computer interface of a system such as Top Class. CONCEPT offers some essential enhancements: making the authoring process more accessible to novice computer users by allowing the user to "see" the overall structure of the Top Class environment.

As discussed here, CONCEPT is far from complete. A finished system design would include a selectable on-line help facility to act as an "author's assistant" and a context sensitive error handling system. Furthermore, documentation should be recognised as an extension to the CONCEPT interface: a complete tutorial manual should be produced to accompany the reference manual.

Implementing a CONCEPT-like interface is not a simple task. Designing the interface requires special skills: a knowledge of programming, a knowledge of authoring systems, and an understanding of the psychological issues involved. However, the starting point for authoring system designers, to borrow the words of Norman (1984) is simple: "...to take the design of the interface seriously, to recognise that both programmers and psychologists must co-operate to do the task."

Courseware:

Interface between Learner and Computer.

When considering computer interaction from a learner's perspective, the relevance of HCI issues take on a different complexion. The underlying nature of interaction is the same, but the goals that drive the interaction process are different. The problem of bringing closer the gulfs of execution and evaluation remains, but the solution must take into account the learners image of the computer.

If the CAL interface is designed with care, the learner will become engaged in a learning conversation with the computer, with the learner's perception focused on the virtual machine: the physical computer will simply become a transparent intermediary in the learning process.

Dialogue Across the Interface

Most people, whether computer experts or novices, are able to recall a frustrating experience when using a computer. Perhaps the machine behaved unpredictably, or simply refused to co-operate ! Such an experience usually stems from the user not understanding what type of responses the computer requires.

These frustrating encounters should be avoided at all costs, they may at best ruin a learning session, and at worst convince the student never to touch a computer again.

With careful planning and design it should be possible to ensure that such experiences are avoided. The key to doing so is to make the dialogue between learner and computer a natural and logical interaction. In doing so the result will be that the student will have the feeling of being in control, and hopefully forget that they are actually interacting with a physical machine.

Dialogue across the interface is a two way process. Typically, this involves information being presented to the user on a Cathode Ray Tube (CRT) display system, and responses inputted from the user via a keyboard. Thus, the issues of human factors in screen design and dialogue design are each considered in terms of the interface device.

Output to the Learner.

As display technology continues to advance, many of today's computers are equipped with high resolution colour graphics visual display units (VDU's), capable of presenting text and graphics information. When designing interactive educational software, the organisation of information presentation via the VDU is an important part of the CAL authoring process. The central theme to good design is to ensure that the learning material is presented clearly, and that the information processing ability of the learner is not exceeded.

Screen Design Issues.

Work performed in the field of psychology on how information is extracted from text is of interest to CAL authors, particularly if these findings can be applied to the presentation of text on CRT displays. Hulme (1984) has performed such work, applying the cognitive psychology of reading to the problems of reading text from a computer display. His work has revealed that reading information from a CRT display is in the order of 30% slower than the reading of text from a book. This he accounts for by the fact that less information is contained on a CRT screen than on a book page (approximately 400 words for a book, compared to 150 words for a CRT screen). Also, when reading, most people prefer to adjust the angle of viewing. Although such an adjustment is easy in the case of a book, a CRT screen imposes strict limitations. Hulme goes on to point out that the luminance of a CRT display causes discomfort, and often results in fatigue, which also detracts from the reading process.

Van Nes (1986) identifies the importance of space, colour and contrast to maximise the legibility of screen based text. He states "Being easy to read, text with a high legibility enables the reader to devote his attention to understanding what ever information the text conveys." The implications for the design of educational software need no further elaboration. The recommendations made by Van Nes for text screen design can be summarised thus:

- that light background/dark foreground combinations are more legible than the reverse, although this only applies on VDU's where the screen refresh rate is above 70Hz (a critical rate for avoiding an annoying 'flicker' effect).
- the use of space on a text screen is important. Legibility can be significantly improved by using blank lines to separate paragraphs, and give screens an orderly appearance.

- colour should be used carefully. Van Nes suggests that not more than three colours should be used on a single text screen. Colour can be used to suggest grouping in text screens. Part of a block of text or diagram may be coloured differently to its surround and thus help the user to search for key information.
- different character fonts can be eye catching, and this may be usefully exploited. However caution should be exercised: some fonts are less legible than others.

To avoid designing a screen that is over burdened, or cluttered, with attention gaining mechanisms, Shneiderman (1986, p.71) suggests that screens should be designed in monochromatic form. The designer may then "judiciously" decide on the use of colour and text size to aid the learner.

Accompanying text information, graphic images may be usefully employed to aid the communication of important information in a meaningful way. Yet, even with graphics, caution should be exercised. Over use should be avoided otherwise the learner may receive more information than his/her short term memory can simultaneously process.

When presenting information in a sequential manner, some concepts can be particularly difficult to convey. Computer graphics can be appropriately employed to produce what appears to be an animated diagram. This can be particularly effective in CAL programs which model situations or processes, allowing important details to be emphasised and unimportant ones to be removed.

Keller (1987) makes an important observation about the use of graphics to help keep the students attention: "Add-on images - happy faces, exploding spaceships and the like - may catch the eye but they won't engage the mind." The CAL designer should avoid using meaningless graphical images. Keller goes on to recommend that the designer should ask himself a simple question when using an image: What does the picture accomplish ?

Screen organisation is important. A good screen design should allow the learner to readily access important information: position within the lesson, help notes and so on. Isaacs (1987) recommends that a small area of the screen, ideally along the bottom, be reserved to permanently display information on how to access help, perform navigation, quit, and so on. The required information can be presented in a pop-up window when needed. Positional information such as topic name and page number should be displayed at the top of each screen.

Input Devices.

Input devices are of special interest when designing a computer based learning system. The choice of input device will allow the designer to match the interface to the requirements and needs of the learner. Although the QWERTY keyboard has become the standard input device, many computer novices find it an awkward and unnatural device for communication. To overcome keyboard skill deficiency, CAL software should, if appropriate, be designed to restrict the amount of keyboard use. Even a simple approach, such as confining input to the use of special function keys, or easy to locate number keys can relieve the burden placed on the learner by the keyboard.

Wherever possible, interaction dialogues should minimise the learner's use of the keyboard. Invariably, the user of CAL software will possess only minimal keyboard skills. By using single keystroke actions for selection, or better still mouse/pointer or touch-screen selections, the learner is relieved of the burden of using a rather unnatural input device.

By designing CAL software that makes use of a selection dialogue, perhaps by employing menus or lists, a number of alternative input devices are available: cursor keys, mouse, trackerball, joystick, lightpen and touchscreen. Discussing these pointing devices, Albert (1982) has made comparisons in terms of speed and accuracy for a cursor positioning task. Although the trackerball was found to be the most accurate, the slowest pointing device, the touchscreen, was still eight times faster than the keyboard.

Of the available pointing devices, Criswell (1989, p.28) prefers the touchscreen because it makes physical interaction with the system so easy and natural. Furthermore, its unobstrusiveness serves to increase the transparency of the system allowing the learner to concentrate on the lesson.

Dialogue Strategies.

Interactive dialogue can easily break down. Problems in CAL often arise because the student simply does not know what information to enter, or perhaps more often than not, how to enter information into the computer.

A number of relevant points are identified in work performed by Hammond & Barnard (1984), who have looked at the interface problem in terms of the style of dialogue used. Their work identifies some styles which should be considered in CAL dialogues:

1. A question and answer dialogue, where the computer asks questions to which the student responds. This dialogue style is not an easy one to design. Allowing the student to enter a character string from the keyboard will require the software to be able to appropriately handle every possible response that the student produces. Fortunately, most authoring systems accommodate this dialogue by providing a range of matching strategies. Typically, these may include:
 - detecting a key word in the user's input.
 - accepting both upper and lower case characters.
 - using a phonetic matching algorithm to avoid a student for poor typing.
2. A menu selection dialogue, where the student selects a response from a menu of pre-determined responses, usually requiring a single number or letter entry. This offers the advantage of easy keyboard use for the student who lacks keyboard skills, hence facilitating a smoother dialogue. When used for testing, choices must be designed to discourage guessing, if the results of testing are to be meaningful.
3. In some CAL programs, such as simulations, the user may need to give directives to the computer, in which case some kind of "command language" may be used. These commands should be chosen that they logically represent their actions.

Essentially, the dialogue style, or mix of styles should be chosen according to the aims of the CAL package, the level of interaction required and the ability of the learner.

Shneiderman (1986, p.323) warns of the dangers of writing software that attempts to give the computer a personality. He points out that the novelty of anthropomorphic software soon becomes unacceptable to the user. He advises dialogue designers to "...focus on the user and use third person singular pronouns, or avoid pronouns altogether. The following examples (from Shneiderman 1986. p.325) illustrate the point:

POOR: I will begin the lesson when you type RETURN.

BETTER: You can begin the lesson by typing RETURN.

BETTER: To begin the lesson, type RETURN.

Similarly, the wording of answer judgements and other forms of feedback to the user must be phrased to be friendly and informative yet not authoritative or too personal.

The reading ability of the student must also be considered when designing text dialogue. The language used must be clear and unambiguous. Green (1988) suggests the use of a Reading Age Test (e.g. Flesch, 1948; Fry, 1977) may be useful for highlighting difficulties, but warns that "...reading and understanding are not the same thing."

Motivation.

Courseware authors must not assume that all learners will be motivated to use a given CAL package. Indeed, deliberate strategies must be employed to induce users to become actively involved in the learning process. To increase motivation, Kearsley and Hillelsohn (1982) make the following suggestions:

- students should be provided with a record of their progress.
- a competitive element should be introduced into the learning.

- allow students to work in pairs or small groups: this will allow them to re-inforce each others learning.
- improve the enjoyment level of the courseware by using animated graphics, simulations, games or humour.
- reduce boredom by designing courseware that will allow the student to explore the learning material and proceed at a rate that they choose.

Documentation.

CAL software should always be accompanied with documentation that supports the user in learning to use a given package. To this end, well produced teacher and student manuals play an important role. Identifying the problems of the new user, Clement (1984) calls for manuals that are designed to meet the user's needs.

The widespread dissatisfaction with software documentation is noted by Sullivan and Chapanis (1983). They offer the following general suggestions for those who are involved in the preparation of user guides and help texts:

- use simple, familiar language;
- use short, active positive sentences;
- make the order of events in a sentence parallel the order of actions that the user will need to take;
- be complete and specific when describing actions;
- use headings and sub-headings to identify sections in a manual;
- use lists rather than long passages in order to present one idea at a time;
- make use of space so that information can be easily scanned.

Sherwood (1987) stresses the importance of presentational issues in improving the usability of manuals. This is a vitally important point if the user is using a manual whilst operating the computer. Careful use of page size, typography, layout and so on can produce a presentation style that allows similarity between printed page and computer screen. This will allow the user to easily shift their attention from book to screen, perhaps as they follow instructions, without losing their place, and destroying the interaction process.

The production of documentation, according to Shneiderman (1986, p.372) should begin as early as possible in the design process to allow "...adequate time for review, testing and refinement." The software author is not always the best person to produce documentation. Their familiarity may well lead to incorrect assumptions being made about the needs of the user.

Summary.

The topics considered in this chapter represent the main issues that a CAL designer/ author must consider if the software that he/she produces is to become worthwhile from a human factors point of view.

Unfortunately, the knowledge that could be applied to educational computing from the fields of psychology and human-computer interaction is not made easily available to CAL designers, the results of most research work being scattered across many different professional journals and publications. It is a mammoth task just finding this information, let alone using it ! Furthermore, most of the information that is available has not been produced as a result of research into computers & learning directly, and so requires interpretation before becoming useful.

It is clear that there is a need for more research into the fundamental human factors issues within the context of computer based learning, in order that guidelines and principles may be made more widely available.

CHAPTER 5.

Courseware Production: A Software Engineering Approach.

The production of educational software is a complex activity, often performed in an environment that is sensitive to cost and productivity. To assist in the production process, techniques employed in the field of software engineering may be of use to those involved in CAL authoring.

In order to examine the potential application of software engineering techniques to the CAL production process, an overview of the software production life cycle is presented. This is followed by a review of a number of areas of the courseware production process that may benefit from adopting a software engineering approach. The issue of software testing is a particularly neglected area of the development process, and for this reason it is discussed at some length.

The Software Life Cycle.

(The life cycle approach to software engineering covers all stages of the production process. Many variations of the basic life cycle model exist, but that shown in Figure 5.1 is representative of those that abound. It identifies the essential elements with which courseware producers are concerned.)

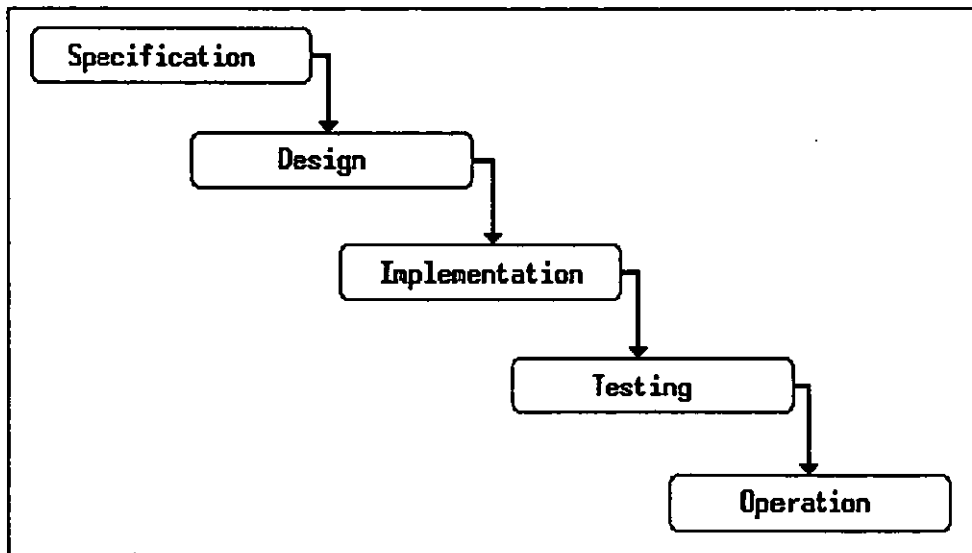


Figure 5.1

(Source: Coleman & Pratt, 1986)

The life cycle model provides a clearly defined schedule with a logical flow that describes the key stages in the production process. These are: specification; design; implementation; testing; and operation & maintenance. From a CAL author's viewpoint, the activities belonging to each stage can be considered as follows:

Specification. This stage is concerned with identification of the purpose of the software, in terms of its functions and the constraints that will affect the design process. To this end, answers to three key questions must be sought:

- What is the proposed use of the software ?
- Who will be the users of the software be ?
- What hardware is the software to be used on ?

Design. This stage is concerned with producing a design for subsequent programming. Vital to this stage is a feasibility study, which should attempt to identify a number of possible alternative outline designs which should then be compared in terms of the following parameters:

- programming method.
- educational strategy.
- production staff.
- costs involved.
- time involved.

From this information it should be possible to choose the most appropriate design for the constraints imposed by the production environment. This will ensure that energy and resources are then directed into the best design.

Implementation. This stage deals with the production of program code from the design documentation. How productively this is achieved is dependent on skill in choosing appropriate production tools, such as the authoring language, use of software tools and the utilisation of specialised hardware devices such as image scanners and digitizers, for example.

Testing. Unlike validation, which should occur at EVERY stage of the life cycle, testing involves:

- exercising the software as it would be used in the real situation.
- detecting and correcting errors generated during design and implementation.
- measuring performance.

Operation & Maintenance. It is at this stage of the production process that the software is handed over for use in the classroom or training situation. Coleman & Pratt (1987) point out that it is at this stage that a piece of software enters its most expensive stage in the life cycle, i.e. maintenance.

Although the life cycle model appears to promote a linear approach to the design/development process, it must be recognised that the model is also concerned with managing the problems that can occur during the design process. Typically, problems may arise due to: incorrect specifications, the need to add new facilities or simply accommodating uncertainty.

Thus, the life cycle model should accommodate the ability to branch back to a preceding stage, so that the design may be modified or refined in response to whatever problems arise. To reflect this, the model is shown in a revised form in Figure 5.2.

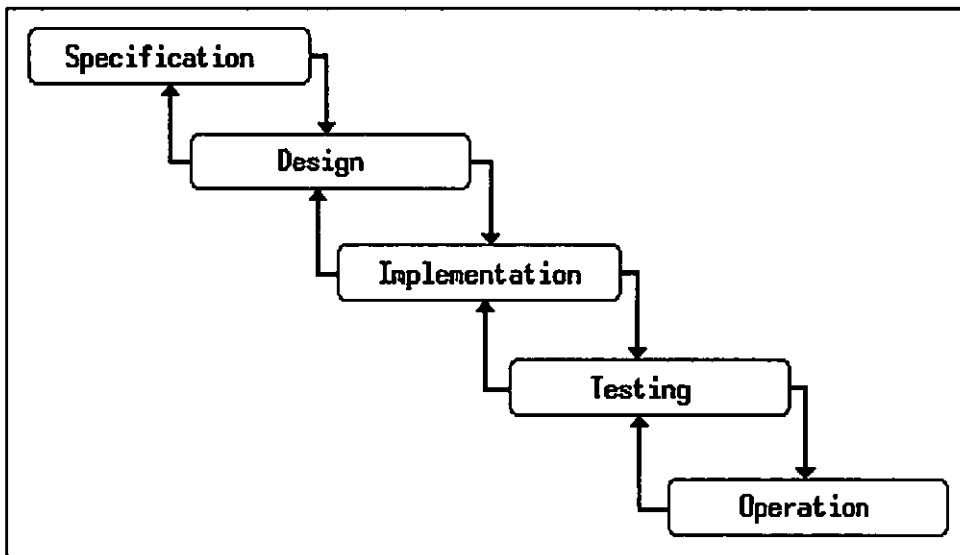


Figure 5.2

It is informative to compare the life cycle model with a list of "Stages in CBI Design" put forward by Criswell, (Criswell 1989, p.51), The list is reproduced below.

Steps in CBI Design, Production and Testing.
<p>Step 1. Conduct environmental analysis</p> <ul style="list-style-type: none"> - Proposed use of courseware - Available hardware - User attitudes <p>Step 2. Conduct knowledge engineering</p> <ul style="list-style-type: none"> - Course content - Concept/task analysis <p>Step 3. Establish instructional goals and objectives</p> <ul style="list-style-type: none"> - Instructional goals - Specific objectives and student performance levels - Instructional objective taxonomies <p>Step 4. Sequence topics and tasks in CBI lessons</p> <p>Step 5. Write courseware</p> <ul style="list-style-type: none"> - Introductions, interactions, remedial sequences review and tests - Tailor interactions for specific student performance levels <p>Step 6. Design each frame, the student computer dialogue, and the student performance record</p> <p>Step 7. Program the computer</p> <ul style="list-style-type: none"> - Programming languages - Authoring languages and packages <p>Step 8. Produce accompanying documents</p> <p>Step 9. Evaluate and revise the CBI</p> <p>Step 10. Implement and follow-up as necessary</p>

(Source:Criswell 1989.)

Both Criswell's list and the software life cycle model demonstrate the importance of completing the design process before coding, or programming, commences. Criswell's list also provides an informative elaboration of the activities occurring when educational software is authored, albeit that her list is geared to the production of *instructional* material. Unlike Criswell's list, however, the life cycle model is more general in nature, and hence more suited to guiding the production of CAL software belonging to any educational paradigm.

The Production Team.

In the commercial CAL software production environment, there will usually be several specialist staff working together as a team to produce a courseware package. Such a team may well be involved in working on more than one project, and have a well defined production schedule and target deadlines. Kearsley (1986), in discussing the team approach to courseware production, lists the production team as being composed thus:

- an Instructional Designer.
- a Subject Matter Expert.
- a Programmer.
- a Graphics Designer.
- a Script Writer.
- an Editor.
- a Project Manager.
- an Evaluation Specialist.

A team approach to courseware production is also advocated by Bunderson (1974, p.478) who identifies a point that Kearsley fails to note: i.e. that the roles identified in the courseware production team do not necessarily have to be undertaken by separate people (for example, the subject matter expert could also act as the evaluation specialist).

With the increasing availability of advanced authoring systems, the team approach to courseware production is under attack. Typically, a modern authoring system (such as TopClass, for example) contains an automatic program generation facility, a graphics editor, de-bugging tools and easy access to hardware resources such as video-disk, touch screen and digitised speech. Given such a production environment, it is often the case

that the roles of subject matter expert, instructional designer, script writer, programmer and even graphic artist are undertaken by an individual who is tempted into acting as a one man band, so to speak.

Indeed, it is in such a *modus operandi* that most CAL material is produced in the educational environment. Usually, by an enthusiastic teacher working as a lone author, working during spare time periods and during evenings, without a development schedule or target deadline.

Considering the CAL production team, Whiting (1989, p.187), warns that "The team approach to CBT authoring often mitigates against more individual designs..." Although this may be true, it should also be recognised that there are few authors who possess all of the skills and experience needed to work as a lone author and hence are able to produce top quality courseware.

By being aware of the roles to be performed, those who produce CAL software in the educational environment should be encouraged to work with others who share an interest in CAL: sharing work between members of a team, perhaps identifying those who have a flair for graphic design or programming, and enlisting their help.

Documentation.

Accurate and clear documentation is needed in order to provide efficient communication between those working on a CAL project. Documentation, if properly produced, will also provide a record of the progress of a project.

Design Documentation.

To provide the detailed information needed to facilitate production, Dean and Whitlock (1984, p.160) emphasise the value of a "program flowchart" as a graphical documentation method, particularly if one is designing instructional software that employs a branching strategy, and the "screen layout form" which allows a story-board of the appearance of a program to be constructed. This flowchart/story-board combination seems to be widely favoured (e.g. Kearsley 1986, Chambers and Sprecher 1983) as a means of documenting a program for design purposes.

Graphical Documentation.

The field of software engineering offers an alternative to the flowchart as a means of providing a graphical representation of a program and its structure. Bell, Morrey and Pugh (1987, p.69) present a case for using the structure diagram for this purpose. They identify the following attributes of structure diagrams:

- they support a top-down approach to design.
- they provide a clear, graphical representation of program structure.
- the method is well defined and therefore will produce consistent results when used by different programmers.
- the method produces a design that is independent of programming (or authoring) language.

Bell et al. go on to point out that "What a program is to do, its specification, is completely defined by the nature of its input and output of data." Clearly, since an interactive CAL program is heavily concerned with input and output, the structure diagram is suitable for representing such software. It must therefore be of interest to those involved with courseware production as an alternative to the flow-chart.

Screen Design Forms.

The screen design form is an attempt at standardising the approach to designing the appearance of the program on a screen by screen basis. Typically, the screen design form mimics the addressable screen area, onto which the screen appearance may be sketched by hand. A resulting sequence of screen design forms provides a 'story-board' of the lesson to be programmed.

Generally, screen design forms are more suited to designing text based screens rather than for those employing graphics. An example screen design form is reproduced in Figure 5.3 (Courtesy of Format P.C. Ltd.).

Top ClassTM

Screen Layout Sheet

Programme _____	Sheet No. _____
-----------------	-----------------

10 20 30 40 50 60 70
Designed by _____

5
Date _____

10
Notes _____

15

20




Figure 5.3
(Reproduced with permission)

Working solely with a screen design form, it is difficult to convey the designer's view of a finished screen particularly if graphics, colour and different text fonts are used. Furthermore, it is difficult to convey the dynamic nature of an interactive program using pen and paper. To overcome this problem designers must consider the use of prototyping, making a mock up of sample screens, to truly appreciate the implications of their design decisions.

Coding Conventions.

Using a standardised coding convention is a technique associated with structured programming methodology. The purpose of using an agreed convention is to facilitate ease of maintenance on large programs. Assuming that control over code structure can be exercised (which excludes the use of authoring *systems*), then such conventions may be of interest to courseware producers.

The actual details of the coding convention used may vary, depending on the constraints imposed by the programming (or authoring) language used, but there are general guidelines which may be used to indicate the properties of coding conventions. Schneider and Bruell (1981) discuss the main elements in some detail. These are presented below:

- code should be modularised, if possible to less than a page.
- use comments to explain the purpose of each module, its data structures and other modules accessed.
- avoid over reliance on global variables: declare variables in the module that they are used.
- use clear mnemonic names for variables, data structures and the like.

- make use of indentation and space to visually indicate the structure of a program.
- use comments to explain the operation of statements that have a complex or unclear purpose.

At the time of coding a program, abiding by some of the rules imposed may appear unnecessary or superficial to some programmers. Yet not to do so may well render the program listing difficult or even impossible to read in, say, six months time when another programmer (or even the original programmer) attempts maintenance on that program. The aim of using a coding convention, then, is simple: to produce clear, self-documenting code that may be readily understood by others.

Software Testing.

Testing is that part of a project where the developers prove the design, or more accurately, discover what is wrong with a design. Hetzel (1985, pp. 5-8) defines software testing as "testing against pre-defined ideals, and embraces a number of activities." Basically, these activities include:

- Checking the program against specifications.
- Determining user acceptability.
- Ensuring that a system is suitable for use.
- Understanding the limits of performance.
- Learning what a system is not able to do.
- Verifying documentation.
- Finding errors.

Having identified these activities Hetzel then makes the statement that "testing is the measurement of software quality."

The above certainly holds true for educational software, but what is meant by quality? Essentially, quality software is that which meets all the requirements placed upon it. So, by defining these requirements, it should be possible to identify the parameters to be examined when testing educational software.

The relevant factors that affect quality are as diverse as the range of tasks for which computers are used. Fortunately, it is possible to identify important features which are common to all educational programs, these are:

- Functional Quality.
- Adaptability.
- Production Quality.

Each of these points will now be considered in turn.

Functional Quality

This is concerned with an external, or user's, view of the software, and is to some extent a subjective area. Functional quality can be considered in terms of both technical and educational factors.

Technical considerations can be treated objectively, with the following factors used to provide the basis for good educational software.

First Impressions: When using a package, particularly for the first time, impressions are formed which last. Hence, good software should be simple and straightforward to load. Furthermore, once loaded, instructions should be given on screen for starting and using the software. To allow for experienced users, a facility to skip over this preliminary information would be useful.

Program Input: A well designed program will make use of the computer's ability to be interactive, and so the user's perception of program input is important. To this end, the program should accept input in a clear, unambiguous manner and allow the user to edit input before it is accepted for use. Simple yes/no type responses could be aided by providing default settings. The program should also be able to deal with incorrect input appropriately, i.e. it should be robust, to the extent that if wrong input persists help instructions are given.

Screen Output: How information is presented to the user also affects the user's acceptance of the software. Text should be presented in a clear and readable style. Large amounts of text on a single screen should be avoided, and use made of highlighting and colours to emphasise particular areas. Unrelated information should not appear on the same screen. Graphics, if available, should be exploited to allow the program to turn static diagrams into moving pictures.

User Friendly: The manner in which the program interacts with the user is worthy of particular evaluation. A friendly, helpful manner will inspire user confidence. Help information should be easily accessible at any point in the program, and should be relevant to the user's needs. A pause facility would enable the user to control the rate of progress through the program, and promote a feeling of being in control.

Educational Parameters

Because of the scope for individual interpretation it is not easy to state good features for educational software. However, there are fundamental qualities that are the key to worthwhile software:

- The content of the software must be accurate and unambiguous. Errors of this type will destroy the users confidence.
- The package must provide an attainable educational goal in order for the user to feel that progress can be made. Such goals must be observable, in order that they can be recognised and measured.
- The user must feel motivated to use the package, so goals must be seen to be worthwhile.

Adaptability

If fully exploited the computer is one of the most powerful teaching tools available, offering interaction with the user and having the ability to branch to various parts of a program as dictated by the user's responses. Good software should make use of this ability to adapt to the user.

Production Quality

This term is used to convey an 'internal' view of the software, and is concerned with how well the package has been considered and put together. Key factors in this sphere include the following:-

- a) The provision of well produced documentation, in the form of:
 - i) a teacher's manual, that gives a full description of how to use the program and its intended place in the curriculum.
 - ii) a student manual, that describes how to start the program, its purpose and provides worksheets if appropriate. It is particularly important that the student manual is written in an encouraging and motivating way.

- b) Well written, structured software that takes advantage of techniques such as windows, menus, icons and pointers.
- c) Appropriate choice of hardware facilities, in using function keys, mouse etc. to facilitate ease of user interaction.
- d) The system should, if possible, be portable to run on the range of machines that are commonly available in education.

Performing Evaluation.

There are some important aspects of software testing methodology, identified by Hetzel (1985, pp.19-27), which have an important bearing on evaluating educational software. These are:-

"Complete testing is not possible." Any computer system, by its nature is complex and so the testing of every eventuality is unrealistic in terms of time and practicability.

"Testing is difficult." This statement is based on the fact that you need to fully understand the system you are testing, in order to test it effectively. As the system is complex, testing (and hence evaluating) is difficult.

"Testing must be planned." The testing process must be structured in order to be effective. A plan will organise how testing (or evaluation) will be carried out, what is to be tested, and provide the results. An ad hoc approach to evaluation will result in ineffective use of time and misunderstood results.

"Testing requires independence." To obtain reliable "unbiased" results, the evaluator must be an unbiased person, whose main objective is to make accurate measurements.

The points made in these four statements have particular significance when considering evaluating educational software, not only in terms of how results are obtained but also in terms of who should undertake the task. As Potosnak (1988) points out, when testing software, users "...should be representative of the intended population." When testing educational software, this means using real students and teachers.

Summary.

The life cycle approach offers authors a structured yet flexible methodology for designing and developing CAL software. Unlike the schemas put forward elsewhere (e.g. Criswell 1989, Chambers and Sprecher 1983, Steinberg 1984) which appear to favour the production of instructional software, the general nature of the life cycle approach allows it to be applied to the production of any software, regardless of educational paradigm.

By using the life cycle approach, or at least an interpretation of it, courseware production should become more productive and directed towards a defined outcome. The resulting software should, in turn, be more reliable and hence require less maintenance.

Many of the parameters discussed concerning testing are, with interpretation, suitable for forming the basis

of an evaluation instrument which may be used by those who are involved with the selection and use of CAL software.

For even the modest sized project, software engineering techniques offer benefits for those concerned with CAL project management and implementation. As Sheppard (1983) points out "Most software engineering techniques are practical and easy to use; in fact, good programmers have typically discovered many of them through experience."

Chapter 6.

Conclusion: A Look to the Future.

Attempting to look towards the future and identify the issues and developments that will shape the working environment of the CAL author is not easy. Technological change is taking place at a staggering rate, and in unpredictable directions, with research effort being directed by the whim of political and commercial decision makers.

Therefore, approach taken here is to avoid pure speculation, and to identify likely developments, based on the knowledge of the present state and current research work. To this end, relevant developments in computer hardware, software and the educational computing environment are discussed.

Hardware Developments.

Computer Hardware.

Computer systems have, over a period of some forty years, undergone many changes. Obvious milestones were the replacement of the thermionic valve with the transistor, and the replacement of the transistor with the integrated circuit. It is the refinement of micro-electronic 'chip' technology that has revolutionised the cost of computers through mass production. Computer performance has increased as rapidly as size has decreased, to produce the desk top computer that can out perform its mainframe counterpart of only ten years ago.

For the sake of this discussion, computer hardware is considered as comprising of two main areas: computer systems and peripherals.

Computer Systems

Using the parameter of computer power, measured in Millions of Instructions per Second, Peled (1987), makes the following predictions (see Figure 6.1), concerning four classes of computer:

- At any point in time, the power of mainframe computers is greatest.
- There will be a tendency for some of the most powerful personal computers to equal, or even exceed the power of some of the mini-computers available.
- Embedded computers (sometimes called 'dedicated computers') rise at a rate approximately the same as that for the other categories of computer.

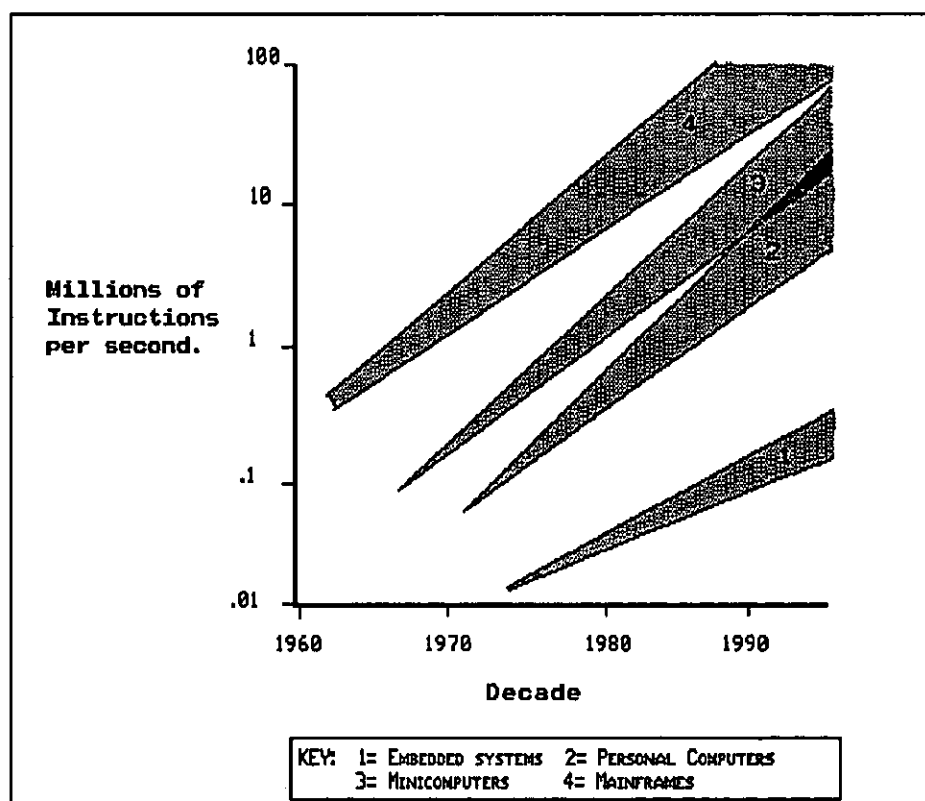


Figure 6.1. (Source:Peled,1987).

For educational computer users, the second point is the most relevant. Evidence of the availability of powerful desk top computers (often referred to as 'workstations') that equal the power of many mini computers can be found by browsing through the advertisements in many of today's computer magazines. Often costing less than £3000, these new workstations are already appearing in a number of well funded educational establishments. As the ratio of computer power to cost steadily improves the spread of these machines into all sectors of education seems inevitable, beginning with colleges and universities and later into schools also.

In the next five years the computers that will be available to education will probably have 5 to 10 Megabytes of memory as a standard configuration and will be driven by not one microprocessor but an array of parallel processors integrated into a device called a transputer.

Yet, at the present time, while many colleges and universities enjoy using 'state of the art' technology, a great many schools are still making do with the computers that are technically obsolete. For the many schools that are faced with financial difficulty, the BBC micro-computer will continue to dominate classroom computing. With no apparent successor having the same extent of software support, it is understandable that this should be the case.

So, taking a realistic view, there will probably be three types of computer in use in education for the foreseeable future:

1. Micro-computers (like the R.M. Nimbus, for example) with a 16 bit microprocessor, 640 K bytes of main memory and a floppy disk. Such machines will become the standard computer for many primary and secondary schools.

2. Personal workstations, typically using a 32 bit microprocessor, or even in some cases a transputer, with a 60 M byte hard disk, floppy disk and probably 5 M bytes of main memory. Such machines will be found in most colleges and universities, and in smaller numbers in well funded schools.
3. Mainframe 'super-computers' will mainly be used by institutions undertaking research work.

Peripheral Devices.

Computer Displays

Modern computer displays are based on an ageing cathode ray tube (C.R.T.) technology, with the picture produced by scanning an electron beam across a matrix of phosphor dots in a similar way to the domestic television system.

A successor to the C.R.T. based display system is emerging. Liquid crystal and gas-plasma technology is providing a more compact and more efficient display method for computer generated text and graphics and is now used on many lap-top and portable computers. As yet, this technology does not support colour display or large screen sizes, but it is expected that in the foreseeable future these problems will be overcome.

Storage Devices

Future computers are not likely to abandon the magnetic storage techniques that are presently the mainstay of computer backing store. Magnetic technology has been refined and developed to offer a cheap, reliable way of storing small to medium volumes (up to 60 Mega-bytes) of data quickly and conveniently on the types of computer currently found in education. This position of supremacy is however, under threat from a newcomer.

The presence of a large market for audio compact disks has produced vigorous development work in the use of optical devices as a storage medium, yielding CD-ROM (Compact Disk-Read Only Memory). Being a read only device, CD-ROM suffers from the limitation that data, and hence information, must be recorded onto it at the time of manufacture. CD-ROM's greatest attraction is that a massive 600 M bytes of data can be stored on a single disk. Hardware is currently available to allow IBM PC and PS/2 machines to read CD-ROM.

As yet however, CD-ROM technology is not widely used, although examples of its application are increasing. Hampshire (1989) reports that over 2000 different CD-ROM titles are currently available, with a world-wide user base of around 70,000. He points out that the majority of the available disks hold special purpose information, e.g. the entire parts catalogue for Nissan Motor dealers. Very few CD-ROM titles appeal to general or educational users but there are notable exceptions: the complete Oxford English Dictionary is now available on CD-ROM from Oxford University Press and the software giant, "Microsoft," is currently releasing its entire programmer's reference and a small business resource disk on CD-ROM.

With prices currently at £600 for an internal CD drive and interface card, the technology is affordable to education. As more and more general information is made available on CD-ROM it will provide a valuable information resource: the availability of census data, public records and so on will provide teachers with an important information tool in many areas of the curriculum in much the same way as conventional databases are used in schools today.

Development in the field of optical disk technology is currently pursuing the goal of producing optical disks that offer read and write capability. When this goal has been achieved, then optical storage may well displace magnetic methods as the mainstay of computer backing store. Until this happens, computers will, in the near future, have three classes of memory: main memory (employing microelectronics), backing store (using traditional magnetic media) and a new 'mass store' using optical disk.

Input Devices and Computer Interaction.

Researchers are increasingly directing their efforts towards the problems of designing systems that are easy to use by computer experts and novices alike. Hence, as many computer users in education are not necessarily computer experts, developments in the field interaction device technology are of particular interest.

Pointing Devices

The mouse, used as a pointing device, has become an important communication mechanism for computer users. In fact for many selection type tasks the mouse is probably used more naturally and easily than the traditional computer input method: the keyboard.

Keyboards

Keyboards present a major interaction obstacle to novice computer users. Shneiderman (1987) notes that the DVORAK layout keyboard can be learned in about one third of the time that it takes to learn to conventional QWERTY keyboard. It remains inconceivable that the population of QWERTY users would learn a new keyboard layout: standardisation appears to have penalised new users for some time to come. On the theme of textual input, Peled (1987) speculates that a typical personal computer of the

next decade may well allow the user to "write" onto a flat liquid crystal type display. The computer will then recognise the characters and translate them into commands, text or drawings.

Speech

Because speech is our (human) natural means of communication, it is highly desirable to use it as a method of computer interaction. Bailey (1984) describes how speech output can be generated the computer either by storing digitised speech on disk or in memory which can be randomly accessed for playback, or by synthesising speech using coded digital information: the Texas Instruments "Speak & Spell" toy is an example of this approach. Digitally stored natural speech, which has been possible for a number of years, has received little attention to date because of the requirement for vast amounts of memory needed to store just a few seconds of speech. However, now that microelectronic techniques have evolved to a suitable level, and very large scale integration memory devices are a reality, it is likely that this form of computer speech generation will receive renewed interest.

Speech input on the other hand, is still very much in its infancy. To get a computer to recognise even a small vocabulary of spoken English words requires vast amounts of memory and processor power. The difficulty of such a task is not fully appreciated until one considers the range of variations in speech (different vocal properties, dialect and syntax) that the computer must be equipped to deal with before speech input can be truly realised.

Despite these and other difficulties, researchers are making progress. Peled (1987) reports that a team working at IBM Research have developed a computer system that is capable of recognising 20,000 different words, provided that the speaker takes brief pauses between words.

The system that achieves this uses a program that "...comprises 60 million instructions" and uses four specialised microprocessors and an IBM PC/AT. Peled states that "...four years ago such a computer would have occupied a room: in five years it will probably take up less space than a card."

Simply in terms of making human computer interaction a more natural process, many educational technologists eagerly await the arrival of speech input and output.

Interactive Video.

In a similar way to audio discs, optical storage is being used for video applications, allowing some 50,000 picture images to be stored on a laser disk. The combination of computer control and laser disk images is seen by many as the perfect combination for achieving successful computer based training.

Wood (1989) reports that interactive video (I.V.) is becoming "an attractive alternative to classroom based training for larger organisations." Wood sees the major advantage of I.V. over classroom and traditional computer based training as being its compactness. By setting up an I.V. station in the corner of the office, training can be delivered to those who need it at their place of work. He also suggests that I.V. offers other advantages over traditional C.B.T: its quality of image presentation allows a more realistic view of the world to be painted than is achieved using computer graphics.

O'Neill (1987) explains the advantages of I.V. in terms of its quality compared to a number of other 'traditional' learning methods. He sees I.V. technology, coupled with Computer Assisted Learning, as providing effective learning through active engagement of the learner, and higher motivation due to the learner

controlling his own rate of progress. Unfortunately, these claims have been heard many times about many educational tools (Programmed Learning and CAL to name two). It is easy to see why sceptics simply dismiss I.V. as another fad.

In direct opposition to the above views, O'Shea and Self (1983) cast doubt on the value of I.V. They claim:

"...the videodisk encourages the freezing of chunks of teaching material and a reversal to modes of teaching which have not been found effective."

Although this view is valid when considering the use of videodisk as a reception-based learning medium, it does not apply if the videodisk is employed as a library of images to be controlled and ordered by a student. Used as an image resource coupled with a hypertext type system (discussed later), I.V. has a lot to offer to those who can afford the high price. Furthermore, I.V. offers a valuable teaching method for certain vocational and skills training activities: as such it will be of interest to those involved in Further Education.

Software Developments.

As computer memory capacity increases and processor units become faster, the software used on microprocessor based machines will become increasingly more sophisticated: programmers, and authors, will gradually be freed of the constraints of working with inefficient hardware.

The face of computer programming in general is changing beyond recognition. Powerful so-called 'CASE' (Computer Aided Software Engineering) environments are

becoming increasingly available to enable commercial software producers to automate virtually every step of the software life cycle (BYTE Editorial Staff, 1989). Armed with these software tools, Jones (1987) believes that programmers will become more productive and turn out more reliable code.

Authoring Software.

It would appear that Barker (1989) expects the trends emerging in commercial computing to be reflected in the next generation of "authoring environment." He identifies key elements of such an environment as comprising facilities for: window management, interface device handling; database utilities; and code generation. Interestingly, Barker also identifies a need for the authoring environment to be able to represent knowledge based structures.

In the light of the educational software production problem identified by Nicholson and Scott (1986), perhaps the authoring environments of the future will be able to assist in combating the educational software shortage without imposing a sacrifice of quality.

Hypertext Software.

Interest is also emerging in an entirely new form of software that offers great potential for educational computing: hypertext systems. These systems are not easy to define, for they do not fit into any existing educational software paradigm. They are quite unique. Put in simple terms, they allow non-linearly arranged items of information to be linked by an author and then browsed by the learner. It is this freedom to "travel through" an information resource that sets hypertext apart from other educational software. Known producers of hypertext-like systems are: Apple (HyperCard); I.B.M. (LinkWay); and Xerox (NoteCards).

The educational use of hypertext systems is still in its infancy. Nevertheless, concern is already being voiced that hypertext systems will challenge our understanding of the already weak areas of cognitive theory in learning and the internalisation and organisation of knowledge (Fischer and Weyer 1988). Even so, research into the use of hypertext type systems promises to yield fresh insights into the organisation and delivery of learning material, if not the nature of learning itself.

Artificial Intelligence.

An emerging area of research within the field of educational computing is that of intelligent computer assisted instruction (ICAI). The characteristics of ICAI differ considerably from the traditional form of CAI.

Traditional CAI is structured procedurally and has no representation of the relationships that exist between the elements of knowledge held. ICAI software however, uses artificial intelligence principles and thus educational content is based on the definition of subject knowledge and the representation of relationships that exist between elements. The resultant knowledge structure is assessed by some type of inference engine.

By its very nature, ICAI appears to offer the means to break the stranglehold of behavioural forms of CAL. ICAI has the ability to replace the question-answer approach to learning with a process that engages the student in a learning conversation, where the computer generates a problem and then guides the student through a problem solving process to reach a solution. Using artificial intelligence techniques, the computer is able to tailor responses to meet the individual needs of the student: the pre-defined feedback offered by traditional CAI cannot.

The essence of ICAI systems, may best be summarised by borrowing the words of Park, Perez and Seidel (1987 p.15), who describe that ICAI attempts to produce a learning system that "...allows both the student and the system a flexibility in the learning environment that closely resembles what actually occurs when student and teacher sit down one-to-one and attempt to teach and learn together."

There are many examples of prototypic ICAI systems to be found. An informative summary of these is presented by Park, Perez and Seidel (1987, pp.22-23). Examples worthy of note are: A self improving quadratic tutor (O'Shea, 1979); SOPHIE, an attempt at creating a "reactive learning environment" (Brown, Burton and Bell,1975); WEST, an arithmetic tutor in a game environment (Burton and Brown, 1979); GUIDON, a program for teaching medical diagnostics (Clancy 1979).

Ridgeway (1988) is critical of ICAI systems, and warns that would-be implementors must guard against the following pitfalls:

- ICAI may well result in the substitution of machine experts for teachers.
- ICAI could devalue human-human communication, with the emphasis of one learner to a computer reducing the perceived value of group work.
- links between artificial intelligence and cognitive psychology may produce a devaluation of creative activities such as music composition or painting.

Clearly, although there is cause for excitement concerning the developments that are taking place in applying artificial intelligence techniques to the production of educational software, the resulting ICAI must not become regarded as a panacea for the present ills of CAL.

Developments Within Education.

Probably the most important organisational issue concerning authors of educational software is that of the curriculum, if their output is to be used in the classroom. In order to foresee what forms of curriculum development might occur in the future, it is appropriate at this point to consider three possible developments, as described by Dunn and Morgan (1987).

Firstly, Dunn and Morgan suggest "...a small-scale non revolutionary change within an accepted curricular tradition or structure." To some extent, this is already evident: computers are being used by many teachers to supplement their existing teaching. Unfortunately, much of the Computer Assisted Learning (CAL) software currently available is quite limited. CAL is seen by many teachers merely as computerised drill and practice. There is, therefore, a danger that if computer based learning continues to be used in its present form, as a supplement to existing methods of teaching, the curriculum of the future will become more prescriptive, discouraging the development of thinking skills, originality and innovation, and encouraging conformity and uniformity.

The second possible change that Dunn and Morgan identify is concerned with "...the breaking down to some extent of the academic subject based structures." For such a change to take place, educators must recognise that there is a significant degree of overlap between many subject areas and that the compartmentalisation of bodies of knowledge (Mathematics, Geography or Science, for example) is artificial and destructive to the long term needs of the learner. I share Dunn and Morgan's hope that new technology will be the catalyst that allows the subject boundaries to be re-defined.

Their third change suggests a "...radical evaluation of new curricular forms." The basis for such a suggestion lies in the explosive potential of the new technology itself. Dunn and Morgan contend that the, as yet, unseen dimensions of information technology will bring new forms of knowledge and suggest that "...the power to access and manage data of enormous complexity will allow the development of new combinations of information, skills and mental processes." This may seem to be a romantic view at the present time, but given the explosive rate at which change is taking place in technology, we should not be so eager to dismiss such an opinion.

Uses of Computing and Technology in Schools.

Having considered the broader issue of the curriculum, it is appropriate to turn and look at the educational role of the computer in the school and college setting.

The software produced to date has not made full use of the computer's ability to interact with the learner. A recent comment on the current state of software is made by Nicholson and Scott (1986), who believe that the use of the computer as an every day teaching tool is threatened by a lack of high quality software. It is a well documented complaint (e.g. Hawkrige 1983 p.87, Maddison 1983 p.7).

A possible explanation for this sad state of affairs is that educational software, on a large scale, is not seen as profitable and hence worthwhile by commercial producers. This point is supported by Nicholson and Scott (1986):

"A commercial organisation can only indulge in such a costly exercise if sales remain at high volume over a long period. Unfortunately sales

are low - a school will buy only one copy of a program compared with ten or more of a textbook- and life expectancy is short, given the pace of hardware development. Few educational publishers are now willing to risk CAL publication."

As a result, much of our present software is being written by enthusiastic amateurs. A solution is offered by Nicholson (1987), who in his paper, "A Short Term Plan for CAL," offers a "...survival scheme" for remedying the current software problems within the next 5 years (taking us up to 1992 !). Essentially, his plan calls for:-

- making the most of current hardware and resources.
- planning actively for the replacement of hardware and software after around 1991.

What Nicholson fails to indicate however, is who should be implementing his plan. It must be assumed that his hopes rest with the teacher. For such a plan to succeed, it is important that work that was funded by the MEP, i.e. implanting skills and knowledge into our schools, is not only sustained, but boosted. The future of CAL is ultimately in the hands of the teacher.

Returning to the theme of artificial intelligence, Neuman (1987) offers an alternative, and refreshing perspective. He argues that by developing computer based artificial intelligence systems, a greater understanding of the processes involved in thinking will be acquired. He argues that by understanding how students learn we will find the means to " re-vitalise the curriculum." In line with Dunn and Morgan's third suggested change in the curriculum, I believe that it is this type of approach that researchers in education should adopt to bring learning and teaching into a new era.

COMPUTER-AIDED INAFILL
IN COST-EFFECTIVENESS ANALYSIS

By RAN BANDA A.M. 1986

3, 4, 7, 8, 9, 15, 22, 23

24, 25, 26, 29, 32-37, 42, 44

An Agenda for the Future.

In the light of the above developments, education must review its objectives and methodologies in order to serve the changing society to which it belongs. My aim, in providing this summary is not to offer predictions, but to point towards a pathway that educators must take. The points listed below are, therefore, an agenda for the future.

Preparing for Change:

Accepting that the curriculum will inevitably change as a result of computers being adopted as a cross-discipline tool, greater levels of co-operation and exchange of ideas are needed between members of staff within schools and colleges. This will assist in preparing for the inevitable change in the role of the teacher, from dispenser of knowledge to that of learning guide.

Teacher Training:

Ultimately, it is the teacher in the classroom who has the task of implementing, or in some cases *producing* computer assisted learning material. Hence the training, and re-training of teachers to fulfil this role is of vital importance to ensure that this objective is realised. It is important to recognise that many serving teachers have had little or no experience in the use of computers or new technology. To raise levels of awareness and competency, LEA's must act quickly, to provide suitable training, not only as a short term remedy, but as an on-going commitment to ensure that staff remain *au-fait* with the rapidly changing technological environment that they are a part of.

Concerning Educational Software:

Presently, there is no money to be made in producing educational software on a commercial footing. The way forward, until educational courseware does become viable is for Local authorities to set up courseware development teams to work with teachers to produce quality not quantity material.

Resources and Funding:

The educational establishment will come under increasing pressure not to be outpaced by technology. Computers, with technology advancing at its present rate, should be replaced after four years. Finding the money to do so will be a major problem. With government pressure and 'incentives' to encourage schools to opt out of local authority control, those schools who cannot raise sufficient funds will lag behind, eventually being unable to offer the high-tech environment that many students and their parents will come to expect. National schemes to support schools must continue to ensure that this does not happen. The experiences gained from the Department of Industry and MEP schemes must be used to get it right next time.

Policy Making:

In order to ensure the formulation of effective policies for the future, there is a need to provide a forum where those who have an interest in the future of education can meet and exchange ideas. It is anticipated that those who should be involved in such a forum, or perhaps, network of forums are:

- representatives of local government.
- education officers.
- representatives of examining bodies.
- members of the teaching profession.
- representatives from industry and commerce.
- representatives of student bodies.

Computers are bound to have profound and as yet unforeseen effects on how learning takes place in schools and colleges in the future. Therefore, we must go out and meet the future before it meets us, we must plan for change, not wait for it to happen...

Bibliography.

- Albert, A.E. (1982) "The Effects of Graphic Input Devices on Performance in a Cursor Positioning Task", in R.G. Edwards (ed.) *Proceedings of the Human Factors Society 26th. Annual Meeting*, The Human Factors Society: Santa Monica.
- Alpert, D. (1975) "The PLATO IV System in Use: A Progress Report" in O. Lecarme and R. Lewis (eds.) *Computers in Education*, IFIP: North-Holland.
- Armstrong, P.K. (1987) *CAL Modes: Classifications*, Paper given as part of the M.Sc. in Computer Education: Loughborough University (unpublished).
- Ausubel, D.P. (1963) "Reception Learning and the Rote-Meaningful Dimension" in E. Stones (ed.) *Readings in Educational Psychology*, Methuen & Co. : London.
- Bailey, P. (1984) "Speech Communication: The Problem and Some Solutions" in A. Monk (ed.) *Fundamentals of Human Computer Interaction*, Academic Press: London.
- Barker, P. (1989) "Authoring for DELTA", *Education and Training Technology International*, Vol. 26 No. 3 pp. 175-185.
- Bell, M. (1985) "The Coventry Computer Based Learning Project", *Programmed Learning and Educational Technology*, Vol. 22 No. 3 pp. 218-223. Serial 370
- Bell, D., Morrey, I. and Pugh, J. (1987) *Software Engineering: A Programming Approach*, Prentice-Hall: London. 001.6425

- Bitzer, D.L., Braunfield, P.G. and Lichtenberger, W.W.
(1962) "PLATO II: A Multiple Student, Computer Controlled Automatic Teaching Device" in J.E. Coulson (ed.) *Programmed Learning and Computer Based Instruction*, Wiley:New York. 371.3944
- Bosser, T. (1987) *Learning in Man-Computer Interaction*, Springer-Verlag:Berlin.
- Bork A. (1984) "Producing Computer Based Learning Material at the Educational Technology Center", *The Journal of Computer Based Education*, Vol. 11, No.4. pp.78-81.
- Brown, J.S., Burton, R.R. and Bell, A.G. (1975) "SOPHIE: A Step Towards a Reactive Learning Environment", *International Journal of Man-Machine Studies*, Vol. 7 pp. 675-696.
- Bruner, J.S. (1988) "Instruction and Learning" in A.Jones & P.Scrimshaw (eds.) *Computer Education 5-13*, Open University Press:Milton Keynes. 370 Serial
- Bunderson, C.V. (1974) "The Design and Production of Learner Controlled Courseware for the TICCIT System: A Progress Report", *International Journal of Man-Machine Studies*, Vol.6 pp.479-491. Serial 001.5
- Bunderson, C.V. (1981) "Courseware" in H.F. O'Neil (ed.) *Computer Based Instruction: A State of the Art Assesement*, Academic Press:New York.
- Burton, R.R. and Brown, J.S. (1979) "An Investigation of Computer Coaching for Informal Learning Activities", *International Journal of Man-Machine Studies*, Vol. 11 pp. 5-24. Serial 001.5

BYTE Editorial Staff (1989) "Making a Case for CASE",
BYTE, Vol. 14 No. 13 pp. 154-171. Serial 001.6

Chambers, J.A. and Sprecher, J.W. (1983) *Computer Assisted Instruction: Its Use in the Classroom*, Prentice-Hall:New Jersey. 371.39445

Chandler, D. (1984) *Young Learners and the Microcomputer*, Open University Press:Milton Keynes. 371.39445

Clancy, W.J. (1979) "Tutoring Rules for a Case-Method Dialogue", *International Journal of Man-Machine Studies*, Vol. 11 pp. 25-49. Serial 001.5

Clement, D. (1984) "The Role of Documentation in Human Computer Interaction", in G. Salvendy (ed.) *Human Computer Interaction*, Elsevier: Amsterdam. X 001.64

✓ Coleman, M. & Pratt, S. (1986) *Software Engineering for Students*, Chartwell-Bratt:Sweden. 001.6425

✓ Criswell, E.L. (1989) *The Design of Computer Based Instruction*, Macmillan:New York. 371.39445

Crowder, N. (1959) "Automatic Teaching By Means of Intrinsic Programming" in E. Galanter (ed.) *Automatic Teaching: The State of the Art*, Wiley-Hill.

Dean, C. and Whitlock, Q. (1984) *A Handbook of Computer Based Training*, Kogan Page:London. 371.39445

Dunn, S. and Morgan, V. (1987) *The Impact of the Computer on Education*, Prentice-Hall:London. 370.02854

Fairweather P.G and O'Neal A.F. (1984) "The Impact of Advanced Authoring Systems on CAI Productivity" in *The Journal of Computer Based Education*, Vol. 11 No.3. pp. 90-94.

- Fischer, G. and Weyer, S.A. (1988) "A Critical Assesement of Hypertext Systems", in *Proceedings of the CHI'88 Conference: Association for Computing Machinery*.
- Flesch, R.F. (1948) "A New Readability Yardstick", *Journal of Applied Psychology*, No. 32 pp. 221-233.
- Freer, D. (1986) "PLATO Across the Curriculum: An Evaluation of a Project", *Programmed Learning and Educational Technology*, Vol. 23 No. 1 pp. 71-75. *serial 370*
- Fry, E.B. (1977) "Fry's Readability Graph: Clarifications, Validity", *Journal of Reading*, No. 21 pp. 242-252.
- Gagné, R.M. (1985) *The Conditions of Learning and Theory of Instruction*, Holt Saunders: New York.
- Goodwin, N.C. (1987) "Functionality and Useability", *Communications of the A.C.M.*, Vol. 30 No. 3 pp. 229 - 233.
- ? Green, D.R. (1988) "Design and Evaluation of Educational Software", *Coursenotes for the M.Sc. in Computer Education*, Loughborough University of Technology (unpublished).
- ? Hammond, N. and Barnard, P. (1984) "Dialogue Design: Characteristics of Human Knowledge", in A. Monk (ed.) *Fundamentals of Human-Computer Interaction*, Academic Press: London. *001.64404*
- Hampshire, N. (1989) "Mass Media", *Personal Computer World*, Vol. 12 No. 4.
- Hartley, J.R. (1987) "The Innovation of Computer Assisted Learning", *British Journal of Educational Technology*, Vol. 18 No. 3 pp. 210-220. *serial 370*

Hawkridge, D. (1983) *New Information Technology in Education*, Croom Helm:London.

Hetzel W. (1985) *The Complete Guide to Software Testing*, Collins.

Hilgard, E.R. (1958) *Theories of Learning*, Methuen & Co.:London. 153.15

Hooper, R. (1977) "An Introduction to the National Development Programme in Computer Assisted Learning", *British Journal of Educational Technology*, Vol.8 No.3 pp.165-173. Serial 370

6 ? Hulme, C. (1984) "Reading: Extracting Information from Printed and Electronically Presented Text", in A.Monk (ed.) *Fundamentals of Human-Computer Interaction*, Academic Press:London. 001.64404

✓ ? Isaacs, G. (1987) "Text Screen Design for Computer Assisted Learning", *British Journal of Educational Technology*, Vol. 18 No. 1 pp. 41-51. Serial 370

Jones, R. (1987) "Software Engineering: Pack Up Your Troubles in a CASE Environment", *Computing*, November 26th, 1987 pp. 28-30. 001.64 ?

Kearsley, G. (1985) "Automation in Training and Education", *Human Factors*, Vol.27 No.1 pp.61-74.

Kearsley, G.(1986) *Authoring: A Guide to Instructional Software*, Addison-Wesley:Reading, MA.

? Kearsley, G.P. and Hillelsohn, M.J. (1982) "Human Factors Considerations for Computer Based Training", *Journal of Computer Based Instruction*, Vol. 8 No.4 pp. 74-84.

7 Keller, A. (1987) *When Machines Teach*, Harper & Row:New York.

7 Kemmis, S., Atkin, R., and Wright, E. (1977) "How Do Students Learn ?" *Working Papers on Computer Assisted Learning Occasional Paper No. 5*, Centre for Applied Research in Education:University of East Anglia.

Klass, R. (1984), "The TenCORE Language and Authoring System for the IBM Personal Computer", *The Journal of Computer Based Education*, Vol.11 No. 3.

Lay, R. W. (1981) "Basic Techniques for Teaching BASIC", in R. Lewis & D. Tagg (eds.) *Computers in Education*, North-Holland.

Lindquist, T.E. (1985) "Assesing the Usability of Human-Computer Interfaces", *IEEE Software*, January 1985, pp. 74-82. Serial 001-6

MacDonald, B. (1977) "The Educational Evaluation of NDPCAL", *British Journal of Educational Technology*, Vol.8 No.3 pp.176-189. Serial 372

Maddison, J. (1983) *Education in the Microelectronics Era*, Open University Press:Milton Keynes.

Morrison, F. (1975) "Planning a Large Scale Computer Assisted Instruction Installation: The TICCIT Experience" in O. Lecarme and R. Lewis (eds.) *Computers in Education: Proceedings of the IFIP 2nd. World Conference*, North-Holland.

Neuman, M. (1987) "The Computer and Thinking Skills: Rationale for a Re-vitalised Curriculum", *AFIPS Conference Proceedings*, Vol. 56. AFIPS Press:Virginia.

- Nicholson, R.I. (1987) "A Short Term Plan for CAL", *Journal of Computer Assisted Learning*, No. 3 pp. 81-88. Serial 370
- Nicholson, R.I. and Scott, P.J. (1986) "Computers and Education: the Software Production Problem", *British Journal of Educational Technology*, Vol. 17 No. 1. pp.26-34. s 370
- Norman, D. A. (1984) "Cognitive Engineering Principles in the Design of Human-Computer Interfaces", in G. Salvendy (ed.) *Human-Computer Interaction*, Elsevier:Amsterdam. 001.64404
- Norman, D.A. (1986) "Cognitive Engineering" in D.A. Norman and S.W. Draper (eds.) *User Centred System Design*, Lawrence Erlbaum Associates: New Jersey. 001.64
- O'Neill, G. (1987) "Interactive Video as an Aid to Learning", *Programmed Learning and Educational Technology*, Vol.24 No.2 pp. 137-144. s 370
- O'Shea, T.(1979) "A Self-Improving Quadratic Tutor", *International Journal of Man-Machine Studies*, Vol.11 pp. 97-124. s 001.5
- O'Shea, T. and Self, J.(1983) *Learning and Teaching with Computers*, Harvester Press: Brighton. 391.39445
- Park, O., Perez, R.S. and Seidel, R.J. (1987) "Intelligent CAI: Old Wine in New Bottles, or a New Vintage ?", in G. Kearsley (ed.) *Artificial Intelligence and Instruction: Applications and Methods*, Addison Wesley: Reading, M.A. 323.39445
- Peled, A. (1987) "The Next Computer Revolution", *Scientific American*, Vol. 257 No.4 pp. 35-42.

Pogue, R. E. (1980), "The Authoring System: Interface Between Author and Computer", *Journal of Research and Development in Education*, Vol. 14 No. 1. pp. 57 - 68.

Potosnak, K. (1988) "Recipe for a Usability Test", in *I.E.E.E. Software*, November 1988, pp. 83-84. 5001.6

Richmond, W.K. (1965) *Teachers and Machines*, Collins:London.

Ridgeway, J. (1988) "Of Course ICAI is Impossible... Worse Though It Might Be Seditious", in J.Self (ed.) *Artificial Intelligence and Human Learning*, Chapman & Hall: London. 371.31445

Romiszowski, A.J. (1981) *Designing Instructional Systems*, Kogan Page:London. 375.001

Schneider, G. M. and Bruell, S.C. (1981) *Advanced Programming and Problem Solving with Pascal*, John Wiley:New York. 001.6424

Self, J.(1987) "The Institutionalisation of Mediocrity and the Influence of Outsiders", in E.Scanlon & T. O'Shea (eds.) *Educational Computing*, Wiley:Chichester. 370.02854

Sheppard, S. (1983) "Applying Software Engineering to Simulation", *Simulation*, Vol. 40 No. 1, pp. 13-19.

✓[?] Sherwood, B. (1987) "Bridging the Computer-User Gap", *AFIPS Conference Proceedings*, Vol. 56 pp. 185-192. Serial 001.64

✓[?] Shneiderman, B. (1987) *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley:Reading M.A. 001.64404

Skinner, B.F. (1954) "The Science of Learning and the Art of Teaching", *Harvard Educational Review*, Vol.24 pp.86-97.

Skinner, B.F. (1958) "Teaching Machine", *Science*, Vol.128 pp.969-977.

Skinner, B.F. (1961) "Teaching Machines", *Scientific American* No. 205, pp 90-102.

Steinberg, E.R. *Teaching Computers to Teach*, Lawrence Erlbaum:New Jersey. 371.39445

Strawford, G. (1988) *Authoring Packages: a Comparative Report*, N.I.V.C:London.

✓ 7 Sullivan, M.A. and Chapanis, A. (1983) "Human Factoring a Text Editor Manual", *Behaviour and Information Technology*, Vol.2 No. 2 pp.113-125. Serial 001.6

Tenczar, P. (1981) "CAI Evolution: Mainframe to Micro" in R.P. Zimmer (ed.) *Proceedings on the International Conference on Cybernetics and Society*, IEEE:New York. pp.443-448.

2 Van Nes, F.L. (1986) "Space, Colour and Typography on Visual Display Terminals", *Behaviour and Information Technology*, Vol. 5 No. 2 pp. 99-118. Serial 001.6

Wellington J.J (1985). *Children, Computers and the Curriculum*, Harper & Row:London.

Winfield, I. (1986) *Human Resources and Computing*, Heinemann:London. 001.64

Whiteside, J., Jones, S., Levy, P.S., and Wixon, D. (1985)
"User Performance with Command, Menu, and
Iconic Interfaces", *Proceedings of CHI '85 Human
Factors in Computing Systems*, A.C.M.: New York.

Whiting, J. (1989) "An Evaluation of Some Common CAL and
CBT Authoring Styles", *Educational & Training
Technology International*, Vol.26 No. 3 pp. 186-200. ⁵³⁷⁰
EDU

Wood, R. (1989) "Interactive Video: The Future of
Training", *Training and Development*, January 1989 p.
26.

Appendix A.

Suppliers of the authoring packages surveyed in Chapter 2.

Crystal.

Produced by: Intelligent Environments Ltd.,
Northumberland House,
15 - 19 Petersham Road,
Richmond,
Surrey.
TW10 6TP.

Mentor II.

Produced by: Mentor Interactive Training Ltd.,
Colonnade,
Sunbridge Road,
Bradford.
BD1 2LQ.

Microtext.

Produced by: Transdata,
61 Lever Street,
London.

ProCAL.

Produced by: VPS Interactive Ltd.
22 Brighton Square,
Brighton.
BN1 1HD.

TenCORE.

Supplied by: Systems Interactive Ltd.
235/245 Goswell Road,
Islington,
London.

Top Class.

Produced by: Format PC Ltd.,
Goods Wharf,
Goods Road,
Belper,
Derbyshire.

Unison.

Supplied by: Castle Learning Systems,
P.O. Box 741,
Chelmsford,
Essex.
CM2 9UL.

Appendix B.

The TenCORE Instruction Set.

Listed below are the instructions to be found in the TenCORE language. The list is grouped by function.

Calculation:

calc	calcc	calcs	clock
compare	compute	date	define
exchang	extin	extout	find
keytype	move	nocheck	pack
packc	recieve	return	set
setbit	setc	transfr	zero

Data Storage:

addname	addrecs	attach	created
createn	datain	dataout	ddisk
delname	delrecs	destroy	detach
disk	dread	dwrite	files
getname	idisk	memory	names
nsdirwr	rename	renamef	resizef
setname			

Display:

at	atnum	beep	blink
box	bright	charloc	chars
charset	circle	color	colore
colorg	dot	draw	ellipse
erase	fill	image	margin
mode	options	origin	page
palette	rotate	scale	screen
show	showa	showh	showt
showv	size	sixex	sizey
smooth	spacing	status	thick
video	window	write	writec

Judging

answer	arrow	endarrow	exact
exactno	ignore	no	ok
store	storen	wrong	

Judging Modification

blanks	copy	exit	holdno
holdok	jkey	jkeyx	judge
loada	long	nomark	noorder
nospell	okextra	okspell	put
putlow	rejudge	storea	

Sequencing

asmcall	back	BACK	backop
BACKOP	base	branch	cstop
data	DATA	dataop	DATAOP
delay	do	doto	else
elseif	endif	endloop	error
exec	exitsys	goto	help
HELP	helpop	HELPOP	if
index	INDEX	indexop	INDEXOP
intcall	jump	jumpop	library
loop	next	NEXT	nextop
NEXTOP	outloop	pause	quit
QUIT	quitop	QUITOP	reloop
restart			

Miscellaneous:

clearu	debug	device	disable
edisk	enable	force	initial
lesson	loadu	nextkey	noword
okword	operate	press	print
score	time	use	

Appendix C.

The TopClass Instruction Set.

The instructions available in the TopClass language fall into three categories:

- DOT commands, which are used to modify the display presentation.
- @ commands, which may be used to control the running and flow of the program.
- GRAPHICS commands, which are an extension of the DOT commands. They may be used if the computer is fitted with a graphics adapter.

Only the first three letters of a command are used by the language interpreter, but the long form of each command is given (in lower case) to enable the commands to be learned and recalled.

Dot Commands:

.ARRow	.AUDio	.BLAnk
.CDRom	.CENTre	.CLEAr
.COLour	.FUNction	.IN
.INFormation	.LET	.LOCate
.OUT	.PAUse	.PLAY
.REAd	.RESet	.SET
.SINgle	.SKIp	.SNAp
.SPEach	.TIME	.TXT
.USE	.WIPE	.WRite

@ Commands

@CHain	@COMport	@DOS
@EQUate	@ESSay	@GOBack
@GOTO	@GRAd	@IF
@MACro	@ON	@OPTion
@QBank	@RND	@SINgle
@STOp	@SYStem	@TESt

@ Question Commands

@BEGin	@END	@REAd
@RIGht	@TRies	@WRONG

Graphics Commands

.CIRcle	.DRAW	.FONt
.GET	.MOVE	.PAInt
.PALette	.PUT	.PSEt
.SCReen		

