INTEGRATING A SOFTWARE ENGINEERING APPROACH AND INSTRUCTIONAL FACTORS IN INSTRUCTIONAL SOFTWARE DEVELOPMENT - ILLUSTRATED BY A PROTOTYPE IN THEORETICAL COMPUTER SCIENCE

by

MARY RUTH DE VILLIERS

submitted in part fulfilment of the requirements
for the degree of

MASTER OF SCIENCE

in the subject

INFORMATION SYSTEMS

at the

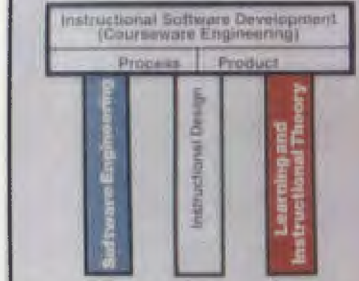
UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: MRS P KOTZé

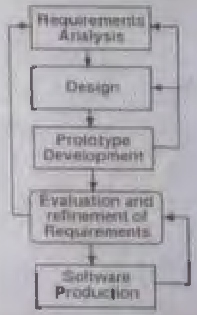
SEPTEMBER 1995

Combining a Software Engineering approach with Instructional Factors in Instructional Systems Development

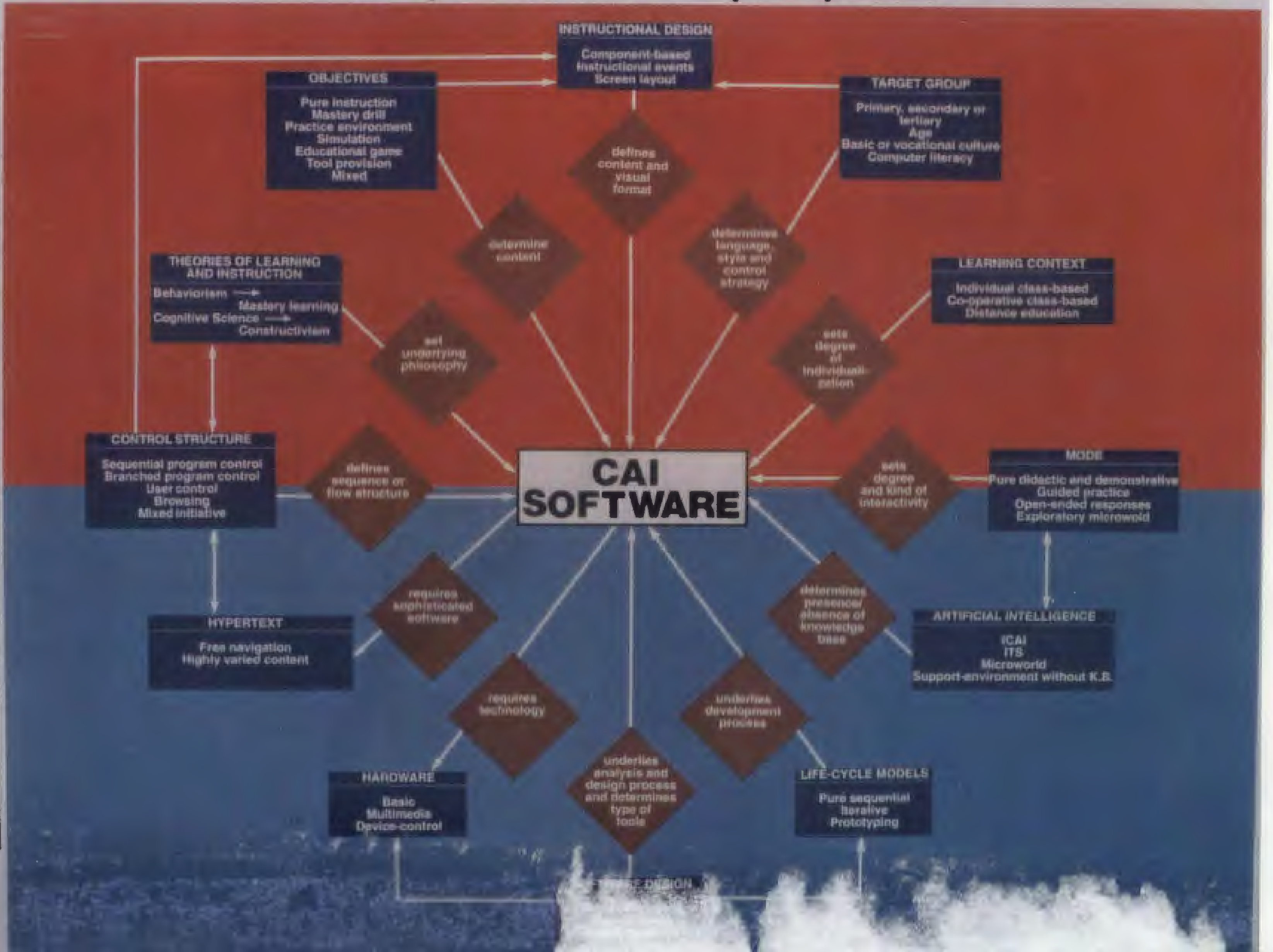
The roles of Software Engineering, Instructional Design and Instructional Theory as pillars of Instructional Systems Development


Instructional Software Development (Courseware Engineering)
Process
Product
Software Engineering
Instructional Design
Learning and Instructional Theory

A Prototyping Life-Cycle Model



The factors, both instructional and computer-oriented, that determine the nature of a CAI product and its development process



CAI SOFTWARE

INSTRUCTIONAL DESIGN
Component-based
Instructional events
Screen layout

OBJECTIVES
Pure instruction
Mastery drill
Practice environment
Simulation
Educational game
Tool provision
Mixed

TARGET GROUP
Primary, secondary or tertiary
Age
Basic or vocational culture
Computer literacy

THEORIES OF LEARNING AND INSTRUCTION
Behaviorism → Mastery learning
Cognitive Science → Constructivism

LEARNING CONTEXT
Individual class-based
Co-operative class-based
Distance education

CONTROL STRUCTURE
Sequential program control
Branched program control
User control
Browsing
Mixed initiative

MODE
Pure didactic and demonstrative
Guided practice
Open-ended responses
Exploratory microworld

HYPERTEXT
Free navigation
Highly varied content

ARTIFICIAL INTELLIGENCE
ICAI
ITS
Microworld
Support-environment without K.B.

HARDWARE
Basic
Multimedia
Device-control

LIFE-CYCLE MODELS
Pure sequential
Iterative
Prototyping

[illegible] DESIGN

defines content and visual format
determine content
determines language, style and control strategy
set underlying philosophy
sets degree of individualisation
defines sequence or flow structure
sets degree and kind of interactivity
requires sophisticated software
determines presence/ absence of knowledge base
requires technology
underlies development process
underlies analysis and design process and determines type of tools

ABSTRACT

This dissertation is a multi-disciplinary study, which integrates a software engineering approach with instructional factors in the decision-making, analysis, design and development processes of instructional software. Software engineering models, tools and representations are used in the process of software construction. With reference to the fundamental characteristics of the software product, several disciplines and factors, from both instructional and computing perspectives are considered, and the most appropriate approach/es selected. Software engineering, instructional design and instructional theory are considered as pillars of courseware engineering.

The object-oriented design paradigm and a prototyping life-cycle model are found to be most suitable for development of computer-aided instruction. The conceptual study is illustrated by prototype development of a component-based multi-activity practice environment in theoretical Computer Science. It offers perusal or practice, in various instructional modes, according to the user's preferred learning style or need.

Key terms (in alphabetic order):

Cognitive science; Component-based software; Computer-aided instruction; Courseware engineering; Instructional design; Instructional systems development; Instructional theory; Instructional transactions; Object-oriented design; Practice environment; Prototyping; Software engineering.

TO . . .

FOUR DELIGHTFUL YOUNG PEOPLE,

THE *ALPHAMERIC QUARTET*

Adrian '72

B

C

Dorothy '74

E

F

Gabrielle '76

H

I

Jonathan '78.

THANK YOU TO . . .

Paula Kotzé, my supervisor, for sound advice and guidance, encouragement, effective chasing, and for consistent prompt returns of the latest version! Also, Paula, for going way beyond the duties of a supervisor in giving technical help with the production of this document.

The Head of Department and my colleagues in the Department of Computer Science at Unisa, for continuous support and encouragement. Particularly, I would like to thank:

- **Prof Chris Bornman**, Head of the Department,
- **Prof Niek du Plooy**, the former Departmental Head, for getting me going,
- **Prof Willem Labuschagne**, the guru of COS101-S, for the pearls of wisdom,
- **Prof Martha Pistorius** and **Prof Lerine Steenkamp** for consistent genuine interest.

Malene le Roux, for sterling pioneering efforts in programming the prototype FRAMES. "Dankie, Malene, ons het baanbrekerswerk gedoen!"

Beatrice van der Westhuizen, for producing the diagrams and patiently working through the many iterations.

Estelle de Kock, for the graphics in FRAMES.

My son, **Adrian**, for the photographs of FRAMES, and to my colleague and friend, **Trish Alexander** for help and support in producing the colour screen displays.

Elsie Verheem, of the Unisa Science Library, and **Hester Mountifield**, the Computer Science Subject-Librarian, for the outstanding quality of their service.

Financial assistance for this research has been provided by the **Center for Science Development** of the Human Sciences Research Council of South Africa. Opinions expressed and conclusions reached in this dissertation, are those of the author and should not necessarily be attributed to the Center for Science Development.

I greatfully acknowledge this scholarship.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SCREEN DISPLAYS

CHAPTER ONE

INTRODUCTION

Computer-Aided Instruction (CAI) developed in the 1960s when the computer was perceived as a presentation medium superior to paper-based programmed instruction. Most early efforts were little more than electronic page turners, albeit with a degree of branching ability. The disciplines which impacted most upon CAI were the learning theory of behaviourism, and later, instructional design. It evolved from an educational perspective, with emphasis on its role as a medium of instruction and with less attention paid to its role as a computer applications program.

In recent years the power and performance of computer hardware has increased greatly and software applications development has become increasingly complex. Running on sophisticated technology, current software packages (referring to general computer software) are powerful, slick, graphically aesthetic and user-friendly, with a variety of interaction modes. These developments in the realm of software engineering have not left CAI behind. In the 1990s some outstanding educational and training packages have appeared on the marketplace.

> Instructional software resides no longer merely in educational territory. CAI programs, both the tutor- and the tool-variety, can be, in their own right, sophisticated pieces of software. As such, **they should be designed using state-of-the-art principles of software engineering, as well as a sound instructional basis.**

1.1 AIM OF THIS DISSERTATION AND AREAS OF INVESTIGATION

This MSc half-dissertation investigates the integration of software engineering with instructional theory and instructional design in Instructional Systems Development (ISD). Instructional theory and instructional design are used to define the requirements and characteristics of *the product*, and software engineering methodologies are combined with instructional design in *the process* of developing instructional software. Figure 1.1 illustrates the pillars of instructional systems development.

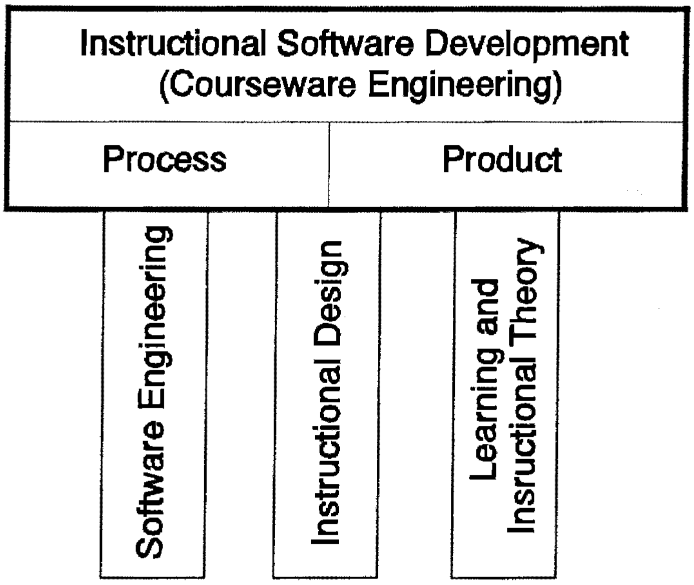


Figure 1.1 The roles of Software Engineering, Instructional Design and Instructional Theory as pillars of Instructional Systems Development.

The study is thus multi-disciplinary, breaking new ground in applying software engineering principles to the design and development of CAI, and systematically investigating the relevance of various factors, theories, methods and techniques, both instructional and computer-oriented, to ISD. The application of software engineering to the development of instructional software can also be termed Courseware Engineering.

Table 1.1 lists the disciplines and factors investigated, relating each to the aspect/s of CAI which it determines. The study of each main discipline commences with a thorough literature study, followed by a pertinent investigation of how it can be practically applied in decision-making. The solution develops as a specific design paradigm and life cycle model are selected. Software engineering modelling techniques are applied to the problem domain to facilitate design decisions. An instructional and learning philosophy is chosen for an educational foundation, and implemented using sound principles of instructional design as applied to computer courseware. Current developments in fields such as artificial intelligence, hypertext, and human-computer interaction are investigated to determine their relevance to the project in hand. Each factor is related to the appropriate phase/s in the systems development life-cycle.

Decisions are implemented and the process is illustrated by the design and development of a **prototype courseware package called FRAMES**, intended for use in distance-education of tertiary level students in Theoretical Computer Science.

Table 1.1 **Factors Investigated and Relevance to CAI**

DISCIPLINE/FACTOR	ENTAILING	RELATIONSHIP TO CAI
Theories and models of thinking, learning and instruction	Behaviourism, cognitive science, constructivism, events of instruction	Sets underlying philosophy of the instructional approach
Instructional design	Procedures and models	Relates to instructional characteristics and the instructional development process
Context, subject-matter and target population	The topic, type of instruction and intended learners	Determines content, style, appearance and level of software
Software engineering methodologies, models and tools	Development paradigms, life-cycle models, prototyping, data models, representations, etc.	Can expedite the development process, where CAI programs are complex pieces of software, offering rich, multi-faceted learning environments
The object-oriented design paradigm	An object-based approach, where an object encapsulates both data and processes on that data	Applicable to any instructional software which presents a variety of instructional components, practice activities, and utilities
Artificial intelligence	Intelligent CAI, intelligent tutoring systems, knowledge representation	Suitability or not of these techniques must be determined
Control, hypertext, usability and user interface factors	Human-computer interaction	Relate to control, interaction, individualization and ease of use

1.2 STRUCTURE OF THE DISSERTATION

Chapter One introduces the study and briefly outlines the subject-matter and target population of the prototype. In Chapter Two the author takes a view of the latest trends in software engineering in the realm of software *per se* (not as related to CAI). In Chapter Three these fields are once again integrated, but in a different way. It is a literature study on previous research combining the fields of software engineering and courseware design. Chapter Four investigates theories of cognition and learning, and Chapter Five relates these theories to instructional design. Instructional design remains under the spotlight in Chapter Six, where it is specifically applied to CAI. In Chapter Seven the role of artificial intelligence in CAI is studied, and Chapter Eight overviews the aspects of user-control, hypertext and human-computer interaction. A description of the development of the FRAMES prototype, with appropriate illustrations, is the subject-matter of Chapter Nine. Chapter Ten concludes the dissertation by referring back to Chapter One and reviewing the major features of the study. Figure 1.2 sets out the structure of the dissertation and shows the inter-chapter relationships.

Each of Chapters Two to Eight comprises a literature study of the topic, followed by an application of the relevant theory and procedures to CAI, relating that factor to CAI in general and, in particular, outlining its role in FRAMES. Chapter Nine sets out the step-by-step development of the prototype, structuring the chapter according to life-cycle phases. It integrates the material covered in previous chapters by describing the approach/es selected for each factor in the context of the appropriate phase.

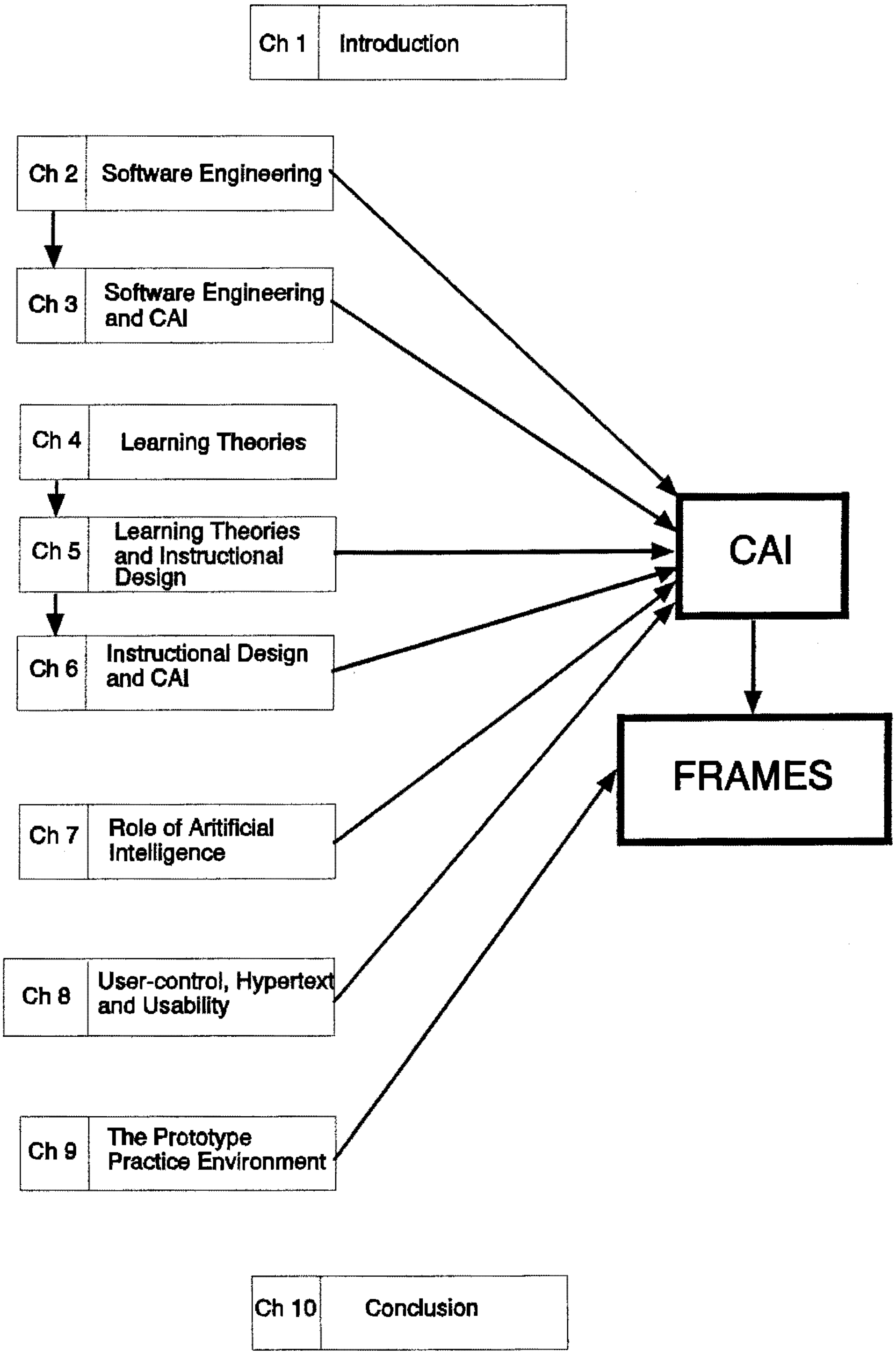

Ch 1
Introduction
Ch 2
Software Engineering
Ch 3
Software Engineering and CAI
Ch 4
Learning Theories
Ch 5
Learning Theories and Instructional Design
Ch 6
Instructional Design and CAI
Ch 7
Role of Aritificial Intelligence
Ch 8
User-control, Hypertext and Usability
Ch 9
The Prototype Practice Environment
Ch 10
Conclusion
CAI
FRAMES

Figure 1.2 Structure of the Dissertation

1.3 CONTEXT AND SUBJECT-MATTER OF THE PROTOTYPE

In order to develop the prototype, FRAMES, which illustrates this study, the researcher must select an appropriate approach for each discipline that impacts on ISD. As described in section 1.2, each of Chapters Two to Eight investigates a factor by considering its features and alternative approaches. In the application at the chapter's end, it is related to FRAMES, and appropriate solutions are recommended. With regard to the factors of **context, subject-matter and target group** of FRAMES, however, the parameters are fixed. Thus they do not require discussion in a separate chapter, and are incorporated into this introduction.

The Tertiary Institution and its CAI development facility

Unisa is a distance-teaching university in Pretoria, South Africa, with over 125 000 students. Tuition is handled largely by correspondence; students receive their tutorial matter and submit assignments by mail. Unisa is also a needs-driven pioneer in the development of instructional multi-media, and written instruction is increasingly supplemented by educational technology media such as radio, audio-cassettes, video, and CAI. The CAI Unit was established within the Department of Computer Science in 1989, and is currently based in the department's Centre for Software Engineering (CENSE). A team-based approach (see section 3.3) is used for the design and development of CAI. In an unpublished MEd thesis, de Villiers [De Villiers 1989a] of Unisa (not the present author) sets out a strategy for using CAI to supplement distance education of Computer Science. The initial products were indigenous custom-built courseware for the Department of Computer Science itself. Contract courseware and lessonware are now produced for various Unisa departments and bureaux, including the Faculty of Science, foreign language departments, administrative training sections, and the Student Services Bureau, for which a study methods package was developed.

The subject-matter

The module COS101-S, Theoretical Computer Science 1 for first-year BSc students, covers relevant mathematical concepts from the field of discrete mathematics, while the other first-level Computer Science modules are oriented towards basic computer systems and fundamental programming concepts. The author, who has a background in mathematics, computer science, and education is on the lecturing team of COS101-S.

Many students experience problems with COS101-S. The subsection involving analysis of "Relations and their Properties" has consistently been difficult for the target group. A binary relation is a set of ordered pairs, where an ordered pair comprises a first co-ordinate and a second co-ordinate.

Examples of ordered pairs are: (x,y), (2,2) and (3,1).

An example of a relation is the set: { (1,1), (1,2), (1,3), (2,2), (2,3), (3,3) }

The author designed a tutorial CAI lesson called RELATIONS [De Villiers, 1993] to supplement the textbook [Labuschagne 1993]. The lesson incorporates instructional segments and question segments on what might be described as *baby relations*, relations such as the example above, comprising a small number of ordered pairs. This study identified a need for further CAI to handle **more extensive and complex relations** defined by a mathematical inter-relationship or formula, rather than by a list, and which have an infinite number of members.

For example, the given relation, {(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)}, can be described as:

> The set of (x,y) pairs, where x is less than or equal to y, based on the domain set, {1,2,3}.

Analysis of the relation is relatively straightforward, based on a low number of inter-element comparisons. Now consider the relation:

> The set of (x,y) pairs, where x is less than or equal to y, based on as domain, the set of all integers.

In this case analysis must be carried out for a generic situation, since the number of inter-element comparisons is infinite. This type of abstract reasoning is far more complex and is handled by mathematical proof techniques.

The specific need

The CAI lesson, RELATIONS, handled the former kind of *baby relation*, but not the latter kind of infinite relation. Students involved in the formative evaluation and pilot-testing requested interactive, individualized material to help them with the more complex examples. The theoretical definitions and principles are the same as those for the elementary examples, hence the need is not as much for formal instruction as for **extensive practice of more complex exercises where the arguments and developments are in the form of mathematical proofs.** Theorem proving and the assessment of responses to open-ended questions are traditionally thorny areas in CAI (sometimes treated by artificial intelligence techniques). A prototype in this area would thus be the precursor of a system to meet an identified need at Unisa.

The target group

The target group comprises about 1000 students, of whom 90% live in the RSA. It is heterogenous with respect to age, race and academic background, comprising young school-leavers, technicians, teachers, and mid-career professionals requiring upgrading. Although some join informal study groups, they work largely in isolation, and experience the problems of a distance education environment. De Villiers, Pistorius, Alexander and du Plooy [De Villiers 1992] mention:

- No immediate feedback on written assignments; normal turnaround time (including postal delivery) exceeds three weeks,
- Irregular study schedules, due to the demands of their employment and/or personal circumstances, and
- Isolation.

Many of these problems are alleviated in a CAI milieu, where students may view material and do exercises in their own time and as often as they wish. Computer Science students, however, have a further advantage over other Unisa students, since computer access is compulsory. Not only are they computer-users, but they should be informed computer-users, and able to navigate through software packages other than merely the most elementary.

The prototype software

The name of the envisaged instructional system is **FRAMES**; the reasons for which are outlined in section 9.4.1. A prototype practice environment has been designed and developed:

- to illustrate the concepts discussed in this dissertation,
- to implement the solutions recommended,
- in the context of a real-life need in the Unisa module COS101-S.

REFERENCES - CHAPTER ONE

[De Villiers 1989a] de Villiers, C. (1989). *'n Rekenaargebaseerde onderrigstrategie vir Rekenaarwetenskap met besondere verwysing na afstandonderrig.* Unpublished MEd dissertation, University of South Africa, Pretoria.

[De Villiers 1992] de Villiers, C., Pistorius, M.C., Alexander, P.M. & du Plooy, N.F. (1992). Constraints on Computer-Assisted Instruction in a Distance Education Environment. *Computing Control Engineering Journal 3* (1), 11-13.

[De Villiers 1993] de Villiers, M.R. (1993). *Relations: A CAI Tutorial in Theoretical Computer Science.* Unpublished MEd mini-dissertation, University of Pretoria, Pretoria.

[Labuschagne 1993] Labuschagne, W. (1993). *A User-friendly Introduction to Discrete Mathematics for Computer Science.* Pretoria: University of South Africa.

CHAPTER TWO

SOFTWARE ENGINEERING

The 1950s and 1960s were decades of hardware. The 1970s were years of transition as the vital role of software was recognized. With the general availability of powerful, cost-effective micro-computing, the 1980s and 1990s are the decades of software. In the late 1960s Software Engineering (SE) was proposed - encompassing models, methods, and tools to enable systematic development and control within the context of burgeoning software production, thus providing methodologies by which to build high-quality, cost-effective software. In simpler terms it is the process of designing, building (i.e. programming) and maintaining application programs, which are computer programs developed to perform a specific purpose, traditionally in government or business. Software engineering evolved into a separate discipline within Computer Science, comprising life-cycle models, system modeling, analysis and design paradigms, programming techniques and tools, quality assurance, implementation and maintenance, and project management [Sommerville 1992; Conger 1994].

Educational software also proliferated in the 1980s, but quality was lacking, both in software aspects and in underlying learning theories. However, computer programs for instructional purposes are currently evolving from their simple beginnings to the realm of complex software, and the established principles, processes and models of software engineering should also be applied to CAI processes and products. Opportunities abound for CAI designers to build packages that optimize the capacities of the processors and peripherals of the 1990s, but cognisance must be taken of current SE techniques.

The approach in Chapter Two is to take an independent overview of software systems development, incorporating SE life-cycle models, design and analysis methods, and tools and representations which are relevant to current CAI. Particular attention is paid to prototyping life-cycle models and to the object-oriented methodology. The sources consulted are pure SE literature, rather than material combining the disciplines of SE and CAI, such as those which are studied in Chapter Three. Thus overlap occurs between the concepts studied in Chapters Two and Three, but the perspectives are different.

> Appropriate software engineering representations are used to demonstrate the factors impacting on ISD. This approach forms an important part of the requirements analysis in any process of CAI development. In subsequent sections of this study SE models are used to facilitate the analysis, design and development of FRAMES.

2.1 SOFTWARE LIFE-CYCLE MODELS

Every software product has a life-cycle, which commences when a need for it is recognized and ends with its retirement. Various Software Development Life-Cycle (SDLC) models exist, which describe the different approaches to the process.

2.1.1 THE WATERFALL LIFE-CYCLE MODEL

The conventional SDLC model [Schach 1990; Henderson-Sellers 1990; Booch 1991; Budgen 1994; Conger 1994] is the *Waterfall Model*, which entails a sequential project life-cycle for the creation of software. Theoretically it comprises a linear series of events, each of which is completed and "signed off" before the next commences. Different authors use different terminology, but in general it is said to have five phases:

1. Requirements analysis,
2. Functional specification,
3. Design,
4. Development (also termed implementation or programming), and
5. Testing.

Some authors extend the life-cycle beyond initial building to incorporate actual operation, thus adding phases such as Maintenance.

Figure 2.1 demonstrates the inter-phase relationships of the Waterfall Model and Table 2.1 identifies the major purpose of each phase. It is a so-called waterfall, because the output from each phase is fed into the next. In practice it is seldom purely sequential or linear:

- The stages often overlap,
- The process of executing each phase is often iterative, and
- The feedback from each phase tends to result in modification to previous phases, hence the weaker arrows on the left which show feedback links.

The broken line represents the boundary between initial development and actual operation.

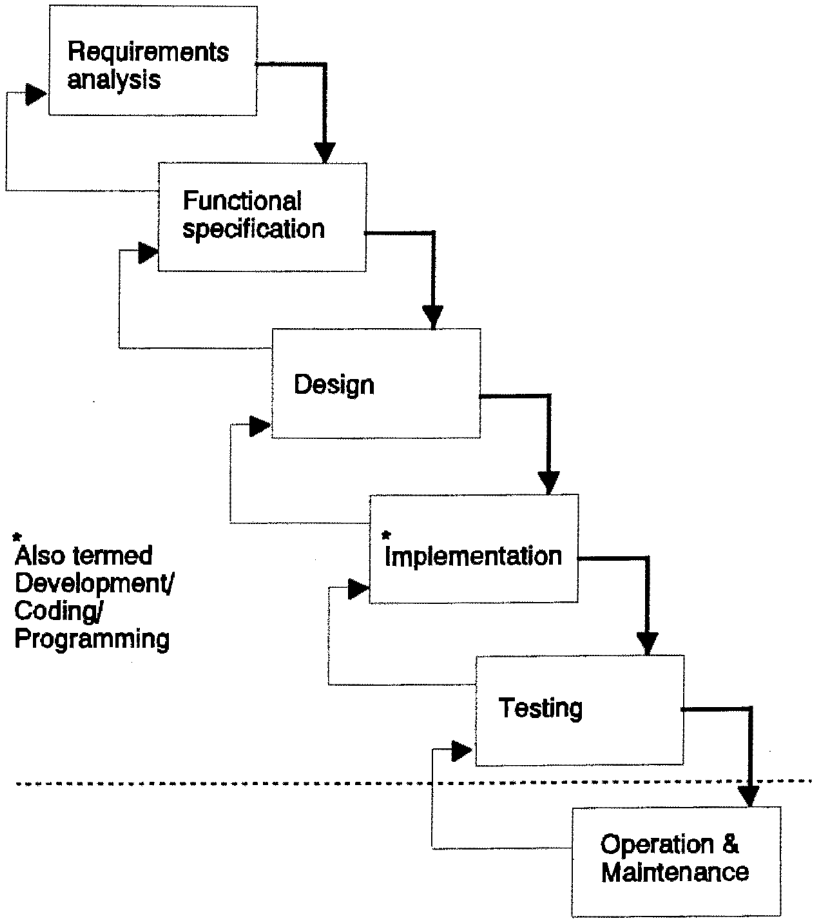


Figure 2.1 The Waterfall Model of Software Development

Table 2.1 The major Purpose of each Phase of the Waterfall Model

PHASE	PURPOSE
1. Requirements analysis	Identifying what is *needed* from a system.
2. Specification	Stating precisely *what* the system should do in order to meet the requirements.
3. Design	Defining *how* the system should perform its tasks, and demonstrating how inputs, outputs and user-interfaces should appear.
4. Implementation	Translating the design into a computer program.
5. Testing	Performing a validation of the implementation, to ensure that it complies with phases 1, 2, and 3.

2.1.2 THE PROTOTYPING LIFE-CYCLE MODEL

Software prototyping

The foundation of successful system development is a reliable set of user requirements. It is often difficult for a client or user to define precise requirements or to visualize how a proposed new system will work, and this can result in fundamental changes soon after installation. A solution is the software development methodology called *prototyping* [Powers 1990; Ince 1991; Conger 1994], which is the development of a limited *working version* of a product, primarily in order 'to obtain a more complete and stable definition of requirements for the new system"[Powers 1990, p. 55]. It entails the production of "a system or system component in a short period of time without formal written specifications" and which "circumvents the overload of documentation from the sequential life-cycle" [Conger 1994, p. 29].

By contrast, traditional software development models emphasize early constraining of design decisions and the production of exact and detailed specifications prior to any programming. The advent of powerful and modular development media and software tools has allowed the prototyping methodology, formerly restricted to fields such as aeronautical engineering, to be applied to software and courseware development. It is particularly beneficial in certain situations [Powers 1990; Conger 1994], of which the first four cases are highly relevant to CAI:

- for on-line interactive systems,
- when requirements are not well understood or cannot be precisely specified in advance,
- when the potential capabilities of the system go beyond the user's experience,
- when the utility or appropriateness of certain software or technology needs to be demonstrated hands-on,
- as a visual means of expediting communication between designers, programmers and end-users, and
- when rapid development of part of a system is required.

Prototyping is an iterative rather than a sequential process. It commences with the initial user requirements, which are revised, refined, and extended as the user interacts with the prototype. The prototype is modified on the basis of the changed requirements. The process may be repeated several times as the working model is built and modified fairly rapidly. Figure 2.2 demonstrates how the Waterfall Model of the SDLC is affected by the incorporation of initial rapid prototyping, with associated modification to the requirements and specification phases. It is a noticeably cyclic process as operations and maintenance send revised input into the development phases.

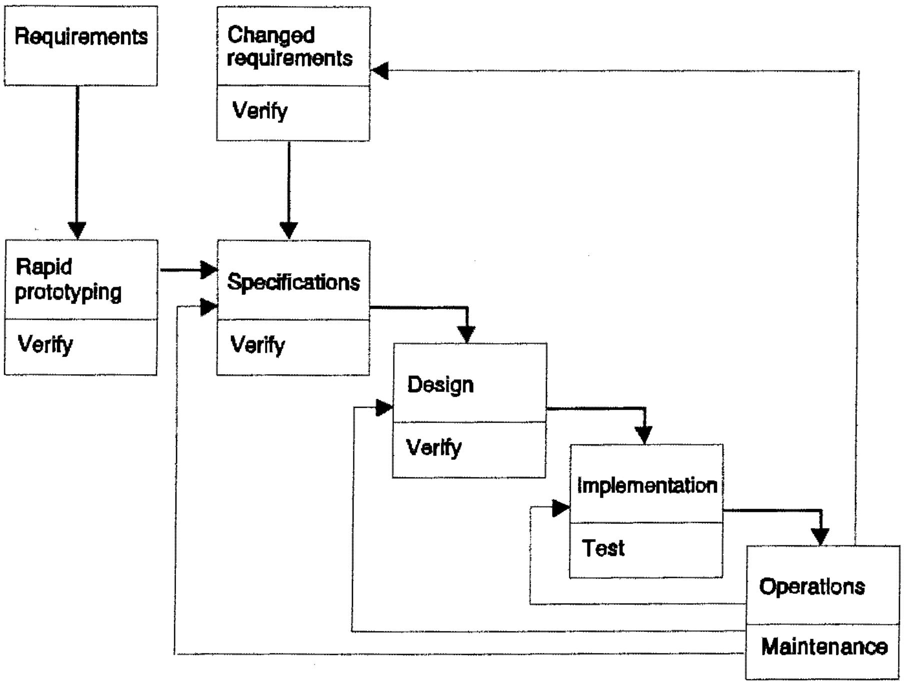


Figure 2.2 Impact of Prototyping on Software Development
[Based on Schach 1990, p. 51]

Ideally prototyping requires interactive software development tools that expedite development of screens, files and routines. A prototype should be produced quickly and cheaply. Various prototyping approaches are used, and, once again, different authors use different terminology. The two main thrusts are **evolutionary prototyping** and **throwaway prototyping** [Ince 1991].

Evolutionary (or Spiral) Prototyping

Evolutionary prototyping entails building a prototype as a limited version of the final product. As the prototype is expanded, modified and refined, it gradually evolves towards the final product. Thus the working model is converted into the production model. The impact upon the SDLC is that an analysis phase, as such, is eliminated, due to the **integration and iteration of analysis and prototyping**. Implementation involves the creation of test data, user-interface components such as menus, and the addition of system functions. Evaluation and modification processes follow; and finally the prototype is tuned to enhance efficiency. In the ideal situation, rapid development occurs, and an executable version of a system is available throughout the project.

Throwaway Prototyping

Often called rapid prototyping, throwaway prototyping involves the development of a version of a system which is used to clarify requirements and is then discarded, with conventional software development following. Lantz [no date] refers to a *mock-up*, but Powers *et al* [Powers 1990] reject this term as a detraction from the prototype's importance as an actual **working** model. The life-cycle can be considered similar to the conventional waterfall model, but with a vitally important *prototyping side-cycle* impacting particularly on requirements analysis and design. It is highly beneficial in situations where exact specifications are hard to formulate, or where optimal requirements cannot even be defined until some experimentation has been done. A prototype may contain imperfections and inadequacies, so long as it is a working model demonstrating at least partial functionality of the goal product. The prime aim is to facilitate communication and agreement between all parties concerned. Once the client or commissioner is satisfied with the visible aspects and products, conventional development occurs. Such a prototype is frequently built using a very high level language or a database query language.

Advantages of Prototyping [Lantz, no date]

- It produces a tangible entity in a situation where the client has difficulty visualizing what he wants a system to do.
- The client may evaluate a "system", rather than a requirements document or specification.
- It improves communication between all concerned parties, while simultaneously reducing documentation.
- End-users can participate in evaluation and design decisions.
- Development time and costs are reduced.
- The discovery of unanticipated problems may force the designer to modify the objectives, the approach, or the strategies.
- It produces a system that has been shown to work.

2.2 SOFTWARE DESIGN PARADIGMS

Software design is in essence a problem-solving task. It is more important to design a solution that will achieve its purpose in doing the required job properly, than to achieve elegance and efficiency at the expense of accuracy and reliability. **A designer needs to abstract the critical features of a system**, so as to concentrate initially on building a logical model of the system rather than becoming over-involved with detailed design and physical implementation at an early stage.

2.2.1 VARIOUS DESIGN APPROACHES

Various methodologies and representations are available to facilitate the processes of analysis, design, and system modeling, in particular, the process-oriented, data-oriented, and object-oriented approaches [Conger 1994]:

- The *process-oriented paradigm* centres around the events, procedures and flows that comprise a system. It is epitomised by concepts and tools such as top-down design, functional decomposition, transaction analysis, data-flow diagrams, structure charts and input-output transformations. It tends to discount evolutionary changes.

- *Data-oriented approaches* are based on the philosophy that data is more stable and unchanging than processes. The underlying theory is that of relational database theory, and key concepts are entities, attributes, relationships and normalization.

- The latest emerging methodology is the *object-oriented paradigm* [Coad 1990; Schach 1990; Bell 1992; Budgen 1994; Conger 1994], which integrates aspects of and uses formalisms from both the other major methodologies, and uses certain concepts from object-oriented programming languages. It is based on objects, which encapsulate both data and operations (processes) on that data. An object is a real-world entity whose processes and attributes are modeled in a computerized application. In object-oriented programming languages, computation is achieved when messages are passed to the objects in the program, and a central aspect is the abstract data type (ADT), which permits operations to be performed on an object without being implementation-specific. Objects incorporating data are identified as data entities and not as specific data structures.

CAI entails little conventional data flow and does not lend itself to the process-oriented paradigm. When applying software engineering methodologies to CAI the most appropriate is the object-oriented paradigm, which is discussed in more detail in the next sections.

2.2.2 THE OBJECT-ORIENTED PARADIGM (OOP)

Problems with traditional development based on the classical life-cycle include minimal iteration, little emphasis on re-use, and no unifying model to integrate the various phases. The OOP [Coad 1990; Henderson-Sellers 1990; Korson 1990; Schach 1990; Booch 1991; Atkins 1991; Bell 1992; Budgen 1994; Conger 1994] combines the strengths of other development methodologies in an effort to overcome their weaknesses. A further advantage of the OOP is that it projects beyond the pure software components and incorporates elements of the environment. Specifically, it recognises not only the data and procedures of an implementation, but also encompasses features such as users and the underlying philosophy as parts of the system.

Key concepts

The OOP has not achieved maturity or consistency in its practices or in its terminology. Nevertheless certain key concepts and practices emerge. The key concepts of object-oriented analysis and design are abstraction, encapsulation, modularity, and inheritance. *Abstraction*, which plays a major role in the design process in general, relates to describing the **essential** features of an object, its behaviour, and its relationships with other objects, while ignoring aspects irrelevant to the current perspective or purpose. *Encapsulation* refers to the integration of data with processes operating upon it. It also incorporates the concept of information-hiding whereby the implementation details of an object are concealed. The concepts of abstraction and encapsulation are closely related. *Modularity*, the partitioning of a system into components, is another concept encountered in the other methodologies. *Inheritance* is a property unique to the OOP. It allows the generic description of objects, so that subclasses "inherit" the properties of their more general class, but are specializations of it in some way. Object classes are arranged in hierarchies, with inheritance up the hierarchy.

Software design and program development

Even in the traditional life-cycle, the distinction between the systems design (broad design) and the program design (detailed design/coding) can become blurred. In the OOP, however, the boundary is even more indistinct, because both top-down analysis and bottom-up program development occur simultaneously or, at least, iteratively. The three traditional activities of analysis, design, and implementation are all present, but the joints between are seamless. The unifying factor is the prime role played by objects and their inter-relationships. Modeling is prominent in object-oriented design, the basic architecture being assembled from models of the entities and the relationships between them.

Object-oriented design and object-oriented programming are often confused. An object-oriented programming language allows the direct implementation of objects, and provides classes and inheritance, but object-oriented design is a design strategy and not dependent on any particular implementation language. An object-oriented design, i.e. a design where a system is designed as a set of interacting objects, can, in fact, be implemented in conventional programming languages.

Software re-use in the OOP

As a result of its prominent class and inheritance features, the object-oriented approach lends itself to code re-use, a characteristic which saves time and increases productivity.

Methods and tools of the OOP

Overlap occurs between the OOP and other development methodologies, therefore several of the standard methods and tools are appropriate, while others are custom-made for the OOP analysis and design. Henderson-Sellers & Edwards [Henderson-Sellers 1990] advocate data-flow diagrams (DFDs), object-relationship diagrams and inheritance charts. Coad & Yourdon [Coad 1990] include entity-relationship-attribute (ERA) diagrams and semantic data modeling in the object-oriented analysis process. They also use a graphical representation tabling the attributes and operations of an object class under its name, a notation re-used by Sommerville [1992]. Booch Diagrams [Booch 1991] are favoured by Conger [Conger 1994]. Certain applicable representations are described in detail in section 2.4.

2.3 OBJECT-ORIENTED LIFE-CYCLE MODELS

In section 2.1 SDLC models were considered, paying particular attention to **prototyping**. Section 2.2 overviewed software development methodologies with particular reference to the **OOP**. Concepts from both these methodologies are combined synergistically in this section on object-oriented life-cycle models. The approaches outlined in this section are particularly relevant to latest generation instructional software.

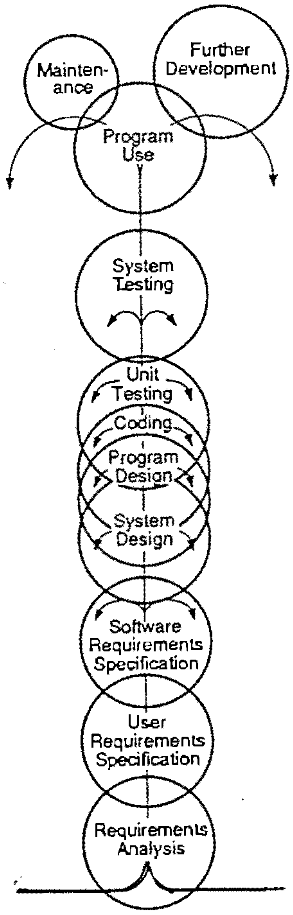
In the environment of the object-oriented paradigm, alternative life-cycle models have been proposed. Henderson-Sellers & Edwards [Henderson-Sellers 1990] developed the Fountain Model (Figure 2.3). It is characterized by:

- Inversion, so that requirements analysis and specification are explicitly shown as the foundation,
- Clear representation of merging, iteration and overlap,

- Upward growth culminating in a peak of program use, then "falling" for maintenance and extension, and
- Compatibility with prototyping.

Booch [Booch 1991] proposes a simpler life-cycle model (Figure 2.4) for object-oriented design. It resembles the waterfall model with a few notable differences:

- The all-embracing feedback loops which indicate the parallel, rather than sequential, nature of **all** phases,
- An "evolution" phase, resulting in high compatibility with prototyping, and
- Uniform strength of all arrows, granting equal status to both forward and backward processes.



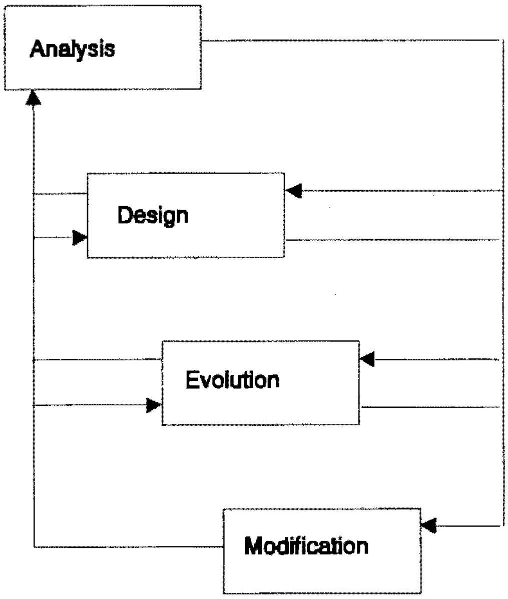


Figure 2.3
Fountain Model for the Object-Oriented Software Development Life-Cycle [Henderson-Sellers 1990, p. 152]

Figure 2.4
Booch Model of Object-Oriented Design in the Software Development Life-Cycle [Booch 1991, p. 200]

2.4 SOFTWARE ENGINEERING REPRESENTATIONS, TOOLS AND MODELS

This section reviews several SE representations and tools which are applicable to CAI analysis and design. They are not actually applied at this point, but their notations and functions are explained.

2.4.1 BOOCH DIAGRAMS

An object is an entity which has a state (whose representation is hidden) and a set of operations that can be performed on it. Objects communicate by passing messages to each other. Booch Diagrams [Henderson-Sellers 1990; Conger 1994], also called module structure diagrams, provide a graphical summary of the objects and messages between them. *Problem domain objects* are shown in vertical rectangles, carrying a smaller inset oval naming the object with small horizontal rectangles beneath to identify the individual processes (operations) on the object. *Service objects* appear in vertical rectangles with their names only. Lines drawn between objects signify *message connections*.

Figure 2.5 is a schematic representation of a Booch Diagram. A Booch Diagram is used in section 9.3.4 to demonstrate the objects, operations and messages in FRAMES.

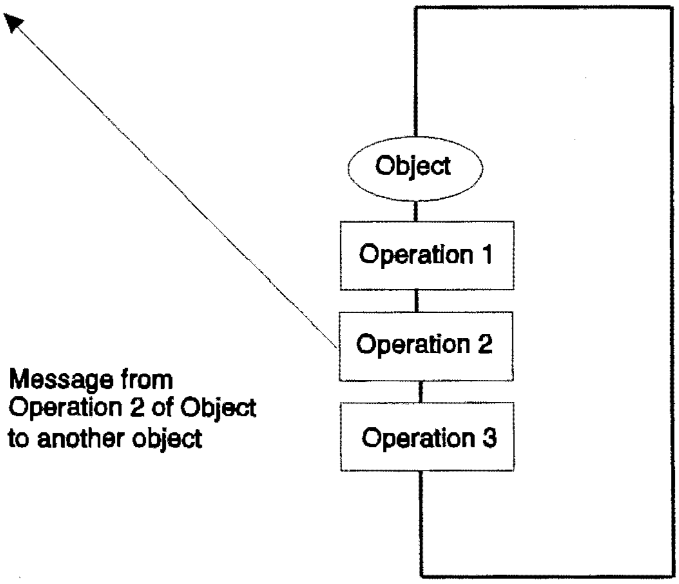


Figure 2.5 A Booch Diagram

2.4.2 MATHEMATICAL SET NOTATION

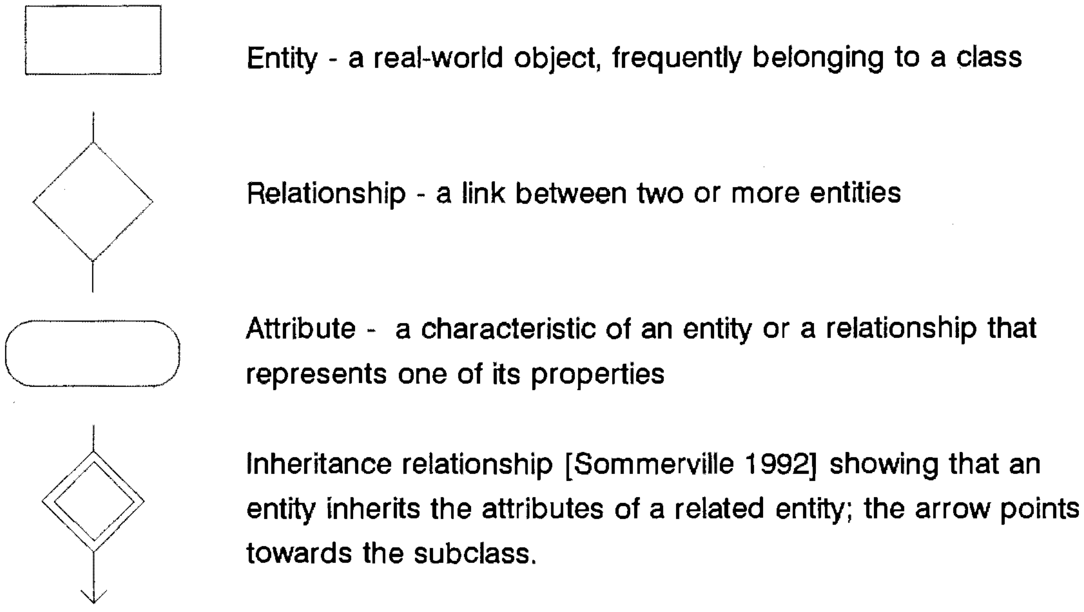
A set is a list of items separated by commas and contained within curly brackets. Items are combined within a set on the basis of a certain similarity or membership of a common class; the relationship may be due to a real life attribute or in terms of a mathematical formula. A set may have a finite or an infinite number of members. Examples are:

$A = \{x,y,z\}$
$\mathbb{Z} = \{\ldots,-2,-1,0,1,2,\ldots\}$
Deer = {elk, moose, reindeer}
$S = \{(x,y) \mid y = 2x-1, x \in \mathbb{Z}\}$

Sets are used in section 9.3.2 to list mathematical concepts used in the FRAMES exercises.

2.4.3 ENTITY-RELATIONSHIP-ATTRIBUTE DIAGRAMS

Chen's entity-relationship notation [Chen 1976] was originally defined for capturing the relationships between static data objects by means of visual portrayal of entities, characteristics and interrelationships. In particular the Entity-Relationship-Attribute (ERA) Diagram [Sommerville 1992; Budgen 1994; Conger 1994] plays a major role in database systems and in general information engineering. The principal symbols of the entity-relationship notation are shown in Figure 2.6:



Entity - a real-world object, frequently belonging to a class

Relationship - a link between two or more entities

Attribute - a characteristic of an entity or a relationship that represents one of its properties

Inheritance relationship [Sommerville 1992] showing that an entity inherits the attributes of a related entity; the arrow points towards the subclass.

Figure 2.6 Entity-Relationship-Attribute Notation

Special relationships are [Bell 1992]:

is-a	subclass-superclass relationship
is-part-of	component relationship
is-like	similarity relationship (useful in abstraction)

Extensions permit inclusion of the cardinality of a relationship (e.g. 1:1, 1:n, m:n) and, in an OOP environment, an inheritance relation, indicating a specialization which inherits the attributes of its superclass. The symbols are linked together to form a graphic representation of the objects and inter-relationships within a domain or system.

Figures 2.8(a) and 2.8(b) in this chapter use ERA notation to represent the factors impacting on CAI and the role each plays in determining characteristics of the system. In section 9.2.2 an ERA diagram models the entities and relationships in FRAMES.

2.4.4 COAD AND YOURDON NOTATION

Coad & Yourdon [Coad 1990] and Sommerville [Sommerville 1992] use a graphic representation based on round-cornered rectangles to set out the attributes of a class of objects and the services or operations that the class provides. It is particularly useful for representing objects that model messages or interactions.

Figure 2.7 illustrates Coad & Yourdon notation, which is used in section 9.4.2 to represent the screen-related objects of FRAMES.

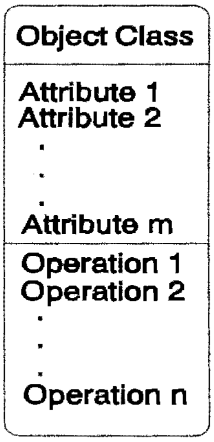


Figure 2.7 Coad & Yourdon Representation

2.5 APPLICATION OF SOFTWARE ENGINEERING TO CAI AND TO FRAMES IN PARTICULAR

In this section the most appropriate of the models, paradigms and tools outlined in this chapter are applied to CAI and to FRAMES in particular.

2.5.1 PROTOTYPING IN CAI

Prototyping in general systems development is discussed in section 2.1.2. The major purposes of conventional software are data processing and information processing, where defined activities occur in a predefined sequence. Instructional software, by contrast, comprises synthesis, presentation, practice and assistance facilities in the complex realm of human cognition, and has a high level of human-computer interactivity. Whether in a situation of program-control where the flow depends on user-response, or in a situation of user-control where the learner may branch or browse at will, the sequence of events and activities varies greatly. State-of-the-art CAI has several of the characteristics mentioned in section 2.1.2 that identify situations where prototyping is beneficial. CAI prototypes can play a vital role in demonstrating proposals on-screen, thus clarifying actual requirements and identifying misconceptions and potential errors at an early stage. Not only should basic aspects such as screen layout and colours be scrutinized, but also the strategies for control and navigation through the material. Usability factors, such as learnability and consistency can be evaluated, also interface aspects such as coherence of textual and visual displays, and accessibility of facilities. The development tools or authoring systems should offer:

- modularity, thus facilitating the removal, addition or adaptation of a segment without affecting other segments or the unit as a whole.
- plasticity, the ability to make changes easily.

If exorbitant time and costs are incurred in developing a prototype, the process is not cost-effective. An ISD prototype may either be *evolutionary,* i.e. a limited version of the final product, later developed through to full functionality, or else a *throwaway.* In the latter case, the software used to build the prototype may not be the same as that used for the final system.

FRAMES is intended to be a practice environment that will break new ground in the type of CAI developed at UNISA. It requires prototyping at the design and programming stages in order to ensure feasibility of intentions, to refine requirements, to reduce excessive written descriptions, to determine the optimal navigation and control strategies hands-on, and also to ensure that an appropriate programming approach is used for implementation.

2.5.2 OBJECT-ORIENTED DESIGN APPLIED TO CAI

The object-oriented paradigm was described in section 2.2.2. Object-oriented development has been used in large, complex systems, compared to which CAI courseware and environments comprise relatively few objects and components. Nevertheless the strategies outlined can be beneficially applied in the analysis, design and development of CAI, and specifically to FRAMES.

Budgen [Budgen 1994] describes an object as an entity which possesses a state, exhibits behaviour, and has a distinct identity. Sommerville [Sommerville 1992, p. 194] proposes the following definition: "An object is an entity which has a state (whose representation is hidden) and a defined set of operations which operate on that state. The state is represented as a set of object attributes". Analysis of a user-controlled practice environment such as FRAMES reveals distinct design elements, or objects, which possess unique identities, certain attributes and relationships, and have operations performed on them. The OOP appears to be the most appropriate software engineering development methodology for FRAMES.

2.5.3 AN OBJECT-ORIENTED LIFE-CYCLE APPLIED TO CAI

Section 2.3 combined the essence of prototyping (section 2.1.2) with the OOP (section 2.2.2) by describing the object-oriented life-cycle. The Booch model (Figure 2.4) of an SDLC, incorporating object-oriented design while emphasizing overlap and evolution, is suitable for development of instructional systems, and with the explicit incorporation of a prototype, could be appropriate for the type of development envisaged for FRAMES. FRAMES has well-defined objects, both concrete and abstract, and with its initial lack of precise specifications, appears to require a development process incorporating evolutionary prototyping.

2.5.4 SE REPRESENTATIONS APPLIED TO CAI DEVELOPMENT

Chapter Nine sets out a description of the FRAMES development process and the prototype software, incorporating use of some software engineering tools and representations mentioned in this chapter. In this section, however, the factors that impact on CAI are set out and an overview is given of some of the design choices for FRAMES. This "preview" sets the context for the subsequent chapters where the major factors are considered one by one.

CAI is highly multi-disciplinary. The prime factors impacting upon instructional computing are the disciplines of learning theory, instructional design and software engineering. Within the compass of each of these, however, are variables and sub-disciplines presenting options between which choices must be made. The decisions made here are part of the pre-design phase of instructional software, and are used in formulating the initial user-requirements.

Figure 2.8 sets out the major factors to be considered prior to the analysis and design of any instructional software, whether of the tutor- or tool-variety. Each factor has various options, and the most appropriate for the purpose in hand should be selected. The two major factors influencing instructional computing are instructional design and software engineering, shown in Figure 2.8(a). The detailed expansion, Figure 2.8(b), sets out all the factors, sub-disciplines and variables that must be considered when deciding on the requirements and design of a CAI system. ERA-style diagrams are used to represent the factors, their possible "values", and the relationships. Instructionally oriented factors appear in the upper half, and the factors from a software engineering or Computer Science perspective are in the lower half. Inter-relationships and inter-dependencies between the factors are also indicated.

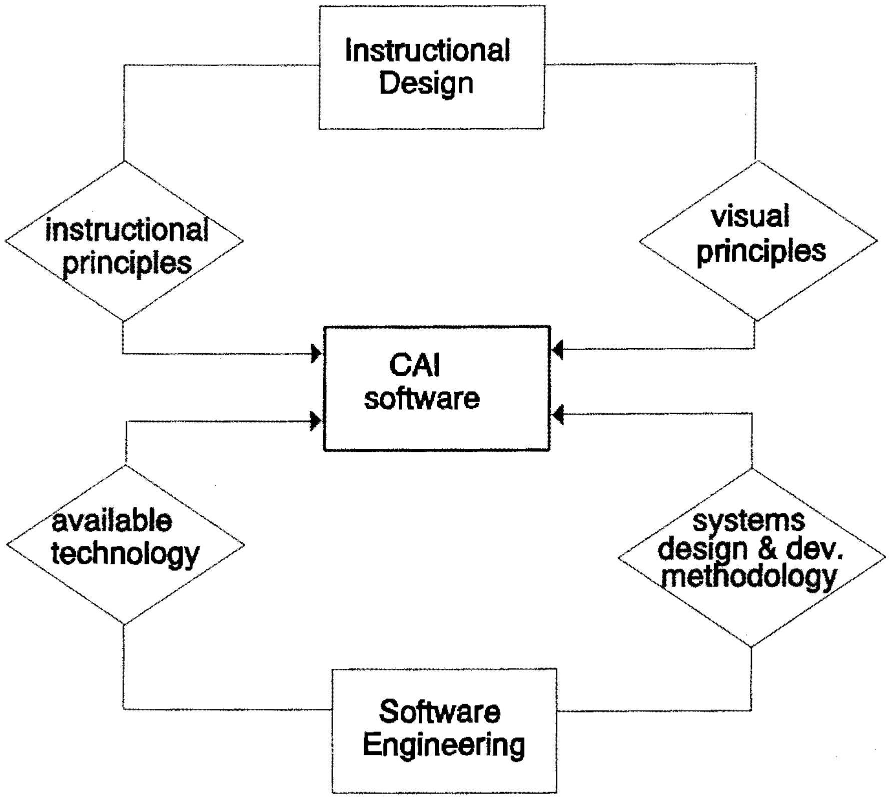


Figure 2.8(a) The Two Major Factors Impacting on CAI.

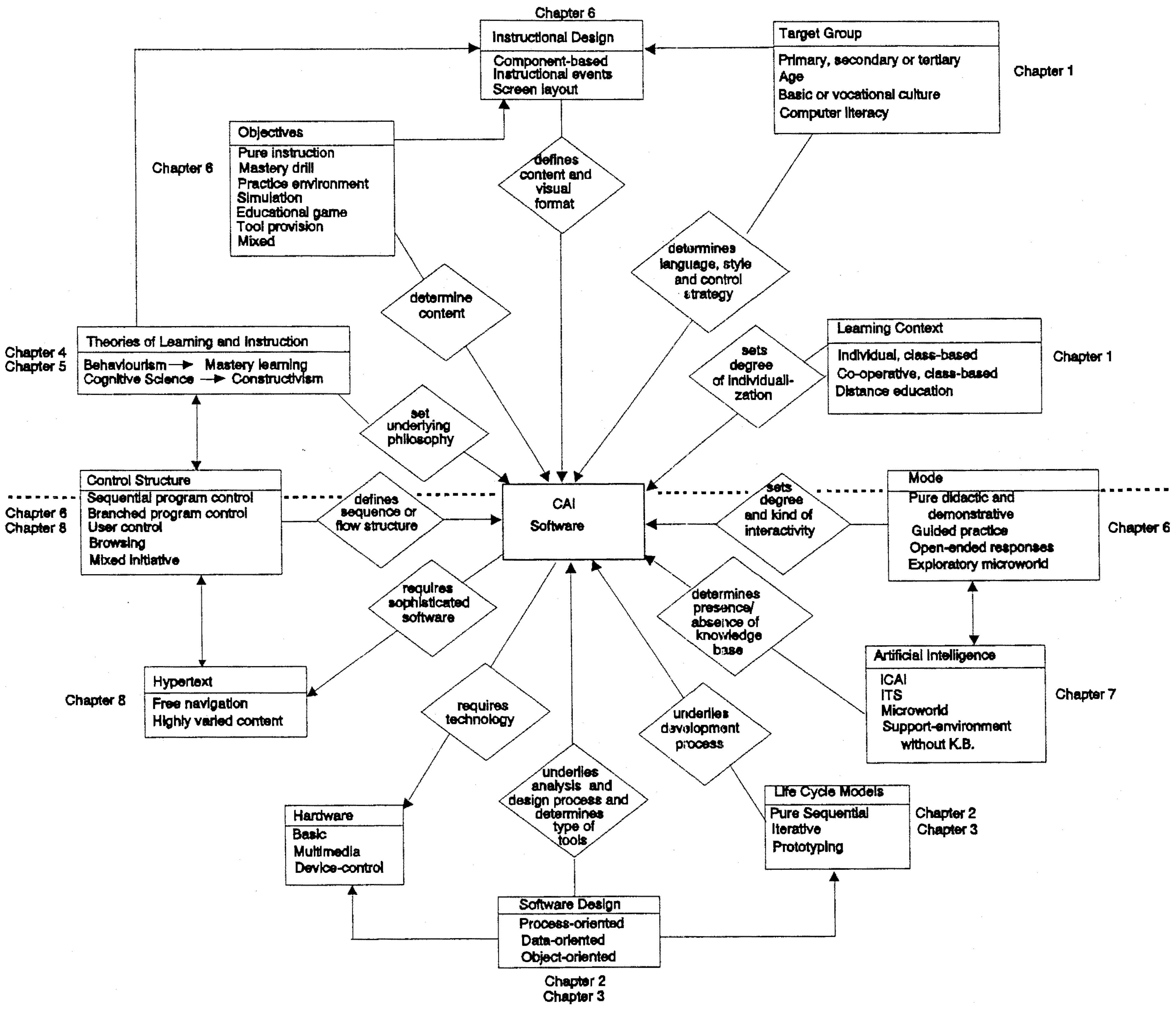


Figure 2.8(b) All the Factors impacting on CAI
(Instructional above and computer-oriented below)

In the ensuing chapters each factor is analyzed and applied to the design of FRAMES. Bearing in mind the subject matter, environment and goals of FRAMES, appropriate choices are made for each factor. The decisions made are summarized in Table 2.2. The target group, context and subject-matter are not considered as variables. They are assumed to be the starting point, parameters fixed at the time a design is commissioned or when an educator is motivated to meet an identified need.

Table 2.2 Design Decisions for FRAMES

FACTOR	DESIGN CHOICE
Target Group (Chapter 1)	Tertiary-level Computer Science students with computer-expertise
Learning Context (Chapter 1)	Distance education; students working in isolation
Software Design Methodology (Chapters 2, 3)	Object-oriented
Life-Cycle Model (Chapters 2, 3)	Evolutionary prototyping
Learning Theory (Chapters 4, 5)	Cognitive science, implemented in a constructivist approach
Instructional Design (Chapters 5, 6) (a) Kind of CAI/Objectives (b) Screen Appearance	Component-based (a) Practice-environment (b) Full-screen layout; graphic icons, some resemblance to windowing environments
Artificial Intelligence (Chapter 7)	Pseudo-intelligent help, minimal knowledge-base
Control Structure (Chapter 8)	Proactive user-control
Hypertext (Chapter 8)	Free navigation among varied content
Mode (Chapters 6, 8)	Three modes of doing examples: Demonstration Guided practice Do-it-yourself (D.I.Y.)
Hardware	Colour monitor, VGA graphics card, 640K memory, high-density 3,5" or 5,25" diskettes, mouse

2.6 CONCLUSION

The chapter overviewed general software engineering models, tools and techniques, and investigated their applicability to instructional systems development.

A life-cycle model which includes evolutionary prototyping appears to be most appropriate for CAI, so that initially fuzzy requirements can be refined and the initial working version can be modified and expanded towards a final operational CAI product.

The object-oriented methodology proves itself to be, in the terms of Korson & McGregor [Korson 1990], "a unifying paradigm", which is appropriate for the analysis and representation of CAI. Viewing a system as object-based provides a more versatile foundation than a view based fundamentally on data modeling or on its functions and procedures. Although user-input plays a major role in determimiing the path through instructional software, there is little conventional data flow. The concept of an object is a utilitarian approach, which brings under one umbrella such varied items as concrete objects, abstract objects, data, processes, and environmental entities external to the software (yet vital components of the system), such as the human user. Incorporation of the user as an object is particularly beneficial in CAI, due to its highly interactive and individualized nature. The tools and representations of the OOP can also be of great value in the analysis and documentation of instructional software.

It is hoped that the object-based control structure developed for FRAMES can eventually be used as a **generic, content-free shell** to present practice exercises in different instructional modes in varying subjects and courses. This would capitalize on the modularity and re-use potential inherent in an object-oriented design.

REFERENCES - CHAPTER TWO

[Atkins 1991] Atkins, M.C. & Brown, A.W. (1991). Principles of Object-Oriented Systems. In: McDermid, J.A. (Ed.), *Software Engineer's Reference Book.* Oxford: Butterworth-Heineman.

[Bell 1992] Bell, D., Morrey, I., & Pugh, J. (1992). *Software Engineering: A Programming Approach* (2nd ed.). Hemel Hempstead: Prentice Hall International (UK) Ltd.

[Booch 1991] Booch, G. (1991). *Object-Oriented Design: with Applications.* Redwood City, CA: Benjamin/Cummings Publishing Company, Inc.

[Budgen 1994] Budgen, D. (1994). *Software Design.* Wokingham: Addison-Wesley.

[Chen 1976] Chen, P.P. (1976). The Entity-Relationship Model: Towards a Unified View of Data. *ACM Trans. Database Systems 1* (1), 9-36.

[Coad 1990] Coad, P. & Yourdon, E. (1990). *Object-Oriented Analysis.* Englewood Cliffs, N.J.: Prentice Hall

[Conger 1994] Conger, S.A. (1994). *The New Software Engineering.* Belmont, CA: Wadsworth Publishing Company.

[Henderson-Sellers 1990] Henderson-Sellers, B. & Edwards, J.M. (1990). The Object-Oriented Systems Life Cycle. *Communications of the ACM 33* (9), 142-159.

[Ince 1991] Ince, D. (1991). Prototyping. In: McDermid, J.A. (Ed.), *Software Engineer's Reference Book.* Oxford: Butterworth-Heineman.

[Korson 1990] Korson, T. & Mc Gregor, J.D. (1990). Understanding Object-Oriented: A Unifying Paradigm. *Communications of the ACM 33* (9), 40-60.

[Lantz ??] Lantz, K.E. (no date - possibly 1985). *The Prototyping Methodology.* Englewood Cliffs, N.J.: Prentice-Hall.

[Powers 1990] Powers, M.J., Cheney, P.H. & Crow, G.B. (1990). *Structured Systems Development: Analysis, Design, Implementation.* Boston, MA: Boyd and Fraser Publishing Co.

[Schach 1990] Schach, S.R. (1990). *Software Engineering.* Boston, MA: Aksen Associates Inc. Publishers.

[Sommerville 1992] Sommerville, I. (1992). *Software Engineering* (4th ed.). Wokingham: Addison-Wesley Publishing Company.

CHAPTER THREE

SOFTWARE ENGINEERING AND COMPUTER-AIDED INSTRUCTION

The previous chapter was an independent overview of models and methodologies of conventional software engineering and a consideration by the author of their applicability to computer-aided instruction. Designers of instructional software should take cognisance of current software engineering techniques to improve the quality both of their process and their product.

Chapter Three also handles the relevance of software engineering to computer-aided instruction, but with a different approach. It overviews literature and previous research on combining the disciplines of CAI and SE. Since the mid 1980s it became apparent that the development procedures of instructional software left much to be desired. Educationalists involved in the process recognized the need both for participation from computing specialists as team-mates, and for the adoption of recognized SE procedures in the analysis, design and development of CAI. The resulting discipline is referred to in some quarters as Courseware Engineering.

3.1 GENERAL IMPACT OF SOFTWARE ENGINEERING ON CAI

A landmark paper [McLean 1989] entitled "Megatrends in Computing and Educational Software Development" followed the pattern of Naisbitt's "Megatrends" [Naisbitt 1982], but identified envisaged changes in **educational computing** due to the changed computing environment, current practices in software design, the user-centred approach, and software development trends. Reviewing the ever-increasing facilities available to instructional software designers, McLean [McLean 1989, p. 56] believes that "**practices from the field of software engineering are now necessary** to produce significant software packages that exploit the newer hardware and operating systems in a coherent and robust way" (bold font by the author, not by McLean).

3.2 SYSTEMATIC DEVELOPMENT AND DOCUMENTATION OF CAI

Self [Self 1985] expressed concern about the authoring systems of that time which enabled non-programmer educationalists to write software. Most were suited merely to the creation of text-based tutorials with a prespecified pattern, or to rigid programmed-learning dialogues. A further problem was the implicit pedagogical style - chunks of knowledge are presented to the learner, without the computer itself having any understanding of the material presented or the pupil's comprehension of it. Despite the advantages of by-passing software professionals in the development process, the result was use of the computer mainly as a delivery mechanism, rather than capitalization on its unique capabilities. Self recognised a need for substantial design effort to improve the quality of CAI and listed seven major stages in general software production, namely:

1. Requirements analysis,
2. Specification,
3. Program design,
4. Program implementation (programming),
5. Testing,
6. Debugging, and
7. Publishing.

He stated candidly that this phased process bore little resemblance to the production processes of most educational software. Referring to the term, "software engineering", he called on CAI instructional designers to note that programming courseware is indeed engineering "in that it is concerned with the economical design of a tangible product of practical value" [Self 1985, p. 89].

An effort was made at the University of Victoria in Canada to develop quality educational software by collaboration between staff from the Faculties of Education and Computer Science [Collis 1987]. The educators focused on the specification of pedagogical content, while the perspective of the computer scientists was to stress software engineering principles. The result was a multi-participant, multi-version approach to computer-based lesson creation. The constructs stressed in software development were those SE tenets of the mid-1980s, documentation and modularization.

An associated development at the same venue was a university course with an educational and a software engineering orientation to train educators in software development. Instructional design principles were taught in such a way as to be relevant both to educators and to software engineers. The prime principles taught are shown in Table 3.1. The emphasis on documentation and modularization is very clear.

Table 3.1 Principles entailed in Educational Software

INSTRUCTIONAL DESIGN PRINCIPLES	SOFTWARE ENGINEERING PRINCIPLES
Instructional objectives Instructional strategies Interaction and feedback Motivation Evaluation	Documentation Modular independence Modularity of program Development by team Multi-version software products

Bowers [1989] also emphasized written documentation in the development of instructional software and proposed an extensive "Design Document", incorporating:

- Description of the purpose and content of the software,
- Development plan,
- High-level overview of program operation,
- Flowchart (or other model),
- Narrative description of program logic,
- Narrative description of main routines,
- Reproductions of discrete screen displays, for example, text, graphics, storyboards, and
- Description of software testing and evaluation procedures, including their results.

A move in the direction of systematic development of instruction using educational technology in general was made by Verhagen & Plomp [Verhagen 1988]. They propose that educational technology should be viewed as the methodology of educational problem-solving, and as such, should be treated holisticly, with a systems approach. A systems approach entails an in-depth study of inter-relationships, both those within a product and those between the product and its environment. It advocates a systematic approach to the planning and development of computer-assisted learning, as was also advocated by Black [Black 1987].

The Unisa CAI Unit was introduced in section 1.3. Pistorius [Pistorius 1992] describes Unisa's 5-phase courseware development model which consists of:

Phase 1:	Preparation for project
Phase 2:	Pre-design
Phase 3:	Design
Phase 4:	Programming and Formative Evaluation
Phase 5:	Summative Evaluation

Phase 4 involves iterative modification based on feedback from the evaluation.

3.3 TREND TOWARDS TEAM APPROACH

As has already been stated, much CAI was developed by educators using authoring systems. As an alternative, educationalists Black [Black 1987, Black 1988] and Black & Hinton [Black 1989] of the University of Surrey, and Chen & Shen [Chen 1989], advise a multi-disciplinary team approach to design and implementation, incorporating, for example, an educational systems analyst or instructional designer (team leader), a subject-matter expert, technical advisor/s (e.g. hardware system specialist, graphics artist) and a programmer. Gery [Gery 1987], a pioneer of Computer Based Training (CBT) within business organizations, defines fourteen roles in CBT development! These are project manager, client (or sponsor), instructional designer, subject matter expert, writer, editor, programmer, data entry specialist, media expert, graphics designer, technical systems specialist, production administrator, CBT administrator and a learner. Roles and expectations should be clearly defined. The roles may not be distinct - in a small project one person may assume several roles; in a larger development several people may perform the duties entailed in one role.

Courseware development teams have been implemented at the Centre for Software Engineering (CENSE) in the Department of Computer Science at Unisa [Pistorius 1992], where CAI is produced for distance education as described in section 1.3. The philosophy of the CAI Group at CENSE is that its strength lies in its inherent team approach. Teams comprise a subject matter expert who also takes the role of designer (after suitable training in instructional design), programmers, a graphics expert, consultants and evaluators. Formative evaluation is the process whereby objective critics appraise firstly the designs and later the running product to identify errors, weaknesses and sources of possible confusion. There should be *peer-evaluation* by colleagues of the subject-matter expert (such as co-lecturers), and *learner-evaluation* by members of the target group.

3.4 ADVOCATION OF SOFTWARE DESIGN METHODOLOGY, LIFE-CYCLE MODELS AND PROTOTYPING

3.4.1 SOFTWARE DESIGN METHODOLOGY AND LIFE-CYCLE MODELS

Without actually using the term *life-cycle*, Self (see section 3.2) was one of the first to recommend a life-cycle approach to courseware development.

Black [Black 1988] and Black & Hinton [Black 1989] investigate the "message" from Software Engineering to Courseware Design Methodology. They recommend that, as in conventional SE, courseware development should be characterized by:

- ♦ Design methodology which is language-independent, and permits the possibility of changing language within a courseware package.
- ♦ The use of software design tools and representations, for example, high-level structure charts to represent logic, route charts (e.g. flow charts) and documentation.
- ♦ Structured programming and stepwise-refinement.

They point out the strong similarities which exist between the analysis and design processes in software engineering and courseware creation. These are shown in Table 3.2.

Table 3.2 Similarities between Analysis and Design in SE and CAI
[Black 1989, p. 29]

SOFTWARE ENGINEERING	COURSEWARE DESIGN
Requirements definition, problem-specification: Define problem; collect information about it	Needs analysis determining training needs
Identify performance requirements, standards for development, and any design constraints and systems dynamics	Identification of aims/objectives of course and lessons
System Design Top down: Decomposition from a high level; from abstract problem definition to a more detailed level	Selection of teaching/learning mode and context of CAI courseware
	Task analysis, selection of CAI modes
Software Design using: a) data flow charts, structure charts, etc. b) possibly modelling/simulation of final package	Elicitation of skills, teaching/learning style from subject expert, captured in: a) documentation including routing charts, b) a prototype with screen designs
Software Implementation:	Programming of courseware package
Coding of the final package, trials, debugging, with continual verification and validation	Trial with learners
	Integration of courseware into instructional programme, field trials and evaluation
	Orientation of trainers

Tripp & Bichelmeyer [Tripp 1990] also point out that software design (part of software engineering) and instructional design (part of education) have similar methodologies.

The classic waterfall model of software design has five steps:

> Requirements analysis, design, implementation (meaning development), testing, and maintenance.

Similarly the interservices Instructional Systems Development (ISD) model for instructional design has five steps:

> Analysis, design, development, implementation (in this context referring to operational use), and control.

Both fields tend to be dominated by individuals, *champions*, so designers must attempt to bring about orderly and replicable practices and should emphasize formative evaluation of products. However, the most fundamental difference between the two is the degree of rigour required. Conventional software packages are based on mathematical logic, where there is no error-tolerance and precise-format input is required. Instructional software, on the other hand, aims to facilitate human cognition and must accept input based upon such. This realm is inherently broader and must accommodate some ambiguity. Nevertheless, instructional designers can only benefit by studying and applying methodologies of software engineering.

3.4.2 PROTOTYPING

The research done by Black [Black 1988] and Black & Hinton [Black 1989] at the University of Surrey strongly advocates the use of a **prototyping life-cycle model for computer-aided learning**. They point out how CAI is intrinsically different from other learning media, because of its dynamic displays and interactivity, overlays, graphics and animation, simulation, and analysis of user-input. A further inherent characteristic is the dependence of control-flow on user-input. The need to observe and evaluate these aspects makes software prototyping a useful component of CAI courseware development. They recommend construction of a limited version with examples of screen design, route schemes, input and corresponding system response. Black suggests that subject experts using graphics packages should implement and demonstrate provisional screens. Authoring systems can be used to experiment with learning sequences. Control in such prototypes can be activated by simple keyboard commands, rather than by the actual user-input that would ultimately drive the interactions. Thus a working model is built that excludes the intricate programming necessary for a finished and polished version. Team-evaluation provides feedback for modification.

Tripp & Bichelmeyer [Tripp 1990] also note the **similarities between software design and instructional design**, and advocate software design methodology and rapid prototyping in

CAI. They refer to Simon [Simon 1981] who proposed the term *sciences of the artificial*, as opposed to the *natural sciences,* to encompass disciplines such as engineering, medicine, architecture, cognition and instruction. These are the sciences of design - concerned "not with how things are but with how they might be" [Simon 1981, p xi]. They encompass domains where solutions are man-made rather than natural, and where problem-solving can be characterized in terms of functions, goals and adaptation. Requirements specification and design of instructional software tends to begin as a conjecture, which, after inspection, undergoes modification. Rapid prototyping permits the pragmatic design principles of minimum commitment and concomitant modification [Sakasai 1990]. Tripp & Bichelmeyer [Tripp 1990] recommend that a prototype should include any required database (or part of it), major program modules, screen displays, and adequate inputs and outputs for interfacing. "The use of rapid prototyping in software engineering is essentially the extension of a successful design methodology into a new domain" [Tripp 1990, p 35].

Lantz [Lantz, no date] draws **comparisons between software and courseware production.** In software development explicit logical and physical definitions are necessary. The corresponding elements in CAI are an instructional objectives and instructional strategy respectively, concepts that can be hard to specify precisely. Rapid prototyping and the inclusion of end-users in evaluation enhance communication between all parties. The initially fuzzy areas evolve into refined definitions and precise designs, and at that stage detailed coding can be undertaken.

Based on the waterfall model, Chen & Shen [Chen 1989] **propose a life-cycle model for instructional software development** (Figure 3.1) with the joint objectives of high-quality products and development effectiveness. Verification and revision occur after each phase, resulting in an iterative, cyclic process. In their approach, prototype construction resides after analysis and before design. Inspection of the model shows a thorough and rigorous approach to all aspects:

- the instructional foundation,
- the instructional development process, and
- the software development phases.

The waterfall model is extended to seven phases, and the seven boxes on the right list the activities in each phase respectively. The model shows a comprehensive integration of traditional instructional design procedures and conventional software engineering operations.

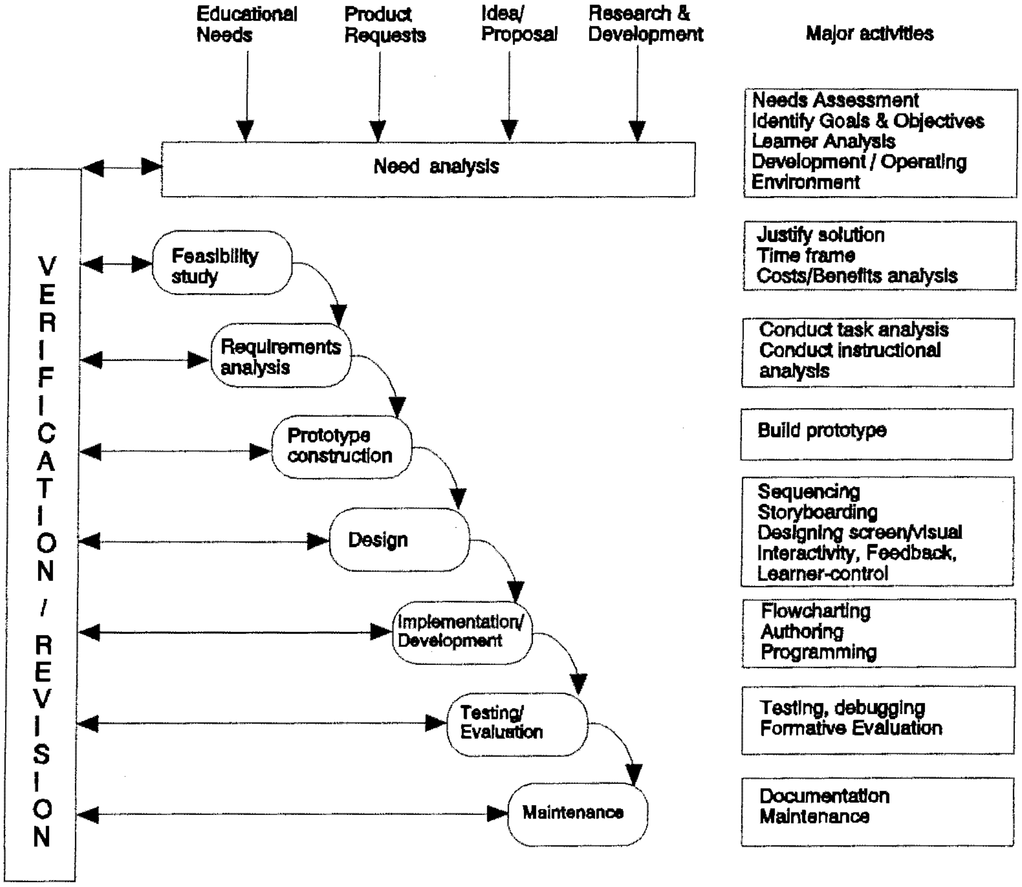


Figure 3.1
Chen & Shen's Life-Cycle Model for Instructional Systems Development
[Chen 1989, p. 11]

In the most recent contributions, Gray & Black [Gray 1994] once again, and also Wong [Wong 1993] advocate quick prototyping of instructional software. Both papers cite the use of hypermedia packages, with their high-technology graphics facilities, for the rapid presentation of key concepts and storyboards. In the work of Gray and his colleagues at the University of Surrey, the prototypes described were built by experienced software developers. Wong, on the other hand, discusses the creation of quick prototypes by non-computing laymen, such as teachers, using an object-oriented approach implemented with fourth generation languages and object-oriented hypermedia. This is the antithesis of the linear structure of the classic SDLC, which introduces systems to teachers only after development, possibly resulting in an inflexible gap between the commissioner's or teacher's expectations and the actual product. Wong suggests that a prototyping life-cycle be incorporated into the overall creation process of CAI, with prototyping by educationalists preceding the building of the final product by software professionals using the classic life-cycle. Figures 3.2 and 3.3 demonstrate Wong's prototyping process and the combined paradigm respectively.

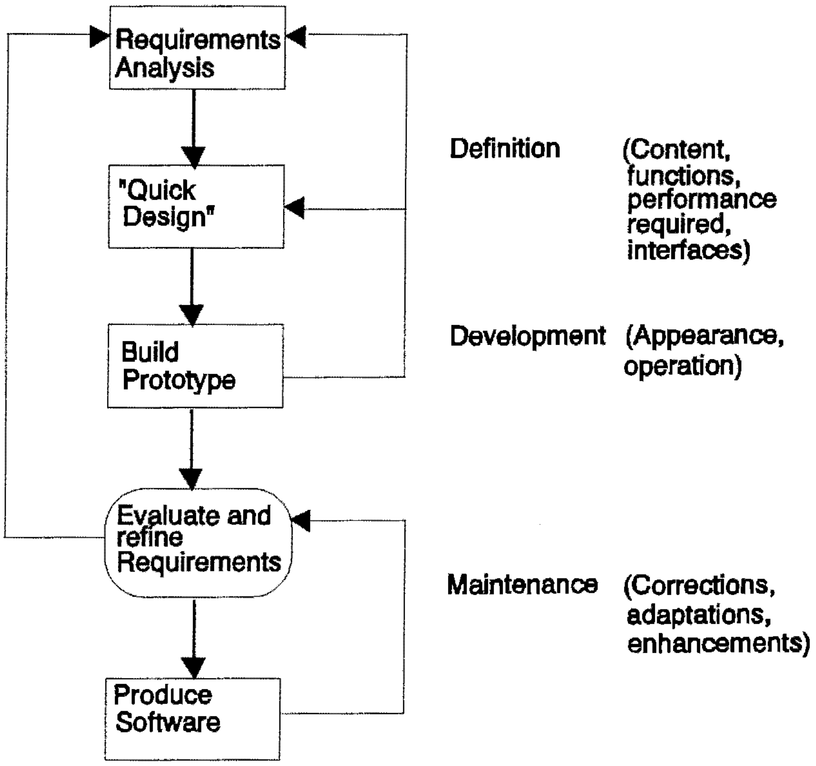

Requirements Analysis
"Quick Design"
Build Prototype
Evaluate and refine Requirements
Produce Software
Definition (Content, functions, performance required, interfaces)
Development (Appearance, operation)
Maintenance (Corrections, adaptations, enhancements)

Figure 3.2 Wong's Prototyping Life-Cycle
[Wong 1993, p. 157]

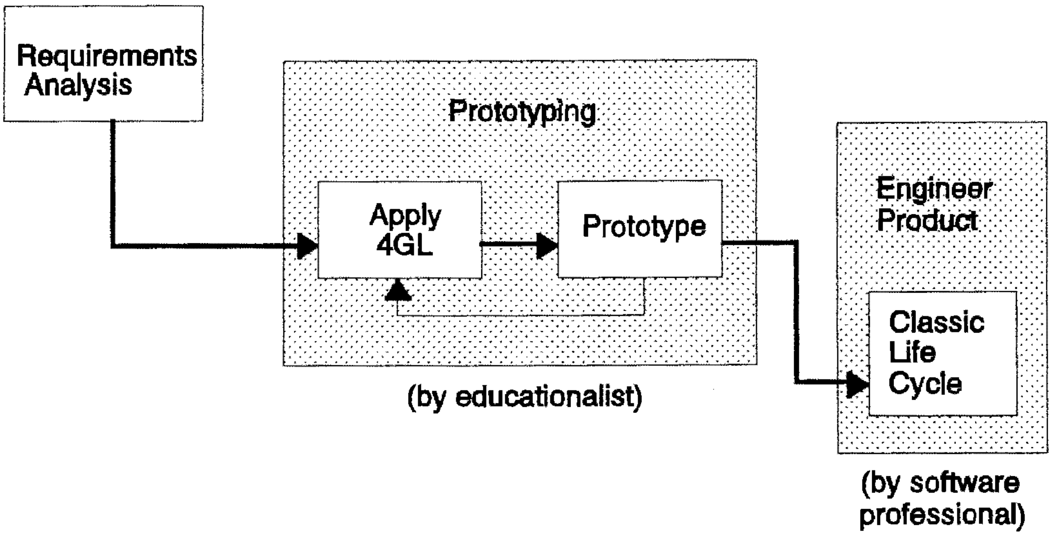


Figure 3.3 The Combined Paradigm
[Wong 1993, p.158]

The present author also investigated the **application of prototyping to CAI** (section 2.5.1), pointing out that latest generation instructional software has many of the characteristics of situations where prototyping is beneficial. The prototype should either be *evolutionary*, i.e. a limited version of the final product which is subsequently developed to full functionality, or else a *throwaway* to refine requirements and test feasibility.

3.5 EFFORTS TO ENSURE QUALITY CONTROL

Along the lines of Naisbitt's [Naisbitt 1982] "*Megatrends*", McLean [McLean 1989] identifies major dimensions along which computing and education may be changing; these serve to identify megatrends affecting educational computing. He suggests several dimensions of change within the **actual development process** of educational software.

From monolithic software to layered software:

The complexity and sophistication of educational software and its environment is on the increase. Such programs cannot be effectively developed by individuals; a team approach with layers of expertise and activities is required. Similarly the process and the product are layered. Within this context, development can be more effective if the principles of software engineering are applied to develop educational software using modularization, documentation and object-oriented design. Portability of software across a variety of operating systems is also desirable.

From testing to quality assurance throughout development:

Current SE practices embody far more than mere testing to identify errors and deviations - quality assurance and continual verification and validation are the norm. In development of courseware too, product quality should be ensured throughout the development cycle. To this end formative evaluation can play a major role.

The matter of quality assurance was also addressed by Christensen & Bodey [Christensen 1990]. To ensure quality courseware, a specific quality assurance audit and evaluation agent was appointed to review each phase of the life cycle, namely: requirements, high-level design, low-level design, coding, testing, and monitoring and maintenance. Thus verification and validation occur throughout the design, development and usage of courseware.

3.6 ADVENT OF AUTHORING SUPPORT ENVIRONMENTS

The original CAI lessons were programmed in conventional programming languages by computer programmers. Subsequently, high level application systems called *authoring systems* enabled non-programmers, usually subject-matter experts or teachers, to create courseware. A more recent trend is the advent of authoring support environments, similar to the Computer Aided Software Environment (CASE) tools of conventional software engineering. An Authoring Support Environment [Kotzé 1995] is tailored explicitly to the information transfer needs of the instructional environment. It is a software tool which supports authors as they translate their subject knowledge and instructional requirements into computer-based instructional material. The facilities it offers support the author as he tackles tasks such as screen design, response analysis and feedback, student data collection and device control.

3.6.1 IDEAL

Ibrahim [Ibrahim 1989] discusses the development of lessons by non-programmers using authoring systems, and the consequent poor quality. In a gesture towards the team methodology, his organization initially used a two-phase approach, separating pedagogical design and coding. Teachers specified lesson behaviour in a detailed script and coders implemented it in a general-purpose modular programming language. To reduce coding effort, a set of re-usable packages was developed to handle aspects such as windows, input devices, message files to separate teaching text from program logic, and pattern-matching algorithms to analyze learner-input.

This led subsequently to an entire development environment, IDEAL (Interactive Development Environment for Assisted Learning), which is both functional and user-friendly. Based on a large screen SUN workstation, and similar to a Computer-Aided Software Engineering (CASE) environment, it incorporates a coherent set of independent tools: script editor, automatic code generator, synchronous multi-window editor, message editor, and a lesson supervisor for network access. IDEAL allows fast prototyping, and facilitates modification.

3.6.2 MCCSE

Approaching the same issue, namely, the lack of reliable high-quality instructional software at reasonable prices, Chen & Chen [Chen 1990] attached partial blame for the situation on the authoring systems available. Hence an intelligent courseware production environment was designed, based both on sound instructional design principles and on software engineering perspectives. With reference to the instructional design aspects, it was necessary first to identify the events of instruction usually included within CAI strategies and the various computer techniques correspondingly used to optimize learning. Gagné's *nine events of instruction* [Gagné 1991; Aronson 1983; section 4.2.3], which support the cognitive processes necessary for learning, are selected as suitable organizing elements for the delivery of CAI. They are:

1. Gain attention,
2. Inform learner of objectives,
3. Stimulate recall of prerequisites,
4. Present stimulus,
5. Provide learning guidance,
6. Elicit performance,
7. Provide feedback,
8. Assess performance, and
9. Enhance retention and transfer.

Computerized techniques were identified to optimize each event of instruction, thus leading to the determination of the required system modes for courseware creation in line with sound principles of instructional design.

With relation to the SE component, functions to attain the desired instructional affects were identified, refined and incorporated into a portable design environment, Minimal Courseware Creation Support Environment (MCCSE). The tools in MCCSE are a system interface, graphic generator, sound generator, frame generator, test generator, statistical generator, sequence organizer, and the template. MCCSE is based on a bottom-up approach, starting with preparation of individual elements, and moving on to the creation of the various formats and frames (e.g. for instruction, testing and feedback). Finally, frames are integrated into sequences according to results of the instructional analyses. In actual learner-usage instruction is individualized by an expert system which determines each user's learning needs and path.

3.6.3 SCALD

Another approach was proposed by Nicholson [Nicholson 1988] who developed an "intelligent" authoring system (which cannot, however, be considered a CASE environment), so that quality educational software can be produced by non-programmers. The Scriptal CAL Designer (SCALD), based upon an expert system and scriptal knowledge representation, allows educator-users to interactively create limited CAL programs for straightforward applications. The script formalism represents knowledge about CAL design and pedagogy, offering a selection of instructional terminology. The standard software development model of:

Specification → Design → Implementation

is once again replaced by:

Prototype → Full script → Specification → Final design

Using a program shell and the user's educational content, automatic code generation then produces the:

→ Implementation.

3.7 APPLICATION TO CAI AND TO FRAMES

Chapter Two concluded with the author's personal application of software engineering methodologies and tools to CAI, and it would be pointless to duplicate the process. Nevertheless, a few aspects are worth highlighting.

Software engineering techniques are clearly relevant to the procurement of sophisticated educational software. The new discipline is termed *courseware engineering*, and it entails a team approach as well as systematic design and development procedures. The comprehensive integration of software engineering principles into instructional software development, as evidenced in the work of Chen and Wong, deserves special mention.

Several life-cycle models have been studied. Section 2.1 introduced the Waterfall Model and the way it is modified by prototyping; two object-oriented life-cycles were described in section 2.3. The models discussed in section 3.4.2 of this chapter were specifically focused on instructional systems development. In general, prototyping of educational software has been rare. However, it is evident that prototyping and evolutionary development are playing an increasing role, and a trend is clearly emerging towards rapid prototyping in the development of CAI. Latest generation CAI is highly visual and interactive; therefore mere paper-based designs or storyboards are inadequate for approval from a commissioning group and for communication between the parties concerned. The situation is exacerbated by the sophistication of the software that can be produced by current technology and by graphical user interfaces. Not only does a prototype clarify requirements and communication of intentions; it also facilitates experimentation with new ideas and alternatives, and allows hands-on testing by end-users. It demonstrates weaknesses in a proposed system and, in line with the principle of constrained design, may result in modification of the original specifications.

Prototyping is even more appropriate to instructional systems development than in general software development, because it provides the flexibility needed in the complex domains of human cognition and human-computer interaction. It can bring very real benefits to the creation of courseware, even in small projects and single lessons. A working model can be used to refine requirements and to ensure satisfactory styles, formats, and functionality before developing the complete and final product, which includes all required content and incorporates robustness and exception handling. A prototyping model should clearly be used in the development of FRAMES.

The use of hypermedia packages was advocated for the rapid presentation of key concepts and storyboards within prototypes. Although these are excellent for assessing the impact of visual displays, they fall short in demonstration of navigation and control.

3.8 CONCLUSION

This chapter overviewed the integration of software engineering principles and methodologies into the development of instructional software, a combination which commenced in the late 1980s. Creation of educational software, also termed courseware engineering, should be characterized both by sound instructional principles and by adherence to the established tenets of the software engineering discipline.

The tendency in ISD has been to emphasize aspects such as learning objectives, instructional activities, judgement and feedback, and target group analysis. Equal attention should be paid to all aspects of the requirements analysis, design and development processes; verification and validation measures should be taken throughout to achieve high quality in the final product and to ensure that it is in harmony with the requirements (or refined requirements!). Development by team, systematic development procedures, modularization and re-use, prototyping life-cycle models, the object-oriented paradigm with its associated modelling tools, and formative evaluation during development are all approaches that can be used to expedite the development process and enhance the quality of the final product.

REFERENCES - CHAPTER THREE

[Aronson 1983] Aronson, D.T. & Briggs, L.J. (1983). Contributions of Gagné and Briggs to a Prescriptive Model of Instruction. In: Reigeluth, C.M. (Ed.), *Instructional-Design Theories and Models: An Overview of their Current Status.* Hillsdale N.J.: Lawrence Erlbaum Associates.

[Black 1987] Black, T.R. (1987). CAL Delivery Selection Criteria and Authoring Systems. *Journal of Computer Assisted Learning 3*, 204-213.

[Black 1988] Black, T.R. (1988). Prototyping CAL Courseware: A Role for Computer-Shy Subject Experts. In: Mathias, H., Rushby, N. & Budgett, R. (Eds), *Aspects of Educational Technology, Vol XXI, Designing New Systems and Technologies for Learning.* London: Kogan Page.

[Black 1989] Black, T.R. & Hinton, T. (1989). Courseware Design Methodology: the Message from Software Engineering. In: Bell, C., Davies, J. & Winders, R. (Eds), *Aspects of Educational and Training Technology, Vol XXII, Promoting Learning.* London: Kogan Page.

[Bowers 1989] Bowers, D. (1989). The Software Design Document: More than a User's Manual. *Educational Technology 29* (12), 15-18.

[Briggs 1991] Briggs, L.J., Gustafson, K.L. & Tillman, M.H. (Eds) (1991). *Instructional Design Principles and Applications.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Chen 1989] Chen, J.W. & Shen, C. (1989). Software Engineering: A New Component for Instructional Software Development. *Educational Technology 29* (9), 9-15.

[Chen 1990] Chen, J.W. & Chen, M. (1990). Towards the Design of an Intelligent Courseware Production System using Software Engineering and Instructional Design Principles. *Journal of Educational Technology Systems 19* (1), 41-52.

[Christensen 1990] Christensen, L.C. & Bodey, M.R. (1990). A Structure for Creating Quality Courseware. *Collegiate Microcomputer 8* (3), 201-209.

[Collis 1987] Collis, B. & Gore, M. (1987). Combining Software Engineering and Instructional Design in a New Type of Course for Educators. *Journal of Research on Computing in Education 20* (2), 104-116.

[Gagné 1991] Gagné, R.M., Wagner, W. & Rojas, A. (1991). Planning and Authoring Computer-Assisted Instruction Lessons. In: Briggs, L.J., Gustafson, K.L. & Tillman, M.H. (Eds), *Instructional Design Principles and Applications.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Gery 1987] Gery, G. (1987). *Making CBT Happen.* Boston: Weingarten.

[Gray 1994] Gray, D.E. & Black, T.R. (1994). Prototyping of Computer-Based Training Materials. *Computers in Education 22* (3), 251-256.

[Ibrahim 1989] Ibrahim, B. (1989). Software Engineering Techniques for Computer-Aided Learning. *Education and Computing 5* (4), 215-222.

[Kotzé 1995] Kotzé, P. (1995). *An Option Space for the Authoring of Interactive Tutoring Systems.* Unpublished DPhil Thesis Proposal, Department of Computer Science, University of York, United Kingdom.

[Lantz ??] Lantz, K.E. (no publication date - possibly 1985). *The Prototyping Methodology.* Englewood Cliffs, N.J.: Prentice-Hall.

[McLean 1989] McLean, R.S. (1989). Megatrends in Computing and Educational Software Development. *Education and Computing 5*, 55-60.

[Naisbitt 1982] Naisbitt, J. (1982). *Megatrends.* New York: Warner Books.

[Nicholson 1988} Nicholson, B.P. (1988). SCALD - Towards an Intelligent Authoring System. In: Self, J.A. (Ed.), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* London: Chapman & Hall.

[Pistorius 1992] Pistorius, M.C., de Villiers, C. & Alexander, P.M. (1992). CAI - Alive and Well at Unisa. *CBE in Tertiary Education, Proceedings of the Third CBE/CBT Conference*, University of South Africa, Pretoria.

[Reigeluth 1983] Reigeluth, C.M. (Ed.) (1983). *Instructional-Design Theories and Models: An Overview of their Current Status.* Hillsdale N.J.: Lawrence Erlbaum Associates.

[Sakasai 1990] Sakasai, Y. & Watanabe, T. (1990). A CAI System for Software Engineers: SOLAS. *Nippon Telegraph and Telephone Review 2* (2), 81-88.

[Self 1985] Self, J. (1985). *Microcomputers in Education: A Critical Evaluation of Educational Software.* Brighton: The Harvester Press.

[Simon 1981] Simon, H.A. (1981). *The Sciences of the Artificial* (2nd ed.). Cambridge, MA: MIT Press.

[Tripp 1990] Tripp, S.D. & Bichelmeyer, B. (1990). Rapid Prototyping: An Alternative Instructional Design Strategy. *Educational Technology, Research and Development 38* (1), 31-44.

[Verhagen 1988] Verhagen, P.W. & Plomp, T. (1988). Educational Technology: A Dutch Contribution to the Debate. In: Mathias, H., Rushby, N. & Budgett, R. (Eds), *Aspects of Educational Technology, Vol XXI, Designing New Systems and Technologies for Learning.* London: Kogan Page.

[Wong 1993] Wong, S. C. (1993). Quick Prototyping of Educational Software: An Object-Oriented Approach. *Journal of Educational Technology Systems 22* (2), 155-172.

CHAPTER FOUR

THEORIES OF COGNITION AND LEARNING

In the previous two chapters CAI was viewed from a software engineering perspective. The focus in Chapters Four to Six is on the second and third pillars of ISD, namely the instructional aspects.

Theories of cognition, once the preserve of psychology, have been enriched in the last three decades due to research by educators and by the Artificial Intelligence (AI) community of Computer Science. Theories of thinking and cognition lead to learning theories, which in turn form the basis of instructional models. This chapter overviews both thinking theories and learning theories, noting the resultant applications to instruction as propounded by psychologists, educationalists and AI specialists such as Newell, Simon and Minsky. The application of selected approaches to CAI is also outlined. Chapter Five is a follow-up which discusses the impact of learning theories on instructional design, since strong associations exist between instructional theory foundations and practical instructional design. Specific details relating cognition and instructional design to CAI and to practice-environments in particular are covered in Chapter Six.

> In relation to the software engineering life-cycle models discussed in section 2.1, the study made in this chapter relates to the requirements analysis phase of FRAMES. The explicit investigation of cognitive strategies tends to be an innovative incorporation in this phase. The selected life-cycle model incorporates a prototype, and thus facilitates evaluation of these strategies.

4.1 THEORIES AND MODELS OF HUMAN THINKING

The components and strategies of human thought and cognition are currently a topic of discussion and research. The logical outflows of **theories and models of human thought** are **theories and models of human learning**, hence their relevance to this dissertation.

4.1.1 SPIRAL MODEL OF THINKING

In the Spiral Model of Thinking Schiever [Schiever 1991] points out how basic cognitive processes such as association and discernment facilitate the five *developmental processes*:

1. Classification - combining similar items,
2. Concept development - identifying an entity (concrete or abstract) as a member or non-member of a class,
3. Deriving principles,
4. Drawing conclusions, and
5. Making generalizations by projecting experience.

Schiever's *development spiral* emphasizes **repetitive encounters** with concepts via experience and instruction. Each re-visitation expands and modifies the five development processes and fosters application of the knowledge to increasingly complex tasks.

4.1.2 PIAGETIAN THEORY

The educational theories of Piaget [Inhelder 1958; Gruber 1977] stress experiential learning and step-wise progress as the basis of intellectual development. Piaget defines four major stages of mental growth, shown in Table 4.1:

Table 4.1 Piaget's Stages of Mental Growth

STAGE	THOUGHT PATTERN	TYPE OF PERCEPTION
Sensori-motor	Develops elementary schemata for external objects	Perception of objects
Pre-operational	Comprehends symbols as representation of external objects	Generalization of object
Operational	Gains ability to do concrete operations	Operational
Pre-adult	Can execute formal operations (e.g. symbolic manipulations)	Operational

Particularly in mathematical education, the needs exist to personalize expertise and to move from the concrete to the abstract. Piaget conceptualized the human intellect as consisting of patterns of thought called *schemata*. When genuine comprehension occurs, it is tantamount to the **re-invention of the theory by the subject**, and such personalized comprehension leads to spontaneous new applications. The teacher becomes less a lesson-giver and more an organiser of situations that lead to curiosity and solution-seeking on the learner's part. The ideal is to provide environments in which exploration leads to self-correction of errors [Gruber 1977; Schiever 1991]. An example is Papert's LOGO microworld [Papert 1980] in which students experiment with graphical geometrical objects, hence discovering concepts for themselves. Papert explicitly states that the theory behind his problem-solving environment draws on the Piagetian approach to thinking and on those aspects of artificial intelligence concerned with thinking about thinking in general. LOGO proponents oppose conventional CAI activities, believing that the use of computers in education entails the global re-design of learning environments [O'Shea 1983; Vockell 1989].

4.1.3 HIGHER ORDER THINKING SKILLS (HOTS)

Vockell & von Deusen [Vockell 1989] sum higher order thinking skillls (HOTS) up as comprising overlapping categories such as metacognition, critical and creative thinking, and thinking skills (or strategies).

They describe Sternberg's tri-archic theory of intelligence which proposes performance components, metacognitive components and knowledge acquisition components. *Performance components* include the skills typically measured in IQ tests; the components involving *metacognition* relate to a learner's ability to plan, evaluate and improve his own cognitive processes (i.e. thinking about thinking); *knowledge acquisition* relates to the processes used for acquiring new information, namely, encoding, combination and comparison.

Intellectual performance improves when learners are capable both of *critical thinking* - reflection on what to believe or do, and *creative thinking* - the development and use of original and flexible ideas. Critical thinking and creative thinking complement each other; the first is primarily evaluative (analytical) and the second primarily generative (focused upon synthesis).

Quality education should set out not only to teach subject content, but also to foster the above processes and skills. An analytical thinker may automatically reason metacognitively, but, in general, some of these skills may be actively taught to the learner. HOTS can be taught in isolation from specific content, but then the learner must generalize and apply them in various settings. Instructional environments and media should explicitly and implicitly facilitate higher-order thinking. HOTS are elaborated in section 4.2.8 in the context of human learning.

4.1.4 THE NEWELL AND SIMON THEORY OF HUMAN INFORMATION PROCESSING

The emergence of the computer following World War II was initially focused on its numerical processing power. In the 1960s its powerful symbol-manipulating capabilities drew attention. The AI community found in the computer new analogies for human cognitive processes. Newell and Simon [Newell 1972] postulated a theory that viewed man as a *human information processing system*, at least, when he is solving problems. They propose that both the operation of the human brain and the computer can be represented by the model of an information processing system shown in Figure 4.1. The *short term memory* (STM) is a component of the processor from which all processes take their inputs and leave their outputs. It has a very small capacity and its information decays quickly. *Long term memory* (LTM) selects certain information from STM for long-term encoding; STM in turn retrieves specific stored data from LTM and combines it with *information input via the receptors* at execution time. In short, there is a series of functions:

- sensation from a stimulus
- perception
- encoding in STM
- association
- encoding in LTM
- retrieval.

In section 4.2.1, this perspective is continued as **learning** is viewed as a human information processing system, and the process illustrated in Figure 4.1.

4.2 THEORIES, MODELS AND ASPECTS OF HUMAN LEARNING

Since the 1980s there has been intensive research in the realm of cognitive development applied to learning. There is a dichotomy over whether knowledge attainment comes from mastering a hierarchy of skills for doing something, or from understanding the underlying reasons. In general, learning of actions occurs as a by-product of performing them, and learning of conceptual material occurs as a by-product of understanding. Current trends emphasize thinking skills and the understanding aspects. Learning theories are related to thinking and understanding, and their major impact is that they form the basis on which methods of instruction are developed.

4.2.1 LEARNING AS HUMAN INFORMATION PROCESSING

This is a logical sequel to the Newell & Simon view (section 4.1.4) which **proposes that thinking is analogous to information processing**. Gagné and Glaser [Gagné 1987] view learning as information processing. Input from external sources is received via human receptors. STM comprises *primary memory* (PM), where information is stored, but restricted in space, and limited in time span to about 20 seconds, and *working memory* (WM), where the actual recognition and "pattern matching" occurs between incoming information and stored information retrieved from LTM. A second function of WM is integration of the new incoming material with existing knowledge structures in LTM, and the third function is rehearsal, the repetition processes by which material in STM can be maintained for longer periods. The structures in LTM are mainly concepts and associations between concepts. Piaget [Inhelder 1958] referred to such associational mental structures as *schemata* or *schemas*. Many theorists assume the concepts to be propositional representations, entities comprising a subject and a predicate [Anderson 1983]. For instance, "16 is-less-than 17" and "a relation is an ordered pair with a first co-ordinate and a second co-ordinate" are propositions. Thus the knowledge representation structures within LTM may be networks of propositions. **Once information has been stored in LTM, it can be considered learned.** Learned material has thus undergone: sensory perception, reception, STM storage, processing in WM, and semantic encoding in LTM. The process is illustrated in Figure 4.1.

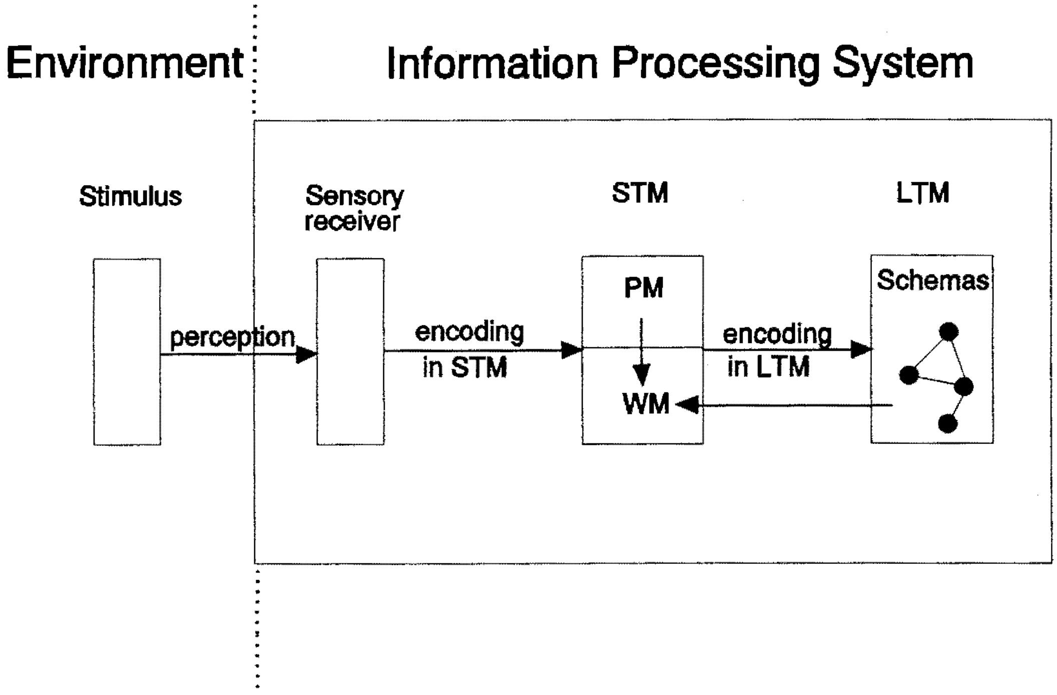


Figure 4.1 General Structure of an Information Processing System

4.2.2 THE GAGNé-BRIGGS MODEL OF LEARNING OUTCOMES

The Gagné-Briggs model of instruction identifies five different types of learning which each require different instructional treatments and different conditions [Aronson 1983; Gagné 1987; Gagné 1991]. Gagné's work is classified as learning theory, but it contributes strongly to instructional theory. The five types of *learning outcomes* are:

1. Verbal information - ability to acquire and recall factual knowledge.
2. Intellectual skills - ability to do mental operations for problem-solving; relating also to abstract concepts. There are five subordinate types:
   - discrimination,
   - concrete concept,
   - defined concept,
   - rule, and
   - problem solving (rule application).
3. Cognitive strategies - ability to plan and control thinking and problem-solving.
4. Motor skills - ability to execute physical movements.
5. Attitudes - predisposition to a positive or negative approach towards a specific object.

The first two outcomes are brought about largely by direct instruction, and the development of the third can be explicitly and implicitly fostered. Instruction must be designed both for initial encoding and to facilitate LTM storage of the specific type of learning. To produce automaticity of skills requires additional instructional strategies.

4.2.3 GAGNé'S EVENTS OF INSTRUCTION

Once a learning outcome has been specified, the necessary instruction must be designed. Gagné proposed learning support based on the nine *events of instruction* [Aronson 1983; Gagné 1987; Gagné 1991]. Each event serves to provide some external conditions of learning:

1. Gaining attention,
2. Informing learner of lesson objectives,
3. Stimulating recall of prior learning,
4. Presenting stimuli with distinctive features,
5. Guiding learning,
6. Eliciting performance,
7. Providing informative feedback,
8. Assessing performance, and
9. Enhancing retention and learning transfer.

CAI designers need to pay attention to all nine events, although not all of them will be applicable to a given lesson. The key factor is to note the effect of the *type of learning outcome* (see section 4.2.2) on the form of the *event of instruction*. In any given situation, the specific forms of an instructional event and its concomitant procedures depend as much on the kind of learning outcome required as on the nature of the event itself.

4.2.4 BEHAVIOURISTIC LEARNING THEORY

Behaviouristic learning theory suggests that learning outcomes are demonstrated by *observable behaviour*. The *stimulus-response* pattern of behaviour is manifested in the learner's overt reactions - a stimulus from the environment results in a response from the learner. According to Skinner [Skinner 1938], a major protagonist of the theory, the correct response should be rewarded with immediate reinforcement, leading to a *stimulus-response-reinforcement* paradigm. The principle of *operant conditioning* states that **if the occurrence of an operant is followed by the presentation of a reinforcing stimulus, the strength is increased** [Skinner 1938]. The initial *stimulus* is typically a question and the *response* is the learner's answer. *Reinforcement*, following the desired behaviour, may be an extrinsic reward or a positive comment. In pure Skinnerian theory no reinforcement is given for an incorrect answer.

The theory was derived initially from experiments with animals, and applied later to human learning. It assumes frequent re-presentation, usually in increasing difficulty levels. Behaviourism views behaviour of an organism as a function of external stimuli; and **learning is viewed as the construction of a set of stimulus-response associations**, induced by repetition and reinforcement. It avoids consideration of internal cognitive processes , dealing rather with **measurable behaviour** [Visser 1995; O'Shea 1983; Venezky 1991], hence the term "behaviourism". This contrasts with the perspective in section 4.2.1, where learning is viewed as the storage of information in long term memory.

4.2.5 MASTERY LEARNING

Closely associated to behaviourism is mastery learning [Regian 1992; Venezky 1991; Vockell 1989], which insists that the learner masters each segment of the curriculum before proceeding to the next. The premise of mastery learning is that, **given enough time, nearly all learners can master objectives**. Skinner believed that the only major difference between learners was their tempo of learning.

Behaviourism and mastery learning are discussed in detail in section 5.3 of Chapter Five.

4.2.6 THE COGNITIVE LEARNING THEORY

Behaviourism focuses on *overt behaviour;* cognitive learning theory focuses on *covert thought processes.* Learning is viewed as the ability to execute internal cognitive processes, such as thought, remembering, conceptualization, classification and problem solving [Visser 1995]. Cognitive scientists compare learning to a processor receiving incoming stimuli (see sections 4.1.4 and 4.2.1), and assimilating them into existing *schemas.* **Learning is considered to be a reorganization of the brain's knowledge structures**, which are considered similar to a semantic network with natural associations as links. The emphasis in cognitive science is on related concepts rather than on isolated facts. Learning theories result in instructional strategies, and cognitive science proposes discovery-learning as one of the most appropriate instructional modes.

As stated in section 4.2.1 on human learning, Anderson [Anderson 1983] considers the units of human knowledge structures to be *propositions.* He claims that human cognition is based on condition-action pairs called *productions.* A production combines a condition proposition and an action proposition. If an element, or a set of elements, matching the *condition* is found in WM, then the production is applicable, and the *action* is triggered. The basic action is to add new elements to WM. For example:

Production rule:

```
IF a relation is reflexive on itself
          and it is symmetric
          and it is transitive
THEN the relation is an equivalence relation.
```

Active elements (propositions) already in WM:

P is reflexive on itself
P is symmetric
P is transitive.

Result: The production is activated, and
a new proposition is stored in WM, namely:
P is an equivalence relation.

4.2.7 CONSTRUCTIVISM

Constructivism is an implementation of cognitive learning theory just as mastery learning is related to behaviourism. It is based upon the tenet of learners *constructing* their own knowledge. Key aspects are *anchoring* of material in appropriate contexts (i.e. *situated learning*), *active learning, collaborative learning, transfer* and *integrated testing.* Current technology offers wide scope for constructivists, but instructional design theory has not kept up [Mehl 1993], and most CAI is behaviouristic. Software with a constructivist bias would actively engage learners in the construction of meaning from information by means of *accretion* (adding new facts), *tuning* (modifying categories according to new information) and *restructuring* (developing structures to interpret and integrate new material) of internal knowledge structures. Rather than passively responding to stimuli, the learner activates his covert mental processes. Mehl & Sinclair [Mehl 1993, p. 13] speculate:

> "If learning ... implies the *construction* of knowledge, will it mean that in future students will learn less *from* the computer, but more *with* the computer?"

In an ideal situation constructivism allows the learner to observe directly and to manipulate a representation, for example, simulations and tool software. In a similar vein Lesgold *et al* [Lesgold 1992] believe in "learning by doing" and recommend *coached practice* and *apprenticeship environments*, which are particularly relevant for situations where an actual work environment can be simulated. They developed SHERLOCK, an intelligent coached practice environment which simulated trouble-shooting in electronic circuits.

Cognitive science and constructivism are discussed in detail in section 5.4 of Chapter Five.

4.2.8 HIGHER ORDER THINKING SKILLS (HOTS)

HOTS, introduced in section 4.1.3, are currently at the forefront in cognition and instruction. The information explosion has led to the realization that it is less important to accumulate knowledge than to acquire problem-solving expertise. HOTS make up a vital subset of human cognitive skills, and fall into overlapping categories [Vockell 1989]:

- Metacognitive skills;
- Critical and creative thinking;
- Thinking processes, e.g. concept formation, principle formation, decision-making, research; and
- Core thinking skills, e.g. the type of classification, analysis and synthesis skills typically measured in IQ tests.

HOTS are important in education at all levels. Without them, tertiary levels and probably academic secondary levels would be unattainable. A competent user of HOTS tends to think at a conceptual rather than a rote memorization level, and to insert information into a network (or schema) instead of treating it in isolation. Vockell & van Deusen [Vockell 1989] give an overview of HOTS, categorizing thinking skills and strategies under the functions, *memory, cognitive discrimination and rules*, and *cognitive control* - see Table 4.2. A single instructional program could not, of course, embody all the different skills.

Table 4.2 Categorization of HOTS According to Function [Vockell 1989, p. 91]

THE BASIS OF PROBLEM-SOLVING		
MEMORY	**COGNITIVE SKILLS: DISCRIMINATION AND RULES**	**COGNITIVE CONTROL STRATEGIES**
♦ Mnemonic systems ♦ Visual association ♦ Whole to part ♦ Self-testing ♦ Creating a context ♦ Personalization ♦ Regrouping ♦ Number of items to remember ♦ Sequence	♦ Higher-order rules ♦ Rules ♦ Defined concepts ♦ Concrete concepts ♦ Discrimination	♦ Simultaneous scanning ♦ Selecting appropriate notation ♦ Identifying multiple solutions ♦ Working backwards ♦ Using a model ♦ Estimating, predicting, projecting ♦ Scanning for clues, hints ♦ Restating the problem ♦ Analyzing ♦ Seeking a pattern or sequence ♦ Brainstorming ♦ Openness to insight, flexibility ♦ Retrieval strategies ♦ Organized fact gathering

Gagné and Glaser [Gagné 1987] believe that the *cognitive strategies* which control other processes of thinking and learning can be generally acquired. The question is whether HOTS should be taught in specific courses on thinking skills, explicitly integrated into content curriculums, or merely implicitly fostered within subject-matter courseware. Proponents of the second view believe that the practice facilitates transfer into other domains. Instructors can actively facilitate the process by pointing out the relationship between those skills and generalized skills, and by ensuring that learners avoid lower-order skills where possible.

4.2.9 ANDERSON'S ACT MODEL

Anderson [Anderson 1983; Anderson 1992] is a proponent of intelligent tutoring (see Chapter Seven, in particular section 7.3.2). His tutors are based upon a cognitive model that is itself capable of solving problems in its domain. Development of such models is complex and time-consuming, and is a foundation of intelligent tutoring. It is not the aim in this dissertation to develop an intelligent tutor with its own domain knowledge, but there are still lessons to be learnt from Anderson's Adaptive Control of Thought (ACT*) Model. Anderson views human cognitive skills as based upon a set of productions (see section 4.2.6), and the computer knowledge-bases in his intelligent tutors are analogously comprised of production rules. Intelligent Tutoring Systems founded on this architecture are the Lisp Tutor, Geometry Tutor and the Three Programming Languages project. The student is given the opportunity to compile his declarative knowledge into production rule format and to practice those rules.

4.2.10 COGNITION AS A FUNCTION OF INSTRUCTIONAL MODE

Comprehension is enhanced or retarded by instructional styles and modes. Some students have fixed learning styles; others prefer different styles at different times. Various modes of presentation [Vockell 1989; Venezky 1991] accommodate the different cognitive styles:

- explanatory approach - direct teaching; explanation of associations and operations,
- demonstration approach - setting out of an application,
- problem-solving - encouraging students to do examples or exercises themselves in:
  (a) multiple-choice mode,
  (b) guided practice mode, or
  (c) independent mode.

The explanatory and demonstration approaches are applicable to the learning phase, and the problem-solving approach to the practice phase. Different types of learning and various modes of presentation are detailed further in sections 6.6.2 and 6.6.3.

4.2.11 TRANSFER

Transfer [Gagné 1985; Vockell 1989] relates to the use of a concept or skill learned in one situation in another setting. In flight-training of pilots, it refers to the application of skills learned in a simulator to a real-life situation. In a mathematical context, transfer occurs when a student learns to think principially, and extends his acquired problem-solving expertise to a more complex example. Students should learn to transfer thinking skills and strategies used in one CAI program to another. True education should be concerned not just with acquisition of content knowledge, but also with the use and generalization of skills in novel situations.

4.3 APPLICATION OF THINKING AND LEARNING THEORIES TO CAI AND TO FRAMES IN PARTICULAR

The goal of learning theory is materials that promote learning. The most relevant concepts are summarized in Table 4.3, and should be used in determining the content of FRAMES.

Table 4.3 Application of Thinking and Learning Theories to Instructional Software

THEORY OR MODEL	APPLICATION
Spiral thinking model (re-visitation)	Access to a topic or exercise in different modes, thus providing multiple encounters; presentation in textual and in graphic mode
Piagetian theory (concrete → abstract)	Synthesis of concrete examples for self-discovery, experiential learning, generalization, application and retention
HOTS: 1. Metacognition	Explicit teaching of higher-order thinking skills User-control of what to do next: ♦ which problem ♦ what aspects ♦ what level
2. Knowledge combination	Explicit links between a problem and relevant definitions
Learning as information processing	Relating a new situation to stored definitions and principles
Limitations on STM	Avoiding overload on learners: ♦ simple directions ♦ chunking of content
Behaviourism and mastery learning	Automaticity in use of sub-skills Stepwise instructions in order of increasing difficulty Branching programs Constructive feedback as reinforcement
Cognitive learning and constructivism	Active use of prior knowledge; anchored instructional contexts; learning by doing
Anderson's ACT* model	Although rules may not be explicitly "encoded", there should be implicit mental formulation
Instructional mode	Provision of options and multiple perspectives
Retention and transfer	Repetition, revision and review

4.4 CONCLUSION

The preserve of learning theories and corresponding instructional models has been enriched by the investigation of a related foundational aspect, namely the realms of thinking and cognition. Theories of cognition lead to learning theories, which in turn result in instructional theories and models. This chapter overviewed various perspectives on human thinking, and, possibly most important, it pinpointed the fundamental difference between the viewpoints of behaviourism and cognitive science. Behaviourism views learning as the construction of a set of stimulus-response associations; the cognitive theory views learning as a reorganization of the brain's knowledge structures. The former has a role to play in instruction, particularly in facilitating the attainment of automaticity in subskills, but the influence of cognitive science, with its associated thinking skillls is currently making inroads into the established behaviouristic educational environment. In addition to theoretical cognitive studies, educationalists are paying attention to pragmatic aspects. Strategies to encourage situated and discovery learning, learning by doing, by active thinking and by planning, should be incorporated into the instructional experience to optimize the learner's cognitive abilities.

In the development of instructional software, it should be an established practice to include a study of approaches to thinking and learning in the requirements analysis phase. Appropriate philosophies should be applied when determining the nature of the content and instructional strategies.

REFERENCES - CHAPTER FOUR

[Anderson 1983] Anderson, J.R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.

[Anderson 1992] Anderson, J.R., Corbett, A.T., Fincham, J.M., Hoffman, D. & Petellier, R. (1992). In: Regian, J.W. & Shute, V.J. (Eds). *Cognitive Approaches to Automated Instruction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Aronson 1983] Aronson, D.T. & Briggs, L.J. (1983). Contribution of Gagné and Briggs to a Prescriptive Model of Instruction. In: Reigeluth, C.M. (Ed.), *Instructional-Design Theories and Models: An Overview of their Current Status.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Gagné 1985] Gagné, R.M. (1985). *The Conditions of Learning.* New York: Holt, Rinehart and Winston.

[Gagné 1987] Gagné, R.M. & Glaser, R. (1987). Foundations in Learning Research. In: Gagné, R.M. (Ed.), *Instructional Technology Foundations.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Gagné 1991] Gagné, R.M., Wager, W. & Rojas, A. (1991). Planning and Authoring Computer-Assisted Instruction Lessons. In: Briggs, L.J., Gustafson, K.L. & Tillman, M.H. (Eds), *Instructional Design Principles and Applications.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Gruber 1977] Gruber, H.E. & Voneche, J.J. (1977). *The Essential Piaget.* New York: Basic Books, Inc. Publishers.

[Inhelder 1958] Inhelder, B. & Piaget, J. (1958). *The Growth of Logical Thinking from Childhood to Adolescence.* New York: Basic Books Inc. Publishers.

[Lesgold 1992] Lesgold, A., Eggan, G., Katz, S. & Rao, G. (1992). In: Regian, J.W. & Shute, V.J. (Eds). *Cognitive Approaches to Automated Instruction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Lippert 1993] Lippert, R.C. (1993). *Computer-Based Education and Training in South Africa.* Pretoria: J.L. van Schaik Publishers.

[Mehl 1993] Mehl, C.M. & Sinclair A.J.L. (1993). Defining a Context for CAI: In Quest of Educational Reality. In: Lippert, R.C. (Ed.), *Computer-Based Education and Training in South Africa.* Pretoria: J.L. van Schaik Publishers.

[Newell 1972] Newell, A. and Simon, H.A. (1972). *Human Problem Solving.* Englewood Cliffs, N.J.: Prentice-Hall Inc.

[O'Shea 1983] O'Shea, T. & Self, J. (1983). *Learning and Teaching with Computers.* Brighton: The Harvester Press.

[Papert 1980] Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas.* New York: Basic Books.

[Regian 1992] Regian, J.W. & Shute, V.J. (1992). *Cognitive Approaches to Automated Instruction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Schiever 1991] Schiever, S.W. (1991). *A Comprehensive Approach to Teaching Thinking.* Boston: Allyn and Bacon.

[Skinner 1938] Skinner, B.F. (1938). *The Behaviour of Organisms: an Experimental Analysis.* New York: Appleton-Century-Crofts.

[Venezky 1991] Venezky, R. & Osin, L. (1991). *The Intelligent Design of Computer-Assisted Instruction.* New York: Longman.

[Visser 1995] Visser, R. (1995). *Rekenaarbenutting in die opleiding van inligtingsvaardige gebruikers aan 'n akademiese inligtingsdiens.* Uncompleted DPhil thesis, Department of Information Science, University of Pretoria, Pretoria.

[Vockell 1989] Vockell, E. & van Deusen, R.M. (1989). *The Computer and Higher-Order Thinking Skills.* Watsonville CA: Mitchell Publishing, Inc.

CHAPTER FIVE

LEARNING THEORIES AND INSTRUCTIONAL DESIGN

The discipline of Instructional Design (ID) is concerned with understanding and improving instruction by applying optimal methods to promote knowledge acquisition and skills in the learner. There are two major thrusts to instructional design. The first is the methodology used towards systemization of instructional models, procedures and materials, ie. its contribution toward the development process. The second is the understanding and promoting of the learning process, using learning theory as the foundation of instructional theory, and thus determining inherent characteristics of the product. Thus a variety of theories and models exist, some founded as explicit prescriptions defining stages and procedures in instruction, and others based on learning theory [Reigeluth 1983; Dijkstra 1990; Wilson 1992]. Figure 1.1 showed ID's role as the central pillar of ISD, contributing, along with instructional theory, to the nature of the deliverable product and, together with software engineering, to the procedures of the development process.

This chapter overviews systematic ID procedures. It re-addresses the two major learning theories, behaviourism and cognitive science, introduced in the previous chapter. Early CAI showed a strong behaviouristic influence, but since the 1980s the impact of cognitive psychology has been evident. Each theory is discussed under corresponding sub-headings, investigating its relationship to learning, its offshoots in terms of instructional paradigms and particularly noting its impact on instructional design and on CAI. It also observes the application and implementation of each, and of a combined approach, in at least one specific instructional implementation.

Instead of devoting a single section at the end of the chapter to "application", as is the usual format, application is distributed over relevant sections.

> In relation to the life-cycle model, the study undertaken in this chapter is relevant to the analysis and design phases of instructional software development.

5.1 FOUNDATIONS OF INSTRUCTIONAL THEORY

Reigeluth [Reigeluth 1983] names the three major components of an instructional theory namely conditions, methods and outcomes. The *conditions* are the basic instructional circumstances, such as the type of learner and the situation. The *methods* relate to the strategies used to achieve different outcomes under different conditions. The *outcomes* are the results of instruction, and must be measurable. Measurement may be in terms of student achievement but may also be an assessment of the appeal of the instruction and the learner's tendency to continue.

Major contributions have been made by the combined work of Gagné and Briggs [Aronson 1983]. Briggs is a writer in the area of *instructional development procedures*, while Gagné is a specialist in *learning theory*, who relates the required objectives and type of learning required to appropriate instructional events [section 4.2.3].

The Gagné-Briggs *model of learning outcomes* [section 4.2.2] emphasizes the teaching of *intellectual skills*. In terms of procedures, it stresses:

1. Identification of prerequisites for each objective.
2. Sequencing, so that prerequisites are taught before superordinate skills.
3. The selection of certain of Gagné's nine events of instruction for teaching each objective, including the specification of media and strategies.

5.2 PROCEDURES OF CONVENTIONAL INSTRUCTIONAL DESIGN

Briggs & Wager [Briggs 1981] believe that systematic design procedures can make instruction more effective, efficient and relevant. The key lies in designing the *objectives*, *content*, *instructional methods*, and *evaluation procedures* in congruence with one another. It is important to distinguish between instructional design models and instructional delivery media. ID as such refers to the process of designing and developing instruction, and need not necessarily specify the media. They specify distinct stages of design:

1. Assessment of needs, goals and priorities.
2. Assessment of resources and constraints.
3. Identification of curriculum, course scope and sequence.
4. Determination of overall structure of courses.
5. Determination of sequence of units and specific objectives.
6. Definition of performance objectives.

7. Preparation of assessments of learner performance
   (tests prepared immediately after definition of objectives).
8. Designing detail content of lessons and materials:
   - a) instructional events
   - b) media
   - c) conditions of learning

   (similarly, done after objectives).
9. Development of media, materials, and activities.
10. Formative evaluation.
11. Field tests, training, summative evaluation, and implementation.

The *systems-approach model* of Dick and Carey is described by Venezky [Venezky 1991]. It bears many similarities to the Briggs and Wager model, but emphasizes:

- required entry-level behaviour,
- development of criterion-referenced tests, and
- revision.

Both the Briggs & Wager model and the Dick & Carey model, particularly the latter with its stress on entry-level skills and criterion testing, imply an implicit behaviouristic flavour.

Merrill & Li [Merrill 1990a] developed an expert system to assist instructional designers. Their *simplified process model* comprises five decisions, each step being directly related to the attainment of the previous step:

1. The *goal* of the instruction.
2. The *content* which best supports the goal.
3. The *modules* necessary to teach the content.
4. Which *instructional transactions*, i.e. strategies and sequences, are the most appropriate for each module.
5. The *elaboration and implementation* of each instructional transaction.

A final methodology is the *Instructional Design Template* (IDT) of West, Farmer and Wolff [West 1991]. More than the previous models considered, it bears witness to the influence of cognitive science. It embodies:

1. *Situational audit* - assessment of the instruction's purpose, the participants' needs, and the required perspective.

2. *Aim and objectives* - includes decisions regarding the topic, the types of learner and learning, and specific objectives specifying what the learner should be able to do under what conditions and at what level; also includes the decision as to whether there will be a pre-test or entry-level analysis.
3. *Content and uses* - specifies the content of the instruction and the use to which the learner will put his knowledge.
4. *Cognitive strategies* - described in section 5.4.2.
5. *Means of instruction* - decisions about methods of learning and media to be used.
6. *Testing* - first decision to be made is whether or not explicit tests will be done. If tests are included they should be relevant and situationally appropriate.
7. *Evaluation* - Formative evaluation and resulting revision of the instructional software is of great importance.

5.3 BEHAVIOURISM AND THE MASTERY LEARNING PARADIGM.

Behaviourism, defined in section 4.2.4, is analysed in this section with reference to its application to learning and instruction, its impact on ID, and its implications for CAI.

5.3.1 BEHAVIOURISM, LEARNING AND INSTRUCTION

Behaviourism in instruction originated in World War II, when psychologists involved in military training identified the outcomes in terms of performance. Behaviourist psychology proposes that learning results from the association of a *stimulus and response* (S-R unit), with *reinforcement* as a third event. It is based on *Skinner's operant conditioning* [Skinner 1938] and entails extensive practice and reinforcement. In the 1960s the process was systematized and formalized as it was integrated into instructional models such as Gagné's. According to Gagné, **learning has occurred when behaviour indicates that a capability has been acquired, i.e. when a set of stimulus-response associations has been constructed** [Aronson 1983; Gropper 1983]. Behaviourism is not concerned with *why* behaviour change occurs, but rather *how* to bring about such changes.

5.3.2 IMPACT OF BEHAVIOURISM ON INSTRUCTIONAL DESIGN

Early ID was strongly influenced by behaviourism; in particular the instructional paradigm of mastery learning bears witness to the kind of behaviouristic principles outlined in this section.

Fleming and Levie [Fleming 1977] investigated the impact of behavioural-science *perception principles* and determined:

- Human information processing capacity is limited; the learner can absorb only a limited amount at a time.
- Perception is organized - humans perceptually construct relationships, groups, objects and events.
- The organization of a stimulus or message influences the speed and accuracy of reception. The simplest perceptual organization is *figure and ground.* Thus spacial and temporal arrangements influence perception of relationship.
- Stimulus figures that are incomplete may be implicitly completed by the perceiver.
- The more familiar a message, the more readily it is perceived, but on the other hand attention is drawn to what is novel and which stands out.
- A moderate degree of uncertainty or anxiety is a strong incentive to resolve a problem (healthy frustration).
- A change in stimulation is necessary for sustained attention.
- The amount processed depends on the number of discrete objects or events, and on the depth to which each is processed.
- At one glance humans can perceive up to about seven items.
- According to their experience, perceivers partition available information into chunks or clusters.
- The more accurately an object or event is perceived, the more feasible and reliable will be further cognitive processing, such as memory and problem solving.

These perception principles have their natural consequences in practical *learning principles* [Fleming 1977]:

- The association of objects or events with each another is facilitated when they are encountered in spatial or temporal contiguity.
- Learning is influenced by the frequency with which stimuli are encountered and the same response made.
- Learning depends on its consequences. When consequences are explicitly or implicitly rewarding, learning is faster and lasts longer.
- The scores obtained by measurement of learning vary as a function of the conditions of testing.
- Familiar or attention-gaining cues facilitate learning.
- The more motivated the learner, the greater can be the size of an instructional unit.
- Spaced practice is more effective than massed practice.
- The more meaningful the relationship between things associated, the greater the learning. Well organized and structured material facilitates knowledge acquisition.
- When the material is introduced at the beginning of a unit, subsequent learning of detail is facilitated.
- Questions inserted within instruction facilitate learning.

- Learner-activity and involvement facilitate learning.
- A wide variety of examples fosters comprehension.
- Examples and non-examples should be shown.
- Prior learning of related concepts facilitates learning.
- Problem solving is helped by situational support which emphasizes crucial elements or reveals an important relationship.
- Situational support for problem solving should present relevant information, but should also provide an opportunity to manipulate various alternatives.

And finally, the two principles most intrinsic to behaviourism:

- The reinforcement that occurs after an act strongly influences the likelihood of that act being repeated in the same context, i.e. the likelihood of a stimulus and response becoming associated is dependent upon consequences.
- In general it is desirable, during initial learning, to reinforce all correct responses. Subsequently, fairly frequent feedback is preferable for consolidation and maintenance of behaviour.

Hannafin & Peck [Hannafin 1988] also derive principles of instructional design using behavioural learning theory (they are clearly a subset of the Fleming philosophy):

1. *Contiguity*: The response should follow the stimulus without delay.
2. *Repetition*: Practice strengthens learning and improves retention.
3. *Feedback and reinforcement*: Knowledge concerning the correctness of the response contributes to learning.
4. *Prompting and fading*: The student should be led to the desired response under decreasingly cued conditions.

Gropper [Gropper 1983] mentions four skills which are significant in behaviourism, namely discrimination, generalization, association, and chaining. The competent performer must be able to *discriminate* between one stimulus and another and produce the appropriate response for each. *Generalization* occurs when students come to recognize analogous stimuli, and can transfer the same skill to new relevant situations. Ability to distinguish between stimuli is not sufficient for mastery performance. The correct response must be *associated* with each stimulus. And finally *chaining* - successful performance in an activity comprises the learning of all S-R units that comprise it and integrating them into a composite chain. The practical application is that an instructional designer should provide instructional and practice opportunities that facilitate acquisition of these four skills.

5.3.3 IMPLICATIONS OF BEHAVIOURISM FOR CAI

It is clear from the features described that most CAI software incorporates the rudiments of behaviourism [Gropper 1983; Poppen 1988]. *Entry-skills analysis* is used to *branch* to a section of the lesson appropriate to the student's level. Programs with a linear structure are sequenced in *steps of increasing difficulty.* A keystone of behaviourism is the provision of *prompts* or *cues* to promote correct responses, with strong cues early in the shaping processes, followed by gradual *fading.* Complex repertoires can be built out of subskills by *shaping* and *chaining.* An important aspect of behaviourism is its goal of *automaticity in subskills*, intended to facilitate the acquisition of composite skills. *Evaluation of student-responses* and the provision of immediate *feedback* are important characteristics, and should supply *reinforcement* for correct answers and corrective or *remedial feedback* for inappropriate or incorrect answers.

A clear example of the implementation of behaviourism is found in *mastery learning* (see section 4.2.5). A student is given as much time as required to achieve a given level (mastery level) in each instructional unit before proceeding to the next. This strategy is particularly effective where the goal is to teach clear and specific performance objectives, but it does not appear significantly to reduce individual differences on standard achievement tests and in general academic performance.

5.3.4 IMPLEMENTATIONS OF BEHAVIOURISM

The programmed instruction of the 1950s and 1960s was based on behaviouristic principles, and so are most current CAI tutorials. Content is chunked, and ordered according to increasing level of difficulty. Text and supporting graphics are placed contiguously; thematic cues are evident; and teaching segments are interspersed with practice opportunities, where the question-answer-judging-feedback process is clearly indicative of the stimulus-response paradigm. Material is introduced at the beginning of a unit and reviewed at the end. Stress is laid on the student's score in the post-test.

Behaviourism is decried by the extreme constructivist viewpoint. The stimulus-response paradigm apparent in much CAI is criticized by Appel & Appel [Appel 1987] in their discussion of a behaviouristic mathematics package. They suggest that, in principle, **computers cannot educate**, and although acknowledging the software to be a sophisticated application of S-R conditioning, they claim that conventional CAI "interactivity" is not dialogue at all. Dialogue, in their view, is open-ended with no preset responses, therefore they do not accept that the giving of correct answers and achievement of learning objectives demonstrate that learning has taken place. It is uncertain whether such learning ensures long-term retention.

5.4 COGNITIVE LEARNING AND CONSTRUCTIVISM

Cognitive Science, defined in section 4.2.6, and constructivism, introduced in section 4.2.7, are analysed in this section with reference to their application to learning and instruction, their impact on ID, and their implications for CAI.

5.4.1 COGNITIVE SCIENCE, LEARNING AND INSTRUCTION

In congruence with the behaviourist stance, important factors in instructional design have been precise objectives, presentation of stimuli, and preparation for anticipated responses, with less concern for the learner's intellectual processing. The emphasis was on performance, although a cognitive perspective did exist side by side with behaviourism. With the advent of the cognitive revolution [West 1991], however, intellectual involvement and active cognitive processing have come to the fore. Cognitive theorists believe that **learning comprises reception, short term storage, encoding, long term storage, and retrieval of information, i.e. a reorganization of the brain's knowledge structures**, as described and illustrated in sections 4.1.4 and 4.2.1. The capacity of STM is limited both in quantity and time-volatility. Encoding occurs as the meaningful information moves from STM to LTM where it is categorized and stored according to its meaning.

In the period pre-dating World War II, Gestalt psychologists propounded the figure-ground approach which stressed ideas such as wholes and patterns within structures. To Piaget [Inhelder 1958] mental growth consisted of the development of logical constructs and *schemata*, which are mental structures used for integrating prior knowledge with new knowledge. There are *state (or data) schemata* and *process schemata.* In line with the analogy between the human mind and the electronic computer, state schemata can be compared to data files and process schemata to programs which are executing.

From the cognitive perspective, perception is considered to be the construction of meaning via integration of the new with the old, within the available schemata, and activated by an event. When a learner can recognize a known concept in a new context, then he is learning cognitively and constructing meaning. Mental schemata do not only facilitate perception and comprehension, but also aid recall.

5.4.2 IMPACT OF COGNITIVE SCIENCE ON INSTRUCTIONAL DESIGN

Hannafin & Peck [Hannafin 1988], derived their first four ID principles (see section 5.3.2) from behavioural learning theory. Based upon cognitive learning theory, they derived their fifth, sixth and seventh principles:

5. *Orientation and recall*: Learning involves the synthesis of prior information, by recalling it from LTM to WM.
6. *Intellectual skills*: Learning is facilitated by the use of existing processes or strategies. By recalling how similar learning objectives were achieved, the student may employ existing methods to learn new information, thus improving learning efficiency.
7. *Individualization*: Learning may be more efficient when the instruction is adapted to the needs and profiles of individual learners.

Jonassen [Jonassen 1988] described major learning strategies, and included techniques other than those solely content-centred and performance-based. He categorizes *material processing strategies* as including recall, integration, organization, and elaboration. *Active study strategies* entail the use of general study systems. Under *metalearning strategies* he includes planning, concentrating, encoding, reviewing, and evaluating. Practical issues such as goal setting, time management, and relaxation are also mentioned. Instructional designers should take cognisance of effective learning strategies, and design so as to facilitate them.

West *et al* [West 1991] proposed nine *cognitive strategies* that can be explicitly incorporated within instruction to foster metacognition and facilitate the active creation of schemata:

1. *Chunking*: Chunking strategies entail rational ordering and classification of knowledge, aiding learners in the intellectual management of large amounts of material or complex processes.
2. *Frames, type 1*: This relates to the presentation of a matrix in which names of concepts, categories and relationships are supplied as column and row headings. Information is presented in the various row-column slots.
3. *Frames, type 2*: These are similar to type 1, but based on the principle that **students** classify the information and supply material for the slots themselves.
4. *Concept mapping*: This is the creation of a visual arrangement to represent the relationships between associated concepts. It provides the 'big picture".
5. *Advance organizer*: This is a brief introduction presented prior to new material. The designer or teacher creates the text of the advance organizer.
6. *Metaphor*, analogy, or simile: This strategy provides a bridge from known information to new knowledge by explicitly suggesting a comparison.
7. *Rehearsal*: This involves revising or reviewing material, asking or answering questions. The learner plays a more active role than the instructional designer, but the designer's task is to provide opportunities.
8. *Imagery*: the mental visualization of objects, events and relationships. Imagery is a great asset in the mental storage of knowledge.
9. *Mnemonics*: These are artificial memory aids, for example, first letter coding or some other verbal string to aid recall.

An instructional approach [West 1991] to Piaget's data schemata and process schemata respectively is to train learners in the methods experts use to *organize knowledge* and the strategies they use to *solve problems.* Cognitive researchers emphasize systematic presentation of all the *declarative knowledge* in a domain and the detailed description of the *procedural knowledge*, i.e. the processes a learner must master.

Instructional designers should incorporate cognitive strategies within instruction. West's techniques can be used to convey content or to activate learning strategies. Whether the strategies are implicitly incorporated or taught as an additional learning outcome alongside content teaching, they should foster metacognition within the learner.

5.4.3 CONSTRUCTIVISM IN LEARNING AND INSTRUCTION

Basic tenets of constructivism

Constructivism, outlined in section 4.2.7, is the belief that learners construct knowledge from experience. It is closely allied to cognitive philosophy. Fundamental principles of constructivist learning [Dick 1991; Merrill 1991] are:

- *Real world*: The problem-solving situation should represent the real world.
- *Constructed learning*: Learning is constructed from experience as the learner builds an internal, personal representation of the knowledge.
- *Active*: Learning is an active rather than a passive experience.
- *Collaborative learning*: Education should foster peer-collaboration to promote multiple perspectives.
- *Situated (or anchored) learning*: Learning should occur in realistic settings. The instruction should be relevant to the context, using learner-control and the facility to manipulate information. In certain situations learning can be anchored in contexts which simulate apprenticeship learning.
- *Integrated testing*: Testing should be integrated into the task instead of being a separate activity.
- *Transfer*: If the above principles are applied the learner should find it easier to transfer the skills to other problem solving situations.

Piaget held a constructivist position [Papert 1988; Pufall 1988], assuming that comprehension and problem solving are based on acting upon and interpreting perception, rather than on passive reception of knowledge. He viewed learning as *assimilation* of basic knowledge and *accommodation* of additional knowledge.

Extreme constructivism

Extreme constructivism argues that **content cannot be prespecified**, and proposes that exploration and discovery-learning are the only type of genuine learning, such as implemented in microworlds. Microworlds are not simulations of reality; they are environments that offer elements and processes to the learner, so that learning-by-exploration can facilitate independent discovery of concepts. They do not prescribe what a learner should learn, rather they set the context within which he can construct knowledge. Papert's Logo and Turtle Graphics [Papert 1980] embody geometric principles, but do not present them conventionally. Children manipulate a "Turtle" with two properties, position and direction, and are intended to discover geometric concepts for themselves, such as the fact that the angles of a triangle total 180^0.

Extreme constructivism **objects to external assessment**, i.e. to explicit testing, of achievement, and does not include:

- specific learning objectives,
- practice and feedback focused on desired outcomes, and
- criterion-referenced assessment to determine whether learners have mastered the desired skill/s.

Disadvantages of constructivism

Pure constructivism argues that assessment should be embedded in the context of learning [Dick, 1991], and that *learning gain* should be measured, rather than whether the student has achieved a certain standard or attained a particular skill. For the constructivist the focus of assessment lies on what has been constructed by the learner as a result of the learning situation. Constructivists tend to be weak on assessment, both regarding *what* to assess and *how* to assess.

A related problem is the apparent lack of concern for entry behaviour of students. In conventional instruction a student is assessed prior to a course to determine what he must know, or be able to do, before commencing. But in constructivism there is no assessment of entry knowledge and skills and no pretest to determine the level of prior knowledge.

5.4.4 IMPLICATIONS OF COGNITIVE SCIENCE AND CONSTRUCTIVISM FOR CAI

Evolution from behaviourism to cognitive science

This sections extends the discussion of the impact of cognitive science on ID in section 5.4.2 by outlining the combined impact of cognitive science and constructivism on ID and CAI. Figure 5.1 [based on West 1991] summarizes the major changes in the way that thinking and learning are viewed in the **evolution from behaviourism to cognitive science.**

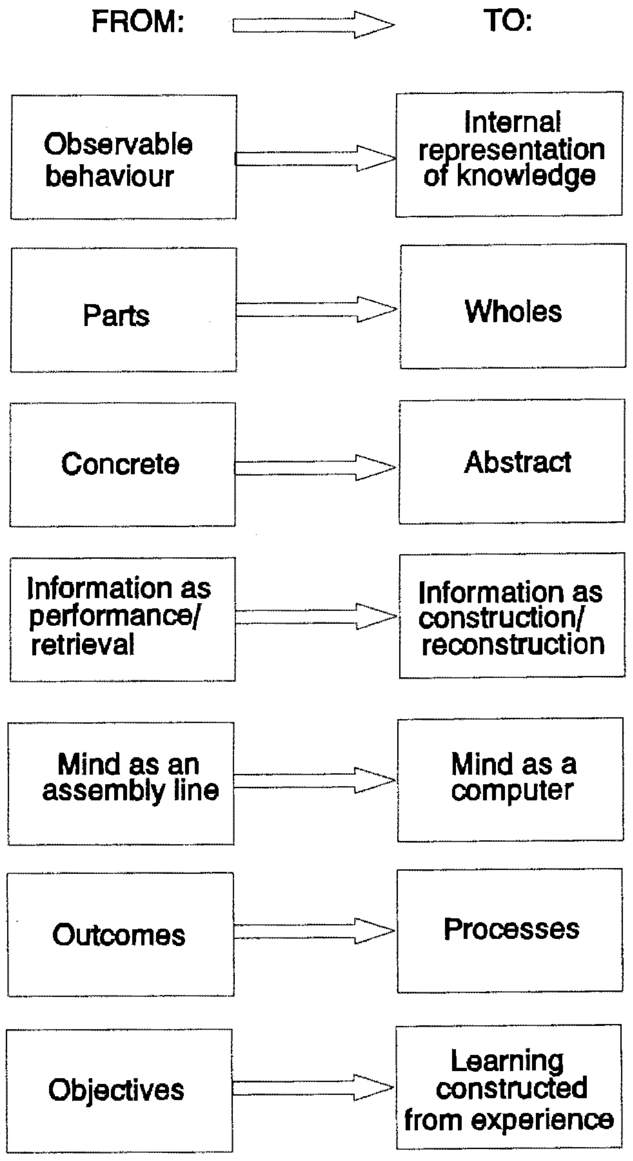

FROM:
TO:
Observable behaviour
Internal representation of knowledge
Parts
Wholes
Concrete
Abstract
Information as performance/ retrieval
Information as construction/ reconstruction
Mind as an assembly line
Mind as a computer
Outcomes
Processes
Objectives
Learning constructed from experience

Figure 5.1 The Major Fronts of the Cognitive Revolution
[Based on West 1991, p. 12]

Cognitive science stresses that the perception, comprehension and assimilation of new or advanced material depends on the way the learner generalizes and relates it to existing internal knowledge. Instructional software designers can facilitate generalization and application of principles to new situations by ensuring that the underlying rules are easily accessible. Instructional strategies should foster movement between the abstract and the concrete.

Constructivism and instructional software

Certain positive tenets of constructivism, such as the *constructed learning* mentioned in Figure 5.1, should be integrated into instructional software where appropriate. *Discovery-learning* is a personal experience which facilitates retention and transfer. *Anchored (situated) instruction* occurs in a context resembling the real-world. Simulations and Intelligent CAI (ICAI), as described in Chapter Seven, may be used to represent reality. Anchored instruction can, however, be effectively applied **without necessarily moving into the preserve of artificial intelligence**. It can be done using strategies that approach a topic from *multiple perspectives* and offer a variety of activities. Bransford *et al* [Bransford 1990] suggest the creation of an *anchor* to *gain attention* and focus the problem. As learners view the anchor-situation from various stances, or in different modes, they experience changes in perception.

Instructional designers are currently being challenged by constructivists to reconsider their Skinnerian theories [Skinner 1938] and behaviouristic practices, to present learners with relevant tasks, and to provide access to tools.

Second generation instructional design

Various authors have examined the potential impact of cognitive science and constructivism on instructional design. Merrill [Merrill 1990b, Merrill 1991] states that first generation instructional design is limited, because:

- it teaches pieces in a linear sequence,
- it does not teach integrated wholes,
- it does not have a component-based teaching approach, and thus does not prescribe the basic subject-matter components necessary to build a complete knowledge base, and
- it fails to integrate the phases of instructional development.

Merrill is developing second generation instructional design (ID_2), with a cognitive rather than a behavioural base. Building on components, ID_2 aims to teach organized and elaborated knowledge, and the relationships between knowledge units in such a way as to facilitate the development of internal *mental models* (similar to Piaget's schemata). Although not intended solely for CAI, it lends itself to computer-based implementation, hence its discussion in this section addressing cognitive science and CAI.

Merrill builds upon **Gagné's different kinds of learning outcomes** and the different kinds of instructional strategies and learning conditions required to promote them. He also, however, considers the implications of **constructivism** for $ID_{2,}$ agreeing that learning is *active and constructive*, but not accepting that interpretation is personal. ID_2 incorporates both *collaborative learning* and individual learning. With regard to the belief that learning is *situated* and *anchored*, Merrill is concerned by the extreme constructivist position that authentic learning should centre around a real-world task, should be only in context, and should incorporate no simplification. Learning from experience only has advantages, but the value of learning from instruction cannot be denied. ID_2 *integrates testing*, but also approves the existence of *separate assessment*.

> A major contribution is Merrill's assumption that **instructional strategy is independent of the knowledge to be taught**. The same strategy can be used to teach different topics and even different subjects, establishing a content-independent instructional strategy.

His expert system for instructional designers (see section 5.2) is based on *content-independent instructional strategies*. Most topics are comprised of *components*. The result of deconstructing a domain into its components is that a *transaction shell* can be used to structure efficient and effective student interaction, drawing on components to help the learner construct a mental model. Each *instructional transaction* is a learner-interaction using a specific component to facilitate the acquisition of a certain kind of knowledge or skill. The key fact is that transaction shells can be used with different content topics, each comprised of components and requiring similar kinds of activities or skills. The designer selects the most appropriate interaction patterns for a given topic, and supplies the subject content to the knowledge base in the required form without defining every specific display or determining a branching structure.

Dick [Dick 1991], investigating constructivism from an ID perspective, suggests a complementary approach, namely *multiple presentation of information*. Complex topics should be covered several times in different ways, so that there are many concrete examples of a concept or process. The best method of presentation for this type of instruction is via hypertext-type control (see Chapter Eight), allowing the learner to select from available resources both what is studied and how it is studied.

5.4.5 IMPLEMENTATION OF COGNITIVE SCIENCE AND CONSTRUCTIVISM

Merrill's Component Display Theory (CDT):

This section overviews two *component-based* theories, Merrill's Component Display Theory and its extension, Merrill's Component Design Theory. Merrill [Merrill 1983] proposed an instructional theory, *Component Display Theory (CDT)*, based upon a set of relationships between the *content* to be taught and the type of *performance* required. The instruction comprises a set of components, and categorizes instructional outcomes on a *two-dimensional matrix* according to content and performance type.

Content categories:
Four content dimensions are indicated - fact, concept, procedure and principle. *Facts* are arbitrary associated pieces of information such as names, dates or events. *Concepts* are groups of objects, events or symbols sharing some common characteristic identified by a similar class name. A *procedure* is an ordered sequence of steps necessary to accomplish some goal or solve a particular class of problem. *Principles* are correlational or cause-and-effect relationships that interpret or predict events or circumstances.

Performance categories:
The three performance levels are remember, use and find. *Remember* is the performance that requires a student to recognize, then reproduce, an item of information. *Use* requires a student to apply some abstraction or approach to a specific test, and *find* requires derivation or synthesis of a new abstraction or an independent walkthrough of a process. Reigeluth [Reigeluth 1983] suggests that the three levels, remember, use and find, correspond to Gagné's three cognitive domains, verbal information, intellectual skills and cognitive strategies, respectively.

CDT is founded on the Gagné-Briggs prescription that there are different kinds of objectives, each requiring unique conditions to promote optimal attainment. Each objective is related to the required content and to the desired performance outcome, and corresponding instructional component thus finds a home in one of the cells on the performance-content grid, which is shown as Figure 5.2. By classifying the components and positioning them on the grid, the instructional designer can determine how adequately he has addressed his objectives, both in terms of the desired type of content and the required performance.

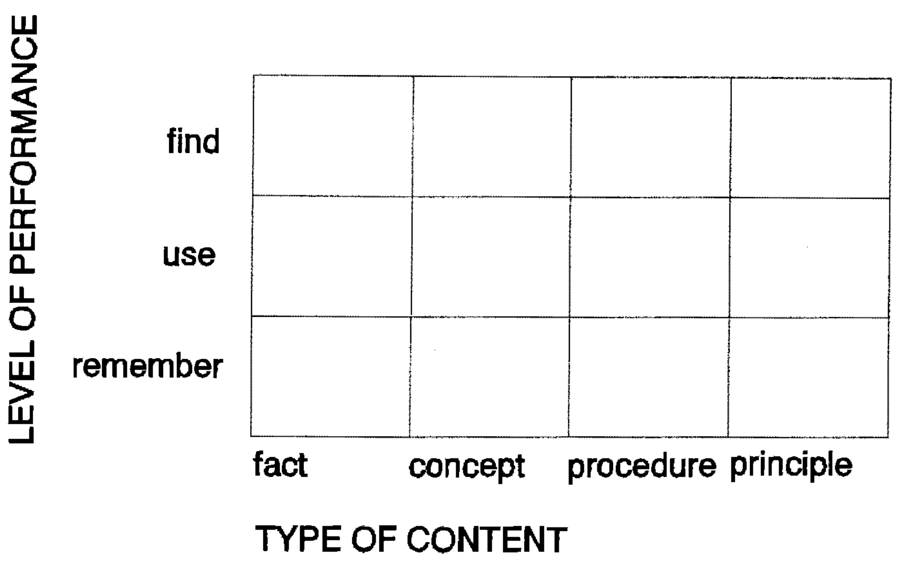


Figure 5.2 Merrill's Performance-Content Grid for CDT
[Merrill 1983, p. 286]

The instruction is designed to cater for individual differences. It stipulates that learner control and selection should permit the student to choose from among the available options, and it explicitly fosters *cognitive processing* within the learner by providing an environment in which he may select both the instructional strategy and the content.

In selecting the *instructional strategy*, i.e. his type of performance, he controls the kind of display, the amount of elaboration, and also the number of examples and practice items.

In selecting *content components*, he tackles the material which is most appropriate at that time.

If all the CDT prescriptions in a given lesson are implemented, the resulting instructional material would be very rich, but it is unlikely that a single student would need all the material available. It is equally probable that in a group or class each of the components would be used by at least some of the students. Thus CDT is strong on *individualization* by accommodating personal learning styles and needs. This approach also teaches the student *learning strategies* which should be of value in general metacognition. CDT's approach of deconstructing a domain into components can be used with a wide variety of subjects and contents, and with virtually any delivery medium. However it is highly compatible with CAI, and two applications of CDT to courseware design are described in section 6.7.

CDT also stresses relationships, termed *constructs*, and identifies five major constructs:

1. *Identity* ("is-a" relationship),
2. *Inclusion* ("is-part-of" or subset relationship),
3. *Intersection* ("and" relationship),
4. The *order* relationship, and
5. The *causal* relationship.

Merrill's Component Design Theory (new CDT)

Merrill [Merrill 1987] extended *Component Display Theory* (CDT) to provide instructional design guidance specifically for courseware authoring in computer-based instructional systems. The latter ID model is termed *Component Design Theory* (new CDT), and is intended to capitalize on the increased capabilities of hardware and software.

New CDT proposes infusion of more "intelligent" material into the traditional branched tutorial CAI, with its inherent text-question-response-feedback cycle, by offering experiential instruction such as expert demonstrations, simulation of suitable subject matter and coaching as the student solves problems. The *experiential model* replaces the displays of original CDT with a variety of active transactions, including demonstration, explanation, prediction, exploration and error detection. A student model and an expert model may be added to achieve intelligent software (see Chapter Seven). The associated *tutorial system* incorporates an advisor function which monitors the student and provides on-line guidance. A constructivist flavour is evident, although there is a significant departure from pure learner control in the form of the guidance, which suggests the content and form which are most appropriate for certain instructional goals. For example, a simple informational component is appropriate for a *remember fact* outcome on the grid, and a completely different experiential component would facilitate a *use principle* outcome.

The new CDT is intended to provide dynamic monitored practice. It should allow the student to demonstrate his learned performance in his selected manner, while being monitored and receiving feedback.

5.5 THE COMBINATION OF COGNITIVE LEARNING THEORIES AND BEHAVIOURISM

This section investigates synergistic combinations of cognitive science and behaviourism, with reference to their application to learning and instruction, their impact on ID, and their implications for CAI.

5.5.1 THE COGNITIVE EVOLUTION

The focus in educational psychology has shifted from behaviourism to cognitive science, with resulting impacts on learning theories and on instructional design. Some researchers refer to the *cognitive revolution*; West *et al* [West 1991] prefer the term *cognitive evolution*, with both models co-existing over an extended period. Rather than viewing the two as being in opposition, many theorists implement the best of both in parallel.

5.5.2 IMPACT OF THE COMBINATION OF BEHAVIOURISM AND COGNITIVE SCIENCE ON INSTRUCTIONAL DESIGN

Hannafin, continuing his ID principles (sections 5.3.2 and 5.4.2), lists those which bridge the behavioural and cognitive approaches:

8. *Academic learning time*: Increasing the time a learner spends actively engaged in profitable instructional activity results in more learning.

9. *Affective considerations*: The attitude of a participant in a learning activity is important to its success.

Larkin and Chabay [Wilson 1992], offering instructional design guidelines for secondary science education, urge both the behavioural and cognitive approaches:

1. Define a detailed description of the processes to be learned.
2. Systematically address all the required knowledge.
3. Conduct instruction primarily through active student involvement.
4. Give feedback as soon as possible after an error.
5. Ensure that students encounter each knowledge unit several times, since revision is essential.
6. Limit demands on the student's attention.

5.5.3 IMPLICATIONS OF THE COMBINED APPROACH FOR CAI

Laridon [Laridon 1989] discusses the combination of mastery learning principles with cognitive theories, and outlines the resultant instructional design features. An experiential base should be provided from which mathematical concepts can be extrapolated. In this respect graphic aids and other visual media can play a major role. The visual elements should be presented and re-presented to gain attention, but should simultaneously lead to the construction of meaning. Laridon also addresses the problem of the limited STM capacity. Here too, graphics can be used to emphasize incoming stimuli so that the images remain longer in STM. Various additional aids can gain attention and enhance retention, such as consistent *icons* and other cognitive learning strategies such as *mnemonics*.

A CAI course in secondary level mathematics was developed initially on mastery principles, but in subsequent redevelopment, use was also made of cognitive instructional design. In line with the models of Dick, Briggs and Gagné which stress the instructional events of *gaining attention* and *presenting the stimulus*, Laridon uses a video sequence to present stimuli. This gains attention and also presents a real-life problem related to the content in line with anchored instruction mentioned in section 5.4.3. The real-life problem re-appears later in order to maintain anchorage and motivation. Cognitive learning theory is used in the assumption that as the perception receives incoming stimuli, it depends on anticipatory frameworks within the mental models. When new elements are encountered, the learner's logico-mathematical experience recalls the relevant schemata to integrate the new knowledge. Laridon applies Piaget's belief that perception of space and time is essential for the development of mathematical concepts.

Section 5.4.5 addressed Merrill's CDT and new CDT for instructional software [Merrill 1983; Merrill 1987]. Although new CDT emphasizes the cognitive domain, it integrates expertise from both the behavioural and cognitive perspectives. It does not explicitly include motivation within its instructional design philosophy, an aspect which is important in the classic Gagné-Briggs model of ID.

5.5.4 IMPLEMENTATIONS OF THE COMBINED APPROACH

Laridon's integration of cognitive theories into a mastery learning course was outlined in the previous section. Two further implementations are considered, the Collins-Stevens Cognitive Theory and Anderson's Programming Tutor.

Collins-Stevens Cognitive Theory of Inquiry Teaching

This is an implementation of the combined approach [Collins 1983] which, although developed from cognitive theory, bears a marked similarity to Gropper's shaping and cuing techniques with respect to strategies. The Collins-Stevens instructional theory is one of learning by discovery in domains with causal relationships. Students are given data from case-studies and learn to induce rules or derive theories, showing how the dependent variable is a function of one or more independent variables. The derived rules are then applied to new cases by the learners. Collins' and Stevens' strategies are:

- Selecting positive and negative (counter) examples.
- Investigating the effect of certain factors by varying them while holding others constant.
- Generating hypothetical cases.
- Forming hypotheses.
- Testing hypotheses, i.e. using students' misconceptions to demonstrate how their errors lead to wrong predictions.
- Considering alternative predictions.
- Tracing consequences to a contradiction.

Anderson *et al*'s ACT Programming Tutor

Anderson's views on cognition are outlined in sections 4.2.6 and 4.2.9, and his tutors are described in section 7.3.2. The three languages programming tutor [Corbett 1993] is a practice environment for students learning to program in LISP, Prolog or Pascal, based on production rules as a model of human cognition. The significant fact is that, despite its cognitive foundation, the tutor contains two features characteristic of behaviourism:

- It explicitly attempts to implement mastery learning. The tutor is divided into sections, each introducing a small set of programming rules. After completing the basic exercises in each section, a student must do remedial exercises until the required criterion score is achieved.
- The tutor includes post-tests to assess the student's ultimate knowledge state.

5.6 FINAL APPLICATION TO CAI AND TO FRAMES

Applications to CAI of the two main learning theories and their combination are described in detail in sections 5.3.3, 5.4.4 and 5.5.3 respectively. This section merely comments firstly on component-based theories, and secondly on the concept of content-independent CAI, implemented by means such as transactional shells.

Merrill's component-based instruction is described in sections 5.4.4 and 5.4.5. Most topics can be **deconstructed into components**, and each component classified according to its type of content and the type of performance it exacts from the learner. The instructional designer would do well to classify his objectives and proposed content according to these categories and position the associated instructional transactions in cells on the performance-content grid. The discipline entailed in working according to this practice would demonstrate gaps in the proposed instruction, thus helping to ensure adequate coverage of the material and conformance to the objectives. CDT suggests a set of principles for more effective instruction, independent of delivery medium, but it is most appropriate for implementation in CAI.

One of the most important implications of the cognitive learning theory, highly appropriate for CAI implementation, is Merrill's belief that there can indeed be **content-independent instructional strategies**, provided that knowledge components and instructional transactions can be identified from the subject matter. A complementary suggestion is Dick's advocation of multiple presentation of information within instructional transactions to enhance learning. It is hoped that the FRAMES software could subsequently lend itself to such content-free extension, while maintaining multiple presentation modes. The structure and layout are sufficiently general to form the basis of other practice environments in related and in non-related fields.

Constructivism has much to offer in today's instructional climate, but holds certain pitfalls, one being that there is no prespecified content. A consequence of this pure constructivist approach is that there cannot be content-independent instructional strategies such as Merrill's instructional transactional shell and CDT.

5.7 CONCLUSION

This chapter has investigated the impact of the behavioural and cognitive science disciplines upon instructional design. Behavioural theory emphasizes performance and observable behaviour, attempting to explain *why* behaviours occur. Cognitive theory sets out to determine *how* learning occurs, based upon internal cognitive processes. Behaviourism stresses the *product* as measured by the learner's performance, and cognitive science the *process* of comprehension. The purist approach to either presents limitations to effective instruction. Rather than viewing them as fields in opposition, the instructional designer should note the evolution from the one to the other, and use features from both, in tandem, to enhance instruction synergistically. Behaviouristic methods can be used to automate subskills; cognitive strategies should be employed to encourage active application of known laws and principles to new situations. An important point relating to cognitively-based instruction is that it should facilitate not only the acquisition of the intended content, but also foster cognitive skills and strategies. In other words, learning strategies should be integrated with courseware.

The stance taken in this research is towards experiential construction of knowledge, but rejecting extreme constructivism. It is clear that *active learning* enhances recall and transfer, and that user-control of both content and strategy increases learner-involvement.

The next chapter considers how the *principles* extracted in this chapter affect the *practice* of instructional design, when specifically applied to CAI. Mehl [Mehl 1993, p. 13] proposes that teaching is concerned with the "*deconstruction* of knowledge", and learning with the "*construction* of knowledge". With reference to the role of the computer in education, he suggests that "in future students will learn less from the computer, but more with the computer".

Merrill's instructional theories using the component-based approach and ID with a cognitive foundation are of great utility. The component philosophy facilitates classification of objectives and promotes organization, both into content units and by varying instructional strategies. Using this approach, each unit can be presented in various ways. CDT can make a valuable contribution to the needs-analysis and design phases of ISD.

REFERENCES - CHAPTER FIVE

[Appel 1987] Appel, M. & Appel, S. (1987). A critique of CAI: the case of SERGO. *South African Journal of Education 7* (4), 278-282.

[Aronson 1983] Aronson, D.T. & Briggs, L.J. (1983). Contributions of Gagné and Briggs to a Prescriptive Model of Instruction. In: Reigeluth, C.M. (Ed.) *Instructional Design Theories and Models: An Overview of their Current Status.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Bransford 1990] Bransford, J.D., Sherwood, R.D., Hasselbring, T. S., Kinzer, C.K. & Williams, S.M. (1990). Anchored Instruction: Why We Need It and How Technology Can Help. In: Nix, D. & Spiro, R. (Eds), *Cognition, Education, Multimedia: Exploring Ideas on High Technology.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Briggs 1981] Briggs, L.J. & Wager, W.W. (1981). *Handbook of Procedures for the Design of Instruction.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Collins 1983] Collins, A. & Stevens, A.L. (1983). A Cognitive Theory of Inquiry Teaching. In: Reigeluth, C.M. (Ed.), *Instructional Design Theories and Models: An Overview of their Current Status.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Corbett 1993] Corbett, A.T., Anderson, J.R. & O'Brien, A.T. (1993). The Predictive Validity of Student Modeling in the ACT Programming Tutor. *Proceedings of AI-ED 93, World Conference on Artificial Intelligence in Education.* Edinburgh: Association for the Advancement of Computing in Education.

[Dick 1991] Dick, W. (1991). An Instructional Designer's View of Constructivism. *Educational Technology 31* (5), 41-44.

[Dijkstra 1990] Dijkstra, S., van Hout Wolters, B.H.A.M. & van der Sijde, P.C. (Eds) (1990). *Research on Instruction: Design and Effects.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Fleming 1978] Fleming, M.L. & Levie, W.H. (1978). *Instructional Message Design: Principles from the Behavioural Sciences.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Forman 1988] Forman, G. & Pufal, P.B. (Eds) (1988). *Constructivism in the Computer Age.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Gropper 1983] Gropper, G.L. (1983). A Behavioral Approach to Instructional Prescription. In: Reigeluth, C.M. (Ed.) *Instructional Design Theories and Models: An Overview of their Current Status.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Hannafin 1988] Hannafin, M.J. & Peck, K.L. (1988). *The Design, Development, and Evaluation of Instructional Software.* New York: MacMillan Publishing Company.

[Inhelder 1958] Inhelder, B. & Piaget, J. (1958). *The Growth of Logical Thinking from Childhood to Adolescence.* New York: Basic Books.

[Jonassen 1988] Jonassen, D.H. (1988). Integrating Learning Strategies into Courseware to Facilitate Deeper Processing. In: Jonassen, D.H. (Ed.), *Instructional Designs for Microcomputer Courseware.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Laridon 1989] Laridon, P.E. (1989). Design Features in Interactive Video Mathematics Lessons. *Proceedings of the First Southern African Conference on Educational Technology*, Human Sciences Research Council, Pretoria.

[Lippert 1993] Lippert, R.C. (Ed.) (1993). *Computer-Based Education and Training in South Africa.* Pretoria: J.L. van Schaik Publishers.

[Mehl 1993] Mehl, M.C. & Sinclair, A.J.L. (1993). Defining a Context for CAI: In Quest of Educational Reality. In: Lippert, R.C. (Ed.), *Computer-Based Education and Training in South Africa.* Pretoria: J.L. van Schaik Publishers.

[Merrill 1983] Merrill, M.D. (1983). Component Display Theory. In: Reigeluth, C.M. (Ed.), *Instructional Design Theories and Models: An Overview of their Current Status.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Merrill 1987] Merrill, M.D. (1987). The New Component Design Theory: Instructional Design for Courseware Authoring. *Instructional Science 16*, 19-34.

[Merrill 1990a] Merrill, M.D. & Li, Z. (1990). An Instructional Design Expert System. In: Dijkstra, S., van Hout Wolters, B.H.A.M. & van der Sijde, P.C. (Eds), *Research on Instruction: Design and Effects.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Merrill 1990b] Merrill, M.D., Li, Z. & Jones, M.K. (1990). Limitations of First Generation Instructional Design. *Educational Technology 30* (1), 7-11.

[Merrill 1991] Merrill, M.D. (1991). Constructivism and Instructional Design. *Educational Technology 31*(5), 45-52.

[Nix 1990] Nix, D. & Spiro, R. (Eds) (1990). *Cognition, Education, Multimedia: Exploring Ideas on High Technology.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Papert 1980] Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas.* New York: Basic Books.

[Papert 1988] Papert, S. (1988). The Conservation of Piaget: The Computer as Grist to the Constructivist Mill. In: Forman, G. & Pufal, P.B. (Eds), *Constructivism in the Computer Age.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Poppen 1988] Poppen, L. & Poppen, R. (1988). The Use of Behavioral Principles in Educational Software. *Educational Technology 28* (2), 37-41.

[Pufall 1988] Pufall, P.B. (1988). Function in Piaget's System: Some Notes for Constructors of Microworlds. In: Forman, G. & Pufal, P.B. (Eds), *Constructivism in the Computer Age*. Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Reigeluth 1983] Reigeluth, C.M. (Ed.) (1983). *Instructional Design Theories and Models: An Overview of their Current Status*. Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Skinner 1938] Skinner, B.F. (1938). *The Behaviour of Organisms: An Experimental Analysis*. New York: Longman.

[Venezky 1991] Venezky, R. & Osin, L. (1991). *The Intelligent Design of Computer-Assisted Instruction*. New York: Longman.

[Vockell 1989] Vockell, E. & van Deusen, R.M. (1989). *The Computer and Higher-Order Thinking Skills*. Watsonville, CA: Mitchell Publishing, Inc.

[West 1991] West, C.K., Farmer, J.A. & Wolff, P.M. (1991). *Instructional Design: Implications from Cognitive Science*. Englewood Cliffs, N.J.: Prentice Hall.

[Wilson 1992] Wilson, B. & Cole, P. (1992). A Review of Cognitive Teaching Models. *Educational Technology Research and Development 39*(4), 47-64.

CHAPTER SIX

INSTRUCTIONAL DESIGN AND COMPUTER-AIDED INSTRUCTION

The previous two chapters overviewed learning theories and their general application to instructional design. This chapter investigates the application of sound conventional instructional design principles to the production of educational software, incorporating the impact of learning and instructional theories. Mehl [Mehl 1993, p. 13] emphasizes the synergism that exists between these pillars of Instructional Systems Development (ISD):

> "The successful application of the computer in education is directly dependent upon *instructional design ingenuity* backed by a solid foundation in *learning theory and learner research.*"

This dissertation, however, incorporates yet a third pillar to the development process, a *software engineering approach*, comprising a design model and a life-cycle model. Application of SE principles to courseware is discussed in Chapters Two and Three. The relationship between the three pillars is set out in Figure 1.1 in Chapter One, and elaborated by Table 6.1, which indicates the role each pillar plays in Instructional Systems Development (ISD), by listing the feature/s on which it impacts.

Table 6.1 The Pillars of CAI

DISCIPLINE	ISD FEATURE ON WHICH IT IMPACTS
Learning and Instructional Theory	The *philosophy* underlying the deliverable product
Instructional Design	Instructional characteristics of the *product* and the instructional development *process*
Software Engineering	SDLC i.e. the software development *process*

This chapter thus commences with an overview of the kinds of CAI and the utility of the computer in instruction, moves on to review generic instructional design for CAI and design features characteristic of instructional software, and then investigates the effect of learning and instructional theory on CAI. The chapter is concluded by showing how instructional design and software engineering (or, more aptly, courseware engineering) methodologies, are combined in the process of developing the intended software.

> In relation to the software engineering life-cycle models discussed in section 2.1, the material in this chapter therefore relates to requirements analysis, but particularly to the design phase of the selected SDLC. The analysis is used to refine the requirements for a CAI practice-environment. The required features are integrated to determine the appropriate design elements.

6.1 KINDS OF CAI

Within instructional computing there are two fundamental roles for the computer - didactic and exploratory. In the *didactic* view the computer is used as a tutor to present instruction or practice opportunities to the learner in a pre-planned, systematic, individualized way. With this approach learning is viewed as acquisition of cognitive structures and procedures. In the *exploratory* view learning is much less systematic. The computer, as a tool, presents phenomena that the student can investigate by inquiry and discovery within the domain. In this approach cognition is seen as an activity situated in a specific context.

There are five basic kinds (or modes) of CAI [Hannafin 1988; Jonassen 1988; Kok 1990; Price 1991]. In the didactic realm there are tutorials, drill-and-practice, simulations, and instructional games. The exploratory variety encompasses the fifth kind of CAI, namely, open learning environments involving the use of the computer as a tool rather than as a tutor.

1. *Tutorial programs* imitate the human tutor, offering one-to-one instruction, and typically teaching well defined instructional objectives. A tutorial engages the learner in an interactive dialogue by presenting information (skills, information, or concepts), providing practice, and then giving feedback to the learner based upon his response to the questions or exercises.

2. *Drill and practice* programs do not actually teach. They follow up on instruction in basic skills by providing extensive practice of sub-skills, often randomly generated. The aim is to facilitate proficiency and fluency and thus to transfer knowledge and expertise from STM to LTM.

3. *Simulations* offer computerized presentations of real-world problems, which require integration and synthesis of the subject matter knowledge. They purport to replicate or emulate the features of some task or context, which in real life is hazardous, costly, or impossible. The learner manipulates parameters, makes decisions and sees the consequences in the simulation. Thus they present the user with experiential enrichment in, for example, aeronautical flight simulators, stock-market operations, chemical laboratory procedures, and history of the past.

4. *Instructional games* are motivational approaches used to reinforce skills and information already taught. Their purpose is partially entertainment, but fundamentally to develop, reinforce and refine aspects of learning.

5. The fifth mode of CAI is an *open learning environment* [Kok 1990] or microworld. The purpose is to allow the learner an independent way of acquiring knowledge by means of problem-solving. Various concepts and elements are offered to the student, so he can manipulate them to learn by exploration.

6.2 WHERE COMPUTER APPLICATIONS ARE SUITABLE

Cumming [Cumming 1990] in a discussion of "learning pull or technology push", emphasizes that, despite the defensible temptation to experiment with the latest hardware and software, the *educational priority* should be the desired learning goal, and that the tools available should be used to achieve it, not the reverse. Course designers should not use computers for the sake of computers. They should determine for which:

- content areas,
- kinds and levels of students (bearing also attitudes in mind), and
- kinds of applications

computers are most effective in developing the selected skill and transferring the required content [Roblyer 1988].

Kontos [Kontos 1985] proposes some answers. He points out that the computer is more effective in raising achievement among high-achievers and low-achievers; it is particularly useful in science and mathematics, and it can change a student's attitude.

An alternative approach [Self 1985; Hannafin 1988] is to investigate the strengths and weaknesses of the computer. The computer is relatively inexpensive. It has the ability to perform accurately and reliably at high speed, to store and manage information, and to perform repetitive tasks without boredom. Computer-use tends to motivate students, and most importantly, it provides interactivity and individualization in a cost-effective manner. It is however unable to respond spontaneously or to teach large groups. The best approach is to use teachers and computers in combination capitalizing on the strengths and minimizing the weaknesses. Thus the computer should be used only in context and to promote learning outcomes for which it is ideally suited.

6.3 HOW THE COMPUTER SHOULD BE USED TO OPTIMIZE LEARNING (CAI STRATEGIES)

Having made the decision to go the computer route in a specific instructional situation, the medium must be used **so as to optimize learning**. In section 6.4 the procedures involved in producing CAI are mentioned, and section 6.5 addresses the general features of CAI and suggests practical methods of capitalizing on the computer's unique capabilities. However, apart from these approaches, research has been done on specific ways of using the computer **to facilitate instruction** [Smith 1984; Self 1985; Price 1991]. In particular, various authors have applied the computer as an instructional delivery medium to Gagné's classic *events of instruction*. Table 6.2, based on the efforts of Smith & Boyce [Smith 1984, p. 9] and Price [Price 1991, p. 92], and augmented by the present author, presents the various CAI techniques that can be used to this end.

The tactics suggested to gain attention (point 1) are less relevant for adult learners, who are usually more self-motivated. Point 2, informing the learner of the objectives, is important in any CAI, to prepare the learner for the kind of instructional experience. The stimulation of prior learning (point 3) can be done either explicitly or implicitly, by making retrieval of background material available on demand. The strategies suggested in points 4 and 5 to present the content and provide guidance can be great value in effective instruction. Graphic representations, portrayal of relationships, good examples, and devices that emphasize key concepts will, if well-designed, enhance and expedite comprehension. Points 6, 7 and 8 relate to the question/answer/feedback process and to assessment mechanisms. Elicitation of performance and supportive feedback are vital in personalizing learning, but formal assessment is not essential in all instructional software. The final aspect, enhancing retention and facilitating transfer, is one of the most vital in effective instruction. If the strategies outlined, such as revision and varied examples, are used in such a way as to help the learner to generalize, retain and project understanding to a wider context, then the instruction has satisfied the ultimate test of its effectiveness.

Table 6.2 CAI Strategies for Instructional Events [Based on Smith 1984, p. 9, Price 1991, p. 92, and adapted by present author]

Event of Instruction	CAI Strategies	Event of Instruction	CAI Strategies
1. Gain attention or engage motivation	♦ Graphics & diagrams ♦ Animation ♦ Games ♦ Sound	(5. continued)	♦ Help facilities ♦ Prompting and fading ♦ Examples and illustrations
2. Inform the learner of the objective	♦ Question ♦ Textual Statement ♦ Graphic Illustration ♦ Interactive demonstration	6. Elicit performance, response or practice	♦ Variety of question types - yes/no, short answer, multiple choice, etc ♦ New problem-solving context
3. Stimulate recall of pre-requisite skills or earlier learning	♦ Questions or pre-test ♦ Textual review ♦ Learner option branching	7. Provide feedback	♦ Review option and help facilities ♦ Correct answer ♦ Reinforcing or remedial feedback
4. Present the stimulus material	♦ Actions that focus learner's attention ♦ Textual - part-whole or static-dynamic display ♦ Graphic - part-whole, colour, animation ♦ Learner-control over sequence and speed	8. Assess performance	♦ Random and/or stratified generation of test items ♦ Variable number of items ♦ Record keeping ♦ Acceptance of synonyms
5. Provide learning guidance	♦ Attention-focusing devices - change in speed, inverse video, highlighting, boxing, flashing, graphics, animation ♦ Relate to wider context	9. Enhance retention and transfer	♦ Revise content ♦ Variety of examples

6.4 CONVENTIONAL INSTRUCTIONAL DESIGN FOR CAI

Procedures of conventional ID, not specifically for CAI, are set out in section 5.2. This section applies the models to CAI. Jonassen [Jonassen 1988] describes the instructional software as the product and the instructional design as the process. The professional activities and processes entailed in producing this "design blueprint" are not fixed; there is a variable sequence of design activities. It is important that it should be based on an instructional theory, such as those discussed in Chapters Four and Five, rather than just on models or principles, or done from a learning perspective. A purpose of this dissertation is also to apply software engineering techniques, in particular a certain life-cycle model, to the design and development of instructional software. However, from the basis of conventional ID, a variety of instructional design theories and models do exist and thus a **generic ID model for CAI production** can be given [Smith 1984; Jonassen 1988; Keller 1988; Morrison 1988: Price 1991]. The basic phases are:

1. Analysis
2. Development
3. Evaluation

Table 6.3 outlines the components of each phase.

Table 6.3 Basic Development Model for CAI

PHASE	COMPONENTS
Analysis (needs assessment)	Subject-matter goals and boundaries Target population Instructional basis
Development	Performance objectives Motivational strategies Instructional strategies Criterion-referenced tests Design of instruction, practice and feedback, and support screens (using storyboards, branching-charts, etc) Programming Revision
Evaluation	Formative (peers, learners) Pilot testing Revision Summative

6.5 DESIGN FEATURES OF CAI

In designing CAI it is important to pay attention to the instructional features unique to or particularly relevant for CAI. An instructional designer's goal should be to **capitalize on the unique capabilities of the computer.** Interactive instructional software differs from general interactive software systems in various respects. A CAI session seldom comprises a fixed schedule of events or sub-procedures. On a macro-level the branching facilities result in an individualized set of activities for each user. Furthermore, on a micro-level, the user-input anticipated does not comprise fixed-format data entries.

De Villiers [De Villiers 1993], listed ten features of CAI programs, particularly general tutorial- and drill-and-practice CAI, and extended the list with an eleventh [Hannafin 1988]:

1. Instructional approach
2. Menus and directions
3. Layout and text
4. Colour
5. Graphics and animation
6. Interactivity
7. Individualization and user control
8. Question/Answer/Feedback process
9. Motivation
10. Scoring and record-keeping
11. Use of complementary and supplementary media

These features are described in the ensuing subsections, along with practical suggestions.

6.5.1 INSTRUCTIONAL APPROACH

Instructional philosophies and strategies are handled in detail in the chapters on learning and instructional theories, along with their impact on instructional design and on CAI.

6.5.2 MENUS AND DIRECTIONS

A variety of menu styles are in common use. It is important to achieve consistency in the menu-type within a single lesson. A menu should not be over-crowded [Price, 1991]; this problem can be solved by sub-dividing it into levels, not all of which should be simultaneously visible on-screen. Directions should incorporate explicit and detailed directions and offer learners only the simplest commands. Certain commands can be entered by means of the function keys. Graphically-based menus, activated by mouse-control, are increasingly common. An adult audience, particularly those who are computer literate, as would be expected of Computer Science students, can be assumed to have a degree of familiarity with windowing packages and consequently may be offered systems involving more complex menu- and direction-techniques. Error messages are on-screen messages that respond to inappropriate or illegal learner-responses, as distinct from feedback to learner-answers. They should appear in a consistent reserved functional area or on a status line. Help facilities should be available - both to clarify the lesson operation and navigation through it, and to elaborate on the content [Keller 1988]. The methods used to access and operate CAI vary from lesson to lesson, therefore clear and concise directions for use should be supplied via a reference manual or card as well as on-screen.

6.5.3 LAYOUT AND TEXT

Screen presentations should have a balanced format, in particular a good balance between text and graphics. The amount of text on the screen should be limited. Ideas should be presented in chunks, with the size of an instructional step appropriate to the learner and to the content. Functional areas should be consistently located. In conventional CAI a *storyboard* or *frame* refers to the content of a single screen of information. The three main types are [Hannafin 1988]:

- *Transitional frames* - which link different aspects of a lesson, providing contextual support, such as menus, directions, or performance reports.
- *Instructional frames* - which present the basic lesson material to the student.
- *Question frames* - frames that offer practice or pose questions (see section 6.5.8).

6.5.4 COLOUR

The colours used should support legibility and be appropriate to the target audience. Colour coding, that is using corresponding colours in two different regions, such as in text and in a diagram to link associated information, can be highly effective. Colour, as well as blocking, should be used to highlight important information or key-words.

6.5.5 GRAPHICS AND ANIMATION

Lanzing & Stanchev [Lanzing 1994] point out the advantages of visualization in courseware engineering to enhance learning. A strong feature of visual presentation is its parallel nature. Text entails a sequential mode of presentation and reception, whereas visuals offer no predefined directionality and items can be viewed simultaneously. Memory research shows that concepts are comprehended from a graphic display far easier than reasoned through without any visual aid. Graphic displays have a particular utility for showing relationships. Soulier [Soulier 1988] presents guidelines for using graphic illustrations with text:

- Keep illustrations appropriate to the audience.
- Place illustrations close to the related text.
- Simple drawings are better than complex pictorials. If complex, diagrams should be built up step-wise.
- Animation should be short, appropriate and learner-controlled.

6.5.6 INTERACTIVITY

A salient feature of CAI is its ability to involve the learner. Content-control and sequence-control are discussed further in the section on individualization. There should be frequent active student-participation, an important form being practice, incorporating learner participation by the doing of examples and the answering of questions. Participation increases attention and facilitates longer learning spans. The interaction should include feedback to the student's response for purposes of remediation and reinforcement (see section 6.5.8). Interaction strengthens cognition and facilitates recall. A variety of interaction devices are available; the mouse, in particular, can be recommended.

6.5.7 INDIVIDUALIZATION AND USER CONTROL

In a *reactive* design the interaction is exclusively program-directed. In a *proactive design* the learner controls both the structure and content of the program. The compromise is a *coactive* interaction where learners do not control the content they view, but do select their own pace and sequence of viewing (possibly also structure and style). Factors that impact upon an individual's learning style are his attitudes, prior knowledge, personal cognitive style, and personality traits [Carrier 1988]. Characteristics that learners can adjust according to personal preference or current needs are the pace, the selection of relevant examples, the amount of practice, and the amount and nature of feedback. An alternative is *learner control with advisement*, i.e. the provision of advice and prompts informing students about the effectiveness of their performance. In general, adult learners can be given more control than juveniles. A common form of personalization is a request for and the subsequent use of the learner's name. This is seldom appreciated by an adult learner. Price [Price 1991] suggests a useful form of individualization - reminding the learner, when appropriate, of a previous response.

6.5.8 QUESTION/ANSWER/FEEDBACK PROCESS

It is important (see section 5.2) to ensure congruence of objectives, instruction and assessment items. In conventional mastery CAI, criterion questions are those that test whether the learner can perform the skills, procedures and tasks specified in the instructional objectives. The student's answers to questions are often used to select individualized instruction to meet the learner's specific needs. It should be clear to the learner how to enter his answer, and the cursor should indicate where. Consistency is important. The response is usually one or several characters, entered via the keyboard or some other device, with completion indicated by pressing <Enter>. The major kinds of questions are true-false and yes-no, multiple-choice, completion or short answer, and open-ended. A learner should be able to edit his answer. The type and quantity of feedback varies according to the kind of learner and stage of learning. It may be a simple "Correct" or "Incorrect", or it may be diagnostic and remedial. During an early learning stage, feedback should focus attention correctly and clarify misconceptions; during the practice phase, it is less important and may arise naturally from the learner's own insight [Hannafin 1988; Vockell 1989; Venezky 1991].

6.5.9 MOTIVATION

Extrinsic motivation includes explicit embellishments or rewards, such as the use of the learner's name, humour, and games. Intrinsic motivation is based on inherent aspects within the instruction that motivate the learner [Alessi 1991; Venezky 1991]. Keller's ARCS model of motivation [Keller 1988] proposed attention, relevance, confidence, and satisfaction as design considerations to create inherent motivation within the learner. Malone, discussed in Alessi & Trollip [Alessi 1991], suggested challenge, curiosity, control, and fantasy as motivating factors. Adult learners usually require less embellishments, and may find extrinsic motivation obtrusive. Designers should not overlook affective considerations, and should bear in mind that success motivates. Satisfactory progress from simpler to more complex topics, with appropriate step sizes, encourages the learner. Students generally have a positive attitude towards CAI, due partly to the fact that it is less threatening and embarrassing when they make errors [Hannafin 1988]. For this reason, some CAI presents no score following question segments, since it is an inherent part of the *instructional* process rather than *assessment*.

6.5.10 SCORING AND RECORD-KEEPING

CAI lessons conventionally keep on-line records, which range from simple score-keeping to elaborate tracking systems [Price 1991]. In a classroom-based network situation, overall results are processed and statistics maintained.

6.5.11 USE OF COMPLEMENTARY AND SUPPLEMENTARY MEDIA.

CAI should seldom be used as a stand-alone educational treatment, but rather as a companion medium to paper-based and/or other media. Instructional software should never be viewed as an adequate substitute for the human teacher! In a distance education situation, where by force of circumstances, students often work in isolation, it is particularly important to use instructional software as a supplement to textbooks and study guides. In particular, as was stated in section 6.5.2, any piece of CAI should be accompanied by written installation instructions and usage guidelines.

6.6 IMPLICATIONS OF LEARNING THEORY FOR INSTRUCTIONAL DESIGN OF COMPUTER-AIDED INSTRUCTION

Chapter Five was an in-depth discussion on the general relationship between learning theories and instructional design. This section relates those learning and instructional theory to the design of CAI, from a base with both behavioural and cognitive foundations. The next section singles out one particularly relevant theory, namely, Merrill's CDT [section 5.4.5], and relates it to CAI design in a section of its own.

6.6.1 FEATURES DESIRABLE TO PROMOTE COGNITION AND LEARNING

Learning theories from a cognitive perspective are related to the instructional design of CAI [Wilson 1992; Lesgold 1992]:

1. *Content:* Teach both domain knowledge and strategic tactics:
   a. *Domain knowledge* is the conceptual and procedural knowledge typically found in text books and other educational materials. It is important but can be insufficient to enable students to approach and solve problems independently.
   b. *Strategic tactics* include:
      - Heuristic strategies - 'rules of thumb" and shortcuts typically developed by experts through repeated problem-solving practice.
      - Control strategies, or metacognition, by which students monitor and regulate their problem-solving activity and facilitate learning.

2. *Situated learning:* Teach knowledge and skills anchored in experience and context of use.

3. *Modelling and explanation:* Show development of a process and explain it. By exposure to both process modeling and the accompanying explanations students can develop the knowledge about how to use their knowledge in problem-solving.

4. *Coaching:* Observe students as they tackle tasks, providing hints and help as needed. Coaching is the closest to one-on-one tutoring. An interactive relationship and a shared load are features of coaching. At present coaching is only fully implemented in resource-intensive intelligent tutoring systems. Yet even in less sophisticated systems a student can learn-by-doing and the computer can aid by monitoring performance and providing on-line help.

5. *Exploration:* Encourage students to try out different strategies and hypotheses and observe their effects, in other words, discovery-learning. In order to achieve transfer, learners must absorb not only the content, but must also develop an ability to solve unfamiliar problems.

6. *Sequence:* Present instruction ordered from simple to complex and increase the diversity of examples, the aims being to activate cognitive strategies within the student and to enhance LTM encoding. West [West 1991] advises designers to use cognitive strategies, firstly, to convey the content itself, secondly, to activate learning strategies known by the student and thirdly, to explicitly teach strategies along with content.

Gagné and Glaser [Gagné 1987], from a platform which synergistically combines behaviourism and cognitive theory, outline factors that play a role in the design of instruction to facilitate encoding and storage in LTM. They view the typical contents of LTM as networks of propositions, and emphasize four aspects of LTM content, which should be born in mind when designing instruction:

1. *Images:* Learners construct internal images of perceived objects and events. Although visual imagery is the most common there are also auditory and tactile images.

2. *The distinction between declarative and procedural knowledge:* Declarative knowledge is knowing *what*, i.e. facts and concepts, whereas procedural knowledge is knowing *how*. With regard to procedural knowledge, they refer to the work of Newell & Simon [Newell 1972] and Anderson [Anderson 1983] who posited the cognitive entity called a *production* to represent an item of procedural knowledge. A production comprises a condition and an action (see section 4.2.6) and is similar to a rule.

3. *Schemata:* Using the cognitive theories of knowledge representation, Gagné & Glaser refer to schemas, scripts and frames. These all fall into the category of schemata which are sets of organized knowledge that represent stereotypes and that facilitate the retrieval of the various concepts and propositions of which they are composed.

4. *Human capabilities:* From a behaviouristic stance it is stated that human learners acquire the capability of exhibiting certain classes of performance, and that not only knowledge, but also capabilities, are stored in and retrieved from LTM.

6.6.2 DESIGN PRACTICES BASED ON LEARNING PRINCIPLES

Association plays a major role in learning. Gagné and Glaser [Gagné 1987] suggest three principles that enhance correct association:

1. *Contiguity:* The learner should experience certain objects or events contiguously, in order to associate them together in the mind. Contiguity can be related to space or time.
2. *The law of effect:* This refers to classic behaviouristic learning. Skinnerian philosophy states an S-R unit is strengthened if it is associated with certain consequences, such as a reinforcing stimulus.
3. *Practice:* Practice is optimally effective when each repetition of the learned association is carried out with contiguity and reinforcement. The instructional designer should provide conditions for practice that personalize rule application and speed up automaticity in productions. From the context of developing higher-order-thinking-skills (HOTS), Vockell & van Deusen [Vockell 1989] also emphasize the importance of guided practice at several stages of instruction.

Designers of instructional software should bear in mind that there are fundamentally four types of learning [West 1991]:

1. *Reception learning*, where the learner reads or observes what is to be learned in its direct form.
2. *Autonomous learning*, where the learner acquires knowledge relatively independently, often from a different perspective.
3. *Guided enquiry learning*, which combines the characteristic of reception and autonomous learning.
4. *Cognitive apprenticeship*, which combines all the features above. It incorporates modeling, coaching, and scaffolding to support the learner as he starts to perform independently. The extent of supervision is lessened (faded) to encourage autonomous learning.

6.6.3 INSTRUCTIONAL MODES FOR CAI

Microworlds and discovery learning environments have been mentioned. This dissertation, however, relates to **planned learning experiences** and thus the instructional delivery modes and strategies mentioned are focused upon the latter approach. Instructional modes can be explanatory, demonstration-based, or presentation of a problem-solving experience [Venezky 1991]:

1. *Explanatory instruction*, which is teaching by direct explanation.

2. *Demonstration instruction*, namely, the application of a rule, principle or process to initiate instruction. Demonstration may be:
   - abbreviated - a solution presented step by step,
   - annotated - each step given and explained, or
   - interactive - each step specified by a student, and executed by a tutor, with or without guidance.

3. *Problem solving instruction*, where problems are presented for application of principles. The fundamental difference between demonstration mode and problem-solving mode is that in the former the instructor carries the main burden of presenting the material and in the latter the student plays the more active role.

Black [Black 1987] also lists alternative computer-assisted delivery modes. The computer may be used in a variety of contexts, as a tutor, as a tool, or in learning management:

- *Presentation* - display of visual text and/or graphics to illustrate a point.
- *Computer-based instruction* - formal didactic learning or drill and practice, both with immediate feedback.
- *Tutoring* - a more flexible approach, less didactic, presenting options, and based upon the learner's needs.
- *Interactive video* - incorporation of CD ROM video disks for dynamic sequences.
- *Record keeping* - of student progress and scores.
- *Simulations* - offering learners the opportunity to input data and see the consequences.
- *Modeling* - use of software packages which allow the learner to create his own model and test it.
- *Data retrieval* - use by a student of, for example, data base software or an expert system.
- *Data capture* - the collection of data from equipment external to the computer, or from the real world.
- *Open-ended problem* - the presentation to the student of a problem having multiple solutions. The student may use software such as word processors or spreadsheets to create a personal solution.

In section 5.4.2 reference was made to West's nine cognitive strategies, namely, *chunking*, *frames* (type 1 and type 2), *concept maps*, *advance organizers*, *metaphors and analogy*, *rehearsal*, *imagery*, and *mnemonics*. In preparing CAI programs for various applications, and in using the different modes, the instructional designer should bear these cognitive strategies in mind and, wherever possible, design modes of instruction that enhance cognitive processing and metacognition within the learner.

6.7 IMPLICATIONS OF CDT FOR INSTRUCTIONAL DESIGN OF COMPUTER-AIDED INSTRUCTION

The previous section applied the learning theories of Chapter Five to Instructional Design of CAI. This section focuses on one particular theory, Merrill's Component Display Theory (CDT) [Merrill 1983], described in section 5.4.5, and pursues its application to instructional software.

Merrill [Merrill 1988] describes two applications of CDT to courseware design. They are the TICCIT authoring system and the Eduware Algebra series. Merrill himself was involved in the design of the former, simultaneous to the development of CDT, but the latter was implemented using CDT independently of Merrill or his associates.

The TICCIT Authoring System

For the purposes of a courseware development system, Merrill subdivides the components into expository presentation forms and inquisitory presentation forms. Both forms have generality and instance variations, an *instance* being a specific illustration of a *generality*.

The *expository forms* impart instruction relating to Merrill's content types:

- definitions of *concepts*
- statements of *principles*, and
- steps and outcomes of *procedures*.

The *inquisitory forms* relate to learner-practice:

- of concepts, by requiring synthesis and classification of examples and non-examples,
- of procedures, by asking the student to explain a given situation.

Student-responses to practice are followed by *feedback*.

Help components give access to attention-focusing information. This help is not intended to be used prior to the student's initial response, and the amount of information provided is faded during the later stages of practice.

The *learner-control* philosophy of TICCIT allows the student to choose not only which content segment, lesson or unit to study, but also which primary (e.g. expository or inquisitory form) or secondary (e.g. help, prerequisites, alternative modes) presentation he would like next. Thus each user selects the combination of displays that is most appropriate for his own aptitude and subject-matter background.

The Algebra series

This series offers tuition and practice opportunities as a supplement to formal mathematics teaching. To maximize personal learning management, each algebra unit is divided into concepts, and each concept presented in four distinct learning styles:

1. *Definition discovery:* discussion about ideas and terminology, in a way that lays a foundation for understanding a concept before presentation of specific problems.

2. *Rules:* display of all the rules relating to that concept.

3. *Examples:* examples available for viewing, as many as required by the student, for rule application or problem solving.

4. *Sample problems:* problems that test the learner's ability to use the information presented in the other modes.

Feedback to student-responses is "Right" or "Wrong". If a student performs unsatisfactorily in the sample problems, he may select another learning style and return to the problem later.

Further features of TICCIT and Algebra.

The type of instructional path through TICCIT and Algebra is the antithesis of the branching programmed-instruction model, where the student plays a passive role in control. Both TICCIT and Algebra have a *map structure* from which the user selects components. In Algebra the student chooses a lesson, then within the lesson, a segment, and once in the segment he chooses his own learning mode. The student himself is responsible for quantity of instructional material he uses to support his problem-solving effort.

Referring to the term *frame*, which in CAI, is considered synonymous with a screen display, Merrill recommends the alternative terminology, *instructional transaction* (also used in section 5.4.4). A transaction, requiring considerable control and processing by the user, is a more active concept than a frame. Each of Merrill's instructional transactions occupies the entire screen, so that only one component appears at a time.

6.8 APPLICATION TO CAI AND TO FRAMES IN PARTICULAR

The entire focus of this chapter is application to CAI. Nevertheless, in line with the format of other chapters, the topics considered in the chapter are applied to CAI and to the envisaged FRAMES practice-environment in particular. Sections 6.1 through to 6.7 are applied to FRAMES in sections 6.8.1 to 6.8.7 respectively.

6.8.1 KIND OF CAI REQUIRED

The intention is to produce a *practice-environment* to supplement written material. Although it may incorporate aspects of tutorials, drill-and-practice and open learning environments, it should not bear close resemblance to any of them. The aim is not to duplicate written tuition, but to present **extensive and intensive practice opportunities** in subskills and composite skills, and also to present other visual aids and learning-events to increase familiarity with the domain in general and with the realm of each problem.

6.8.2 WHETHER OR NOT A COMPUTER APPLICATION IS SUITABLE

The priority is the educational goal, and appropriate tools and media are to be selected to achieve it. For a distance-education student working in isolation on a mathematical module, computer-based practice can provide valuable individualization and feedback, particularly when the software can be used as desired in terms of quantity, style and type of exercise.

6.8.3 HOW THE COMPUTER SHOULD BE USED, I.E. CAI STRATEGIES

Table 6.2 lists CAI strategies for various instructional events. Strategies appropriate for a practice environment include *graphics* to engage attention and clarify descriptions, definitions and rules, *review* of prior learning, *learner-control* over content, quantity and sequence, and *attention-focusing devices* (eg. arrows, boxing) to emphasize relationships and salient points. *Help facilities*, *worked-examples*, and *feedback* to user-responses all play a role in providing guidance. Practice can be stimulated and encouraged by providing *variety of types and modes* to cater for individual learning styles and for the same learner's needs at different times. Finally, the examples presented should be widely varied to enhance *retention and transfer.*

6.8.4 THE DESIGN PROCESS

In the current climate of sophisticated technology and powerful development software it is important that the design and development process should be in line with state-of-the-art software engineering techniques, and not just instructional design procedures, important though they may be.

6.8.5 DESIGN FEATURES

Attention should be paid to the basic design features relevant to CAI:

- *Colour* - good selections and combinations are desirable to enhance aesthetics and facilitate perception.
- *Directions* - must be clear, concise and consistent.
- *Graphics* - use visual illustrations to re-present concepts.
- *Interactivity* - user-participation is the mechanism that promotes user-engagement.
- *Individualization* - in a practice environment it is essential that the system behaves as a tool presenting each learner with the opportunity to meet his personal needs.
- *Motivation* - for tertiary-level students, this should be intrinsic, obtained by a sense of progress. The system should support self-motivation by not frustrating the user.

6.8.6 APPLICATION OF LEARNING THEORY

Learning theories give birth to instructional theories, which in turn influence the modes and strategies used in the design of instruction. The concepts and techniques advocated by various authors in section 6.6 have been considered specifically with reference to a CAI practice-environment. The following relevant strategies have been extracted or derived:

- *Metacognition* - learners should be able to monitor and control their own problem-solving activity.
- *Heuristics* - ways of relating principles directly to the problem on hand. In each problem solving situation, a learner should ask himself, "What do the rules say?".
- *Development of a process* - should be demonstrated step by step.
- *Explanations* - should be used for elaboration.
- *Real-world perception* - if a learner has an accurate perception of a domain and its elements, it facilitates the answering of principial and generic questions.
- *Images* - graphic aids and the synthesis of examples promote construction of internal images.
- *Schemata* - the construction of schemata for each concept should be fostered, ie. the organization of knowledge according to relationships such as, for example, mind-maps or sets of " if ... then ... " rules.
- *Spatial contiguity* - can be implemented, for example, by pull-down elaboration and definitions viewed simultaneously with problem-solving.
- *Different kinds of practice* - by combining features from West's four kinds of learning and Black's and Venezky's instructional delivery modes, three fundamental learning/practice situations can be classified:

1. Reception: Learner absorbs information from an explanation or demonstration where tutor plays the active role.
2. Guided practice: Coaching or guided enquiry, where tutor leads or supports learner through an exercise.
3. Autonomous problem-solving: Student plays active role, may tackle open-ended problems.

6.8.7 APPLICATION OF CDT

The component-based model is most appropriate for CAI, and appears to be an ideal approach for the FRAMES practice environment. Simultaneous use of several components is required, however, so FRAMES screens should present a window-based appearance, rather than single-transaction displays.

6.9 IMPLICATIONS FOR THE ANALYSIS AND DESIGN PROCESS OF CAI AND FOR FRAMES IN PARTICULAR

6.9.1 REQUIREMENTS ANALYSIS

Section 9.2 of Chapter Nine addresses the overall requirements analysis phase of FRAMES, incorporating all aspects - philosophical and pragmatic. This section merely sets out certain required features of a practice environment discussed in this chapter.

In the preceding six sub-sections certain features have emerged as candidates for incorporation in a CAI practice environment. Figure 6.1 sets them out indicating the purpose each would serve. Outward arrows indicate desired features and requirements that must be provided in a practice environment. Inward arrows show the role that should be played by various factors in the context of instructional software.

To satisfy all these requirements an extensive and varied set of instructional components is required. To work through the entire set would be prohibitively lengthy and would entail covering material irrelevant or not required by the learner at that time. The implication for the design of FRAMES is a **component-based system, in which a set of options should be presented to the learner.** User-control should allow the learner to select instructional transactions as advocated by Merrill in CDT and new CDT (see section 5.4.5).

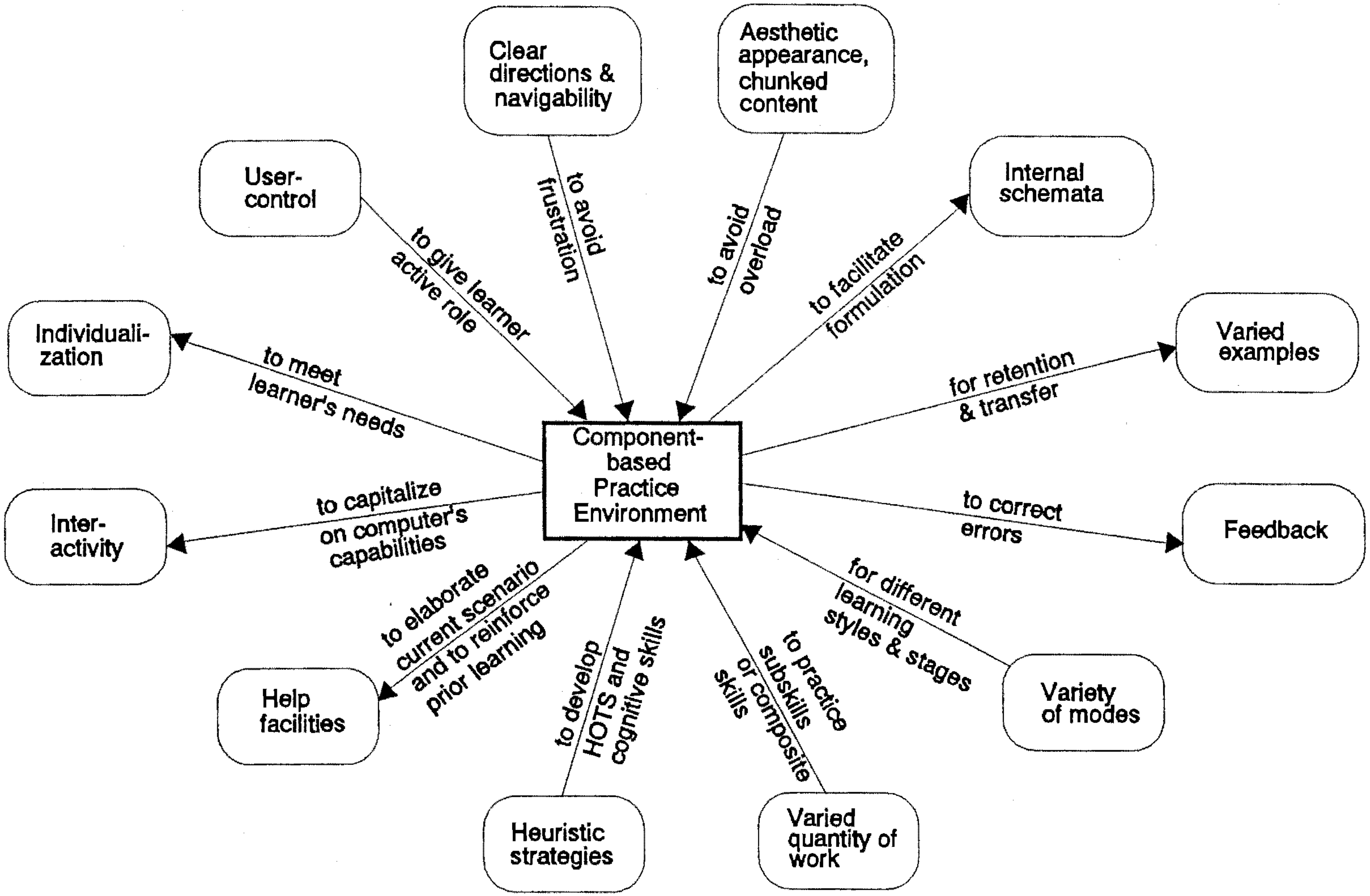

Component-based Practice Environment
Clear directions & navigability
to avoid frustration
Aesthetic appearance, chunked content
to avoid overload
User-control
to give learner active role
Internal schemata
to facilitate formulation
Individualization
to meet learner's needs
Varied examples
for retention & transfer
Inter-activity
to capitalize on computer's capabilities
Feedback
to correct errors
Help facilities
to elaborate current scenario and to reinforce prior learning
Variety of modes
for different learning styles & stages
Heuristic strategies
to develop HOTS and cognitive skills
Varied quantity of work
to practice subskills or composite skills

Figure 6.1 The emergent Characteristics of a Practice Environment

6.9.2 DESIGN

A comprehensive study of the design phase of FRAMES is found in section 9.3 of Chapter Nine. The top levels of a typical piece of software are represented by a *structure chart* as shown in Figure 6.2. The chart is adapted to represent instructional software rather than conventional software.

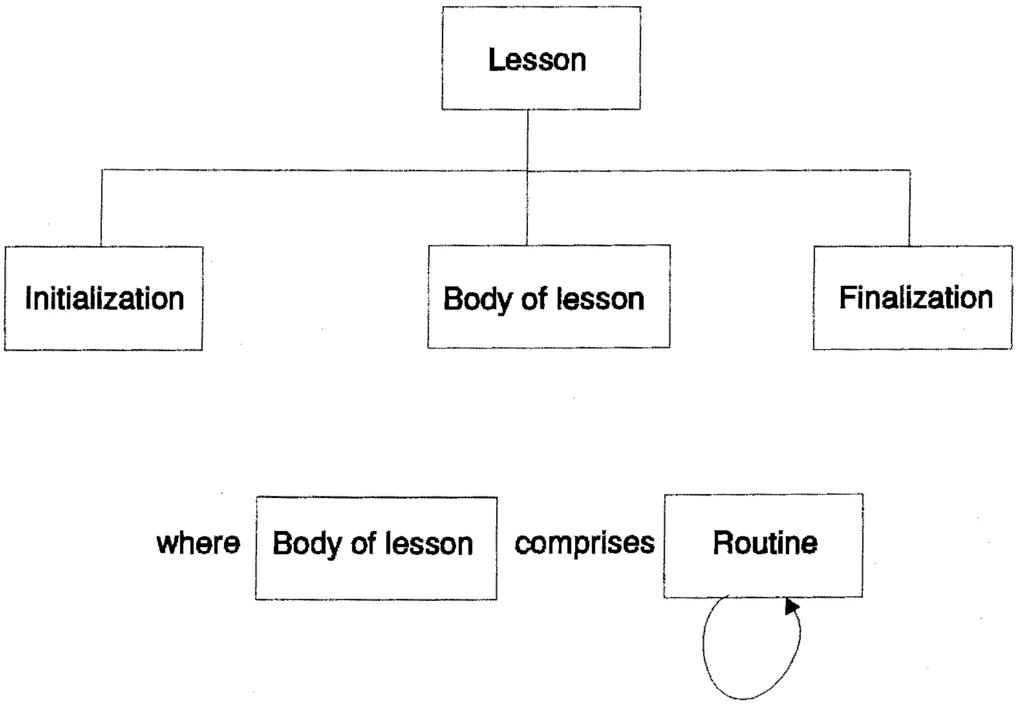


Figure 6.2 Structure Chart to represent Highest Levels of Instructional Software

Initialization would comprise the introductory orientation screens and main menu.

The *routines* represent the instructional and practice components. In tutorial CAI they would consist of teaching segments followed by question-response-feedback segments. In the kind of practice-environment envisaged there would be no formal didactic components, the emphasis being on exercises in various modes - either worked model-answers or proofs to be tackled by the student with varying degrees of assistance. The extent of the exercise should be variable, so as to allow the learner either to practice single components of a problem or to choose a more extensive proof, comprising composite skills. The instructional transactions should also include components aimed at increasing the learner's intrinsic grasp of the domain, such as visuals and concrete examples.

Figure 9.4 extends the general structure chart of Figure 6.2 to a particular design for FRAMES, showing its specific components and options presented to the learner. Some routines would require active involvement from the user; others entail more passive observation. Table 6.4 lists the active and the passive processes.

Table 6.4 Active and Passive Processes in a Practice Environment

ACTIVE PROCESSES	PASSIVE PROCESSES
Selection of a component	Reading a definition
Request for help	Viewing an illustration or graphic aid
Development of automaticity in sub-skills	Studying a step-by-step problem-solving demonstration
Handling of a composite proof or problem	
Synthesizing concrete examples of an abstract problem	
Relating an example to the appropriate theory	

6.10 CONCLUSION

In conventional ISD, instructional design determines characteristics *both of the process and of the product.* The premise in the introduction of Chapter One states that instructional theory and instructional design are used to define the requirements and characteristics of *the product*, and software engineering methodologies are combined with instructional design in *the process* of development. Figure 1.1 is repeated as Figure 6.3 to reinforce the statement.

This chapter stresses the role of ID as the central pillar of instructional systems development, which straddles the dividing line between the process and the product, i.e. it relates both to the desired characteristics of the end product, and to the tasks and procedures used in creating the desired software. With regard to the development *process*, the procedures of conventional ID models are described in sections 5.2 and 6.4. To an extent, these are subsumed by, or incorporated within, the software engineering life-cycle approach of this dissertation. With relation to the deliverable *product*, it is important that a designer should not overlook the inclusion of valuable instructional and cognitive features, over and above the actual content.

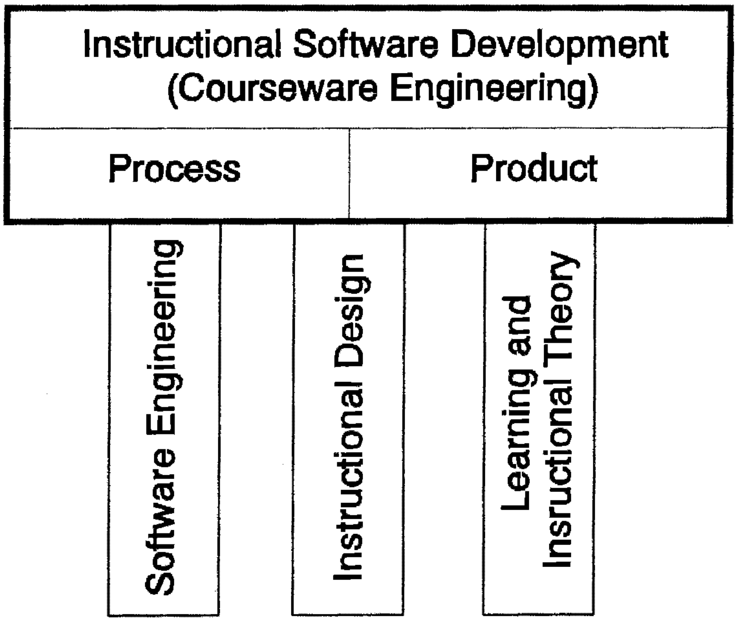


Figure 6.3
The roles of Software Engineering, Instructional Design and Instructional Theory as pillars of Instructional Systems Development

The practical outworkings of learning theories and instructional models reviewed in the previous chapters have been applied to the requirements of the prototype. Appropriate requirements for the FRAMES practice environment have emerged, and the conceptual design of the instructional transactions has been determined.

Integration of CDT and CAI could be synergistic, producing a component-based environment conducive to primary instructional presentation, as well as promoting active mental processing and practice on the part of the learner.

REFERENCES - CHAPTER SIX

[Alessi 1991] Alessi, S.M. & Trollip, S.R. (1991). *Computer-Based Instruction: Methods and Development.* Englewood Cliffs, N.J.: Prentice Hall.

[Anderson 1983] Anderson, J.R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.

[Black 1987] Black, T. R. (1987). CAL Delivery Selection Criteria and Authoring Systems. *Journal of Computer-Assisted Learning 3,* 204-213.

[Carrier 1988] Carrier, C.A. & Jonassen, D.H. (1988). Adapting Courseware to Accommodate Individual Differences. In: Jonassen, D.H. (Ed.), *Instructional Designs for Microcomputer Courseware.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Cumming 1990] Cumming, G. (1990). Artificial Intelligence and Images of Natural Learning. In: McDougall, A. & Dowling, C. (Eds), *Computers in Education.* Amsterdam: Elsevier Science Publishers B.V. (North Holland).

[De Villiers 1993] de Villiers, M.R. (1993). *Relations: A CAI Tutorial in Theoretical Computer Science.* Unpublished MEd thesis, University of Pretoria.

[Gagné 1987] Gagné, R.M. & Glaser, R. (1987). Foundations in Learning Research. In: Gagné, R.M. (Ed.), *Instructional Technology: Foundations.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Hannafin 1988] Hannafin, M.J. & Peck, K.L. (1988). *The Design, Development, and Evaluation of Instructional Software.* New York: Macmillan.

[Jonassen 1988] Jonassen, D.H. (Ed.) (1988). *Instructional Designs for Microcomputer Courseware.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Keller 1988] Keller, J.M. & Suzuki, K. (1988). Use of the ARCS Motivational model in Courseware Design. In: Jonassen, D.H. (Ed.), *Instructional Designs for Microcomputer Courseware.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Kok 1990] Kok, W. & Poorthuis, G. (1990). The Effects of Different Teaching Strategies in Three CAI Programs within the Same Content Area. In: Pieters, J.M., Simons, P.R.J. & de Leeuw, L. (Eds), *Research on Computer-Based Instruction.* Amsterdam: Swets and Leitzinger B.V.

[Kontos 1985] Kontos, G. (1985). Instructional Computing: In Search of Better Methods for the Production of CAI Lessons. **Computer Education** 49, 16-19.

[Lanzing 1994] Lanzing, J.W.A. & Stanchev, I. (1994). Visual aspects of Courseware Engineering. *Journal of Computer Assisted Learning 10,* 69-80.

[Lesgold 1992] Lesgold, A., Eggan, G., Katz, S., & Rao, G. (1992). Possibilities for Assessment using Computer-Based Apprenticeship Environments. In: Regian, J.W. & Shute, V.J. (Eds). *Cognitive Approaches to Automated Instruction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Lippert 1993] Lippert, R.C. (Ed.) (1993). *Computer-Based Education and Training in South Africa.* Pretoria: J.L. van Schaik Publishers.

[Mehl 1993] Mehl, M.C. & Sinclair, A.J.L. (1993). Defining a Context for CAI: In Quest of Educational Reality. In: Lippert, R.C. (Ed.), *Computer-Based Education and Training in South Africa.* Pretoria: J.L. van Schaik Publishers.

[Merrill 1983] Merrill, M.D. (1983). Component Display Theory. In: Reigeluth, C.M. (Ed.), *Instructional Design Theories and Models: An Overview of their Current Status.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Merrill 1988] Merrill, M.D. (1988). Applying Component Display Theory to the Design of Courseware. In: Jonassen, D.H. (Ed.), *Instructional Designs for Microcomputer Courseware.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Morrison 1988] Morrison, G.R. & Ross, S.M. (1988). A Four-Stage Model for Planning Computer-Based Instruction. *Journal of Instructional Development 11* (1) 6-14.

[Newell 1972] Newell, A. & Simon, H.A. (1972). *Human Problem Solving.* Englewod Cliffs, N.J.: Prentice-Hall Inc.

[Price 1991] Price, R.V. (1991). *Computer-Aided Instruction: A Guide for Authors.* Pacific Grove, CA: Brooks/Cole.

[Regian 1992] Regian, J.W. & Shute, V.J. (Eds) (1992). *Cognitive Approaches to Automated Instruction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Roblyer 1988] Roblyer, M.D., Castine, W.H. & King, F.J. (1988). *Assessing the Impact of Computer-Based Instruction: A Review of Recent Research.* New York: Haworth Press.

[Self 1985] Self, J. (1985). *Microcomputers in Education: A Critical Evaluation of Educational Software.* The Harvester Press.

[Smith 1984] Smith, P.L. & Boyce, B.A. (1984). Instructional Design Considerations in the Development of Computer-Assisted Instruction. *Educational Technology 24* (7), 5-11.

[Soulier 1988] Soulier, J.S. (1988). *The Design and Development of Computer Based Instruction.* Boston: Allyn and Bacon, Inc.

[Venezky 1991] Venezky, R. & Osin, L. (1991). *The Intelligent Design of Computer-Assisted Instruction.* New York: Longman.

[Vockell 1989] Vockell, E. & van Deusen, R.M. (1989). *The Computer and Higher-Order Thinking Skills.* Watsonville, CA: Mitchell Publishing, Inc.

[West 1991] West, C.K., Farmer, J.A. & Wolff, P.M. (1991). *Instructional Design: Implications from Cognitive Science.* Englewood Cliffs, N.J.: Prentice Hall.

[Wilson 1992] Wilson, B. & Cole, P. (1992). A Review of Cognitive Teaching Models. *Educational Technology Research and Development 39* (4), 47-64.

CHAPTER SEVEN

ARTIFICIAL INTELLIGENCE AND COMPUTER-AIDED INSTRUCTION

This chapter investigates the application of Artificial Intelligence (AI) techniques to CAI. Intelligent Computer-Assisted Instruction (ICAI) has evolved from a base of Computer Science and Cognitive Psychology, rather than from the realms of education. It focuses more upon subdisciplines of AI, such as *knowledge representation, natural language* dialogues, and *inferencing* than on instructional features. The origin and kinds of ICAI are investigated. The components of Intelligent Tutoring Systems (ITSs) are overviewed. The chapter concludes with an historical perspective on ICAI, a discussion of problems incurred, and a view of some of its current tendencies.

> In relation to the software engineering life-cycle models, the study in this chapter relates to decisions that must be made in the requirements analysis phase. The prototype life-cycle permits a degree of experimentation and consequent refinements to the requirements specification in situations where initial expectations are unrealistic.

Intelligent tutoring systems differ considerably from conventional computer-assisted instruction programs. The two have complementary approaches, which are outlined by Larkin & Chabay [Larkin 1991]. CAI programs are *interaction-centred* - the author or designer sets out with a definite goal, aiming for a certain type of interaction and specific activities, founded on teaching experience and instructional principles. The programs are generally *algorithmic*, with user-input determining branching. Intelligent tutors, in contrast, are *knowledge-centred*. It is not hand-crafted programming code, but a set of knowledge-based models, which determine how the software responds to the student. After interaction with the student, the model communicates with the interface or another model, which uses its own knowledge to instruct the student and convey input back to the central model. The logic of an ITS is not algorithmic, but based on repeatedly applying knowledge, *encoded as rules*, to react to a current situation. The rules are often encoded in the classic symbol-processing languages of AI, such as LISP and Prolog.

7.1 GENERATIVE SYSTEMS

Generative CAI [Keller 1987; Sleeman 1982; O'Shea 1983] was the precursor to ICAI. In the mid 1960s drill-and-practice software was developed which presented a learner with problems at an appropriate difficulty-level. The personalized degree of difficulty was determined by task-selection algorithms which measured the behaviour of the learner and used it to **select suitable exercises**. Such programs were called adaptive, because the problems presented were individualized.

In another form of generative software the **computer generates random numbers or words** (within certain restrictions), and uses them to create specific values for exercises in computational and linguistic domains respectively.

Thirdly, a question bank can be compiled for a particular subject. The random number generator can **select items at random** from the bank for practice-exercises or for testing students.

7.2 ARTIFICIAL INTELLIGENCE AND COMPUTER-ASSISTED INSTRUCTION (AI + CAI = ICAI)

7.2.1 DEFICIENCIES OF TRADITIONAL CAI

Traditional scripted CAI has deficiencies when compared to an intelligent human teacher [Keller 1987; Thomas 1990]. It does not:

- have any intrinsic knowledge of the subject matter it teaches,
- model students, thus it has no knowledge of the individual student, nor any understanding of his kind of error,
- permit students to express a full range of variety in their responses,
- allow students to talk in natural language, ask questions, or take initiative, or
- have the capability to instruct using a variety of instructional strategies and non-deterministic sequencing.

7.2.2 AI'S CONTRIBUTION: KNOWLEDGE REPRESENTATION

To address these deficiencies and behave intelligently, the program requires knowledge. In an effort to produce intelligent CAI, techniques that emerged from the AI *knowledge representation* research of the 1960s and 1970s have been used to provide underlying knowledge structures for CAI. The key to intelligent processing is the proper organization of knowledge, both *declarative* and *procedural* knowledge. Woolf [Woolf 1988, p. 3] explains:

> "Intelligent teaching systems are designed to represent both the concepts to be taught and how a student might learn those concepts".

The major knowledge representation formalisms [de Villiers 1989b] are *networks, frames*, and *production rules*:

Networks

Quillian [Quillian 1968] developed *semantic networks* of objects and relations between them, the term "semantic" being due to their origin in the field of natural language processing. His work on semantic memory is based on *associative links*, and is related to Simon's (section 4.1.4) model of human information processing. Nodes are related to one another by different kinds of links, each having a different meaning. Networks lend themselves to representation by diagrams or other notations, and can be encoded using various symbols to represent the different kinds of links.

Frames

The concepts, terms and techniques of networks paved the way for the subsequent development of *frames*, a variant of network with a more complex structure. Minsky [Minsky 1975; Minsky 1985] proposed a frame or schema as a method of representing both human and machine knowledge. A frame is a data-structure, consisting of nodes and relations, that represents a *stereotyped* object or situation. The various slots (or terminals) have values, and where a specific value is not provided, a default assignment exists to cater for the general, or typical, case. Frames are stored and can be accessed for inferencing or to retrieve information.

Frames are applied, or *instantiated,* as follows: on encountering a situation, a frame that fits is selected from memory to represent the situation. The default values assigned are used where appropriate; other aspects may be adapted to fit reality. In a CAI situation, an author may fill a knowledge frame or script with domain material, and be prompted by requests for any information missing [Nicolson 1988]. Once encoded, the knowledge is available for retrieval and manipulation by the computer.

Production Rules

Production rules, described in section 4.2.6 represent knowledge by "if ... then ... " propositions, formally described as *condition-action* or *antecedent-consequence* pairs.

They are used for inferencing as follows: In *forward chaining* data patterns are matched against the clause/s in the antecedent and the resulting action or consequence is triggered. *Backward chaining* is working backwards from a goal, pattern-matching data against the consequence; if found, then the corresponding antecedent holds true [Anderson 1983; Fischetti 1990].

7.2.3 INTELLIGENT CAI (ICAI)

The AI techniques outlined above make possible the creation of knowledge platforms on which intelligent CAI programs [Kearsley 1987; Kok 1990] can be constructed. The learning theories discussed in previous chapters also play a role, in that traditional CAI emerged from behaviourism, whereas ICAI has roots in cognitive science. In fact ICAI lies "at the intersection of computer science, cognitive psychology, and educational research" [Kearsley 1987, p 3], but it is largely the contribution of AI that distinguishes ICAI from traditional CAI and CBE. Both CAI and ICAI strive to be adaptive, but the nature of the adaptivity and individualization within ICAI is far richer.

ICAI is a broad field, embodying various paradigms:

- *Mixed-initiative dialogues*: such as SCHOLAR (see section 7.2.4), which could generate questions about the material in its knowledge base, but could also answer questions posed by students.

- *Coaches and diagnostic tutors*: which observe the learner's performance, providing advice, prompts, interrupts, and error-diagnosis. The assessment may be based on a comparison of the learner's problem-solving approach with expert strategies. Intelligent tutoring systems are discussed in section 7.3.

- *Microworlds (or open learning environments)*: which are computational and/or graphic tools that allow learners to explore a problem domain. The classic example is Papert's Logo (described in section 5.4.3).

7.2.4 THE FIRST EXAMPLE OF AI IN CAI

The first computer-assisted learning system which set out to simulate a human tutor by using AI techniques was SCHOLAR [Carbonell 1970; O'Shea 1983; Venezky 1991]. SCHOLAR was intended to review the student's knowledge of the geography of South America. The system itself "knew" the material it was intended to teach. Its database or *information structure*, comprising facts, concepts, relationships and procedures, was a semantic network [Quillian 1968] from which associations and inferences could be made. No specific pieces of text, pre-specified questions, or anticipated answers were formulated in advance. The semantic network was used in two ways. Firstly, the system could construct text or generate questions according to *stereotypes*. However, SCHOLAR offered the learner a *mixed-initiative* dialogue, thus students themselves could also pose questions. The system,"knowing" the structure of its underlying network and the meaning of the relationships, could make inferences to answer the student's queries.

It had the potential for diagnostic capabilities, although such were not actually incorporated. Carbonell believed that, should problems occur, the ultimate objective was [Carbonell 1970, p. 198] "for the student to overcome them, not for the teacher to diagnose them". Even a human tutor frequently handles student-errors by giving remedial treatment and basic instruction, without fully entering into the student's misconceptions.

Of particular note is the fact that SCHOLAR'S executive program is independent of subject matter. The *content-free* executive program can operate on different applications when the subject-matter content of the semantic network is changed. Computational applications can be addressed by creating a semantic network that is strong on procedures. With reference to the matter of problem-diagnosis in the previous paragraph, it is obvious that this context-free approach makes it harder to program the computer to diagnose errors.

SCHOLAR set a precedent, and was a seminal landmark in terms of its influence on the developing field of AI within CAI, but the archetypal intelligent system which succeeded it seldom incorporated a mixed-initiative strategy. Although intelligent systems, which are discussed in detail in the next section, do adapt to the learner, they do so in a context of program-iniative.

7.3 INTELLIGENT TUTORING SYSTEMS (ITSs)

7.3.1 CHARACTERISTICS AND COMPONENTS OF AN ITS

A particular kind of ICAI software is called an Intelligent Tutoring System (ITS) [Self 1979; Keller 1987; Self 1988; Fischetti 1990; Kok 1990; Nix 1990; Thomas 1990; Goodyear 1991; Venezky 1991; Regian 1992]. SCHOLAR, with its underlying information structure containing knowledge of the domain, is considered to be the classic from which ITS research evolved. An ITS is a limited, or focused, ICAI program, which sets out to instruct and offer practice within a specific area of a curriculum, using some of the ICAI techniques outlined above, but excluding microworld features. The difference between traditional CAI and ITS is not necessarily in their functionality, but in their structure, control and methods. **Both strive for individualization** - adaptation to the individual needs of each learner, but the intelligent software has a more powerful approach, based on inferencing and the *explicit encoding of knowledge*, rather than on *pre-specified branching*. Since the fundamental purpose is to emulate a human tutor in a one-to-one relationship, an ITS often emphasizes interactive practice of skills more than explicit teaching. An ITS is sensitive and adaptive to the learner, in that it knows *what* is being taught, *who* is being taught, and *how* to teach it to him [O'Shea 1983; Fischetti 1990; Goodyear 1991; Regian 1992].

In general an ITS is characterized by a user-interface and a diverse set of knowledge bases, being some or all of the following:

- *Expert model (what)*: explicit representation of domain knowledge. It knows the facts and concepts of the topic, not as a human would, but in terms of the facts and concepts of the subject matter and the relationships between them, so that it can solve problems in the domain.

- *Student model (who)*: representation of the learner and what he knows in terms of knowledge and skills; it is constructed and dynamically updated during the interaction. One approach is to compare the student-performance with, or to view it as a subset of, the expert-model.

- *Diagnostic model (why)*: for error-diagnosis and identification of misconceptions.

- *Tutoring module (how)*: knowledge about how, what, and when to teach. It uses the expert model and the student model to make its decisions; its knowledge is often represented as strategy rules.

The inter-relationship between the components of an ITS is illustrated in Figure 7.1.

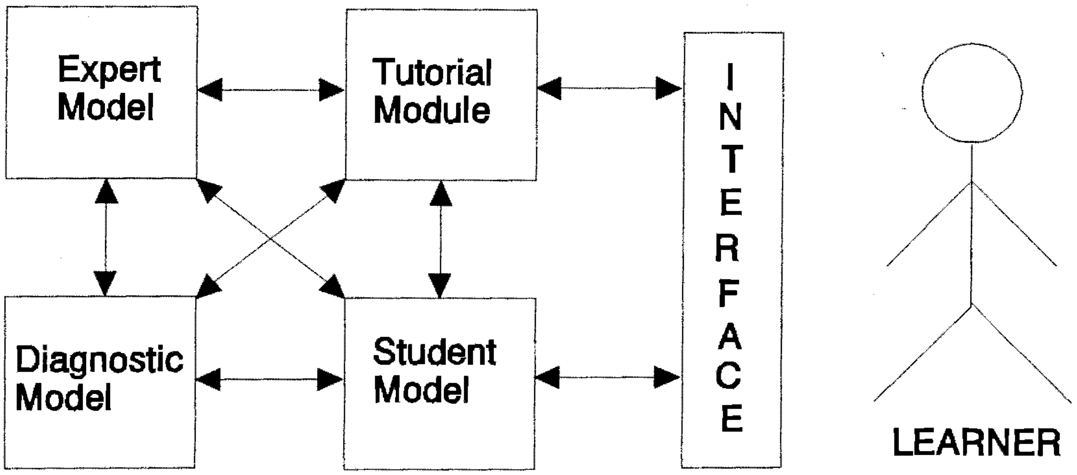


Figure 7.1 Architecture of an ITS

7.3.2 ANDERSON'S ACT* THEORY, COGNITIVE PRINCIPLES AND INTELLIGENT TUTORS

John Anderson's [Anderson 1983; Anderson 1987; Anderson 1990; Anderson 1992] Adaptive Control of Thought (ACT) theory of the operation of the human cognitive system relates to the control of higher-level cognition, using *production rules to model human memory and cognitive skills.* A fundamental distinction is drawn between declarative and procedural knowledge. Declarative knowledge is largely factual and is that which can be encoded quickly; procedural knowledge, on the other hand, is acquired through the use of declarative knowledge, thus the theory of procedural learning emphasizes learning by doing. Procedural knowledge is represented by a set of production rules. ACT underwent various revisions; the final reformulation is called ACT*. ACT* theory in its entirety is complex, but the procedural aspects are particularly relevant to the tutoring of cognitive skills. Tutoring principles, listed below, have been derived from ACT* theory, and empirically confirmed:

1. Cognitive tutors should provide an explicit model of the student's ideal behaviour, i.e. there should be an *ideal model*, analogous to an expert model, capable of solving problems, and a *student model* that records actual behaviour. In ACT* theory, these models should be represented by production systems. The student model is more than just an indicator; it sets out the student's errors and the knowledge he is acquiring.

2. Human behaviour in problem-solving is based on goals, i.e. a learner believes that
   IF <a certain condition>
   THEN <a certain consequence is the case
   or a certain action must be taken>.

   Production rules are *goal-factored* and lend themselves to this type of situation. Students use rules by backward chaining from the desired consequence to the condition/s he must achieve or prove. General goals should be given primacy and the appropriate productions communicated to the student.

3. Students learn more effectively if presented with the necessary information during problem-solving. Learning by doing is enhanced by receiving instruction in the problem solving-context; this helps the learner form his own more specific productions.

4. Abstract high-level understanding should be promoted, training the learner to grasp general principles and rules, and apply them in the context of a specific example.

5. Load on working memory should be minimized by providing rapid updates of necessary information.

6. *Model-tracing methodology* is applied, using *cognitive models* to interpret the student's problem-solving behaviour and identify the rules he is using. The model-tracing tutor incorporates both the correct production rules and a set of *buggy rules* to account for the student's errors.

7. Rapid feedback should be given when a learner applies an incorrect production. This prevents further errors.

Intelligent tutors in LISP, algebra, geometry, and a multiple programming languages project have been built upon the ACT* theory [Corbett 1991; Anderson 1987; Anderson 1990; Anderson 1992]. These tutors have been tested and used in a university and in high schools over a period of ten years. Some of the implementation issues and practical features of these tutors are listed:

- ♦ The tutors are termed "intelligent", because they are capable of generating correct solutions to the exercises and helping learners to do so.

- ♦ The tutor provides an environment, similar to an editor, in which a student can do exercises. Unlike an editor, it checks the student's input and ensures that the final solution is correct.

- The current state of the student's solution to a problem is stored in a limited capacity working memory.

- Exchanges between student and tutor are in the context of the problem-solving exercises, and occur only when the student is having difficulties or makes a mistake.

- Certain exercises have alternative correct solutions. The tutors enforce stylistic constraints, but are intended to recognise any reasonable attempts.

- Most of the instruction occurs when students request help, but intervention also occurs whenever errors are made.

7.3.3 TOWARDS ITS SHELLS

An *ITS shell* [Fischetti 1990] is a domain-independent tutoring system, which can operate over a variety of application domains. The domain knowledge for a particular subject or topic is used to construct a domain-specific knowledge base (KB). It usually requires an expert AI programmer or a knowledge engineer to create the knowledge representation, since no adequate tools are available that permit domain experts to construct KBs directly. It is unlikely that a general-purpose, domain-independent tutoring system will be available in the near future.

Clancey developed GUIDON, a domain-independent teaching program that instructs students in diagnostic strategy [Clancey 1981; Clancey 1982; Clancey 1987]. It originated from the knowledge-based expert-system, MYCIN, which diagnoses infectious diseases. An *expert-system* is an advisory or decision-making program, with a separate knowledge-base (KB) and inference mechanism. An *expert system shell* is an expert-system without a KB. The expert system, MYCIN, spawned the domain-independent expert-system shell, EMYCIN. EMYCIN uses MYCIN's production rule language and inferencing mechanisms, but "plugs into" KBs custom-built for particular domains. A tutorial, GUIDON, was built to operate on any EMYCIN-compatible KB, but instead of actually performing diagnosis, it teaches students, by means of a tutorial dialogue, how an expert approaches diagnosis and problem-solving. The teaching expertise is represented explicitly and is wholly independent of the domain knowledge base.

7.4 PROBLEMS IN ICAI

Many of the prototype intelligent tutors are largely experimental; most are limited in domain and capabilities. Very few are commercially available [Fischetti 1990].

Intense effort is required to create knowledge-based programs to simulate human tutors. O'Shea & Self [O'Shea 1983, p. 120] state that it "requires man-years of expert programming". Lippert [Lippert 1989] gives a figure of over 1500 hours of development time for one hour of ICAI instruction time.

7.4.1 PROBLEMS WITH STUDENT MODELLING

In particular there are problems with relation to student modelling [Self 1979; Ridgeway 1988; Cumming 1991b]. A prime purpose of a student model is to represent changes in the student's knowledge, and a secondary purpose is to provide an explanation of his errors or identify inadequacies in his problem-solving process. Knowledge representation techniques simplify the extension or alteration of the student model to record what the learner currently knows, but the incorporation of a diagnostic component to identify the actual learning processes is not simple. Furthermore, there is a huge range of possible models, and in a complex domain it is unlikely that an exhaustive set could be compiled.

7.4.2 PROBLEMS WITH THE TUTORING MODULE

Another fundamental problem is the overall control exercised by the tutorial module, and the concomitant lack of student-initiative. Learning could possibly be more effective and lead to more self-growth if the learner had more control. Cumming & Self describe ITSs as having an "authoritarian tutoring style" [Cumming 1991b, p. 87], and propose instead an *intelligent educational system*, in which the user is a collaborator, sharing even the facility of inspecting and changing the user model. Ridgeway [Ridgeway 1988, p. 28] views ICAI as "focused on individual tuition for technical mastery".

7.4.3 PROBLEMS WITH CREATION OF THE KNOWLEDGE BASE (EXPERT MODEL)

A particular problem in the development of ITSs is the creation of knowledge-bases. The question arises as to whether the expert model, i.e. the domain KB, should be created by the domain expert himself or by a computer scientist intermediary called a *knowledge engineer*, who is instructed by an expert and then encodes the knowledge. Ideally, knowledge should be entered by the domain expert to ensure completeness, accuracy, and the inclusion of subtleties, but the programming can be a problem. The tendency, however, in ICAI and ITSs has been development by computer scientists, i.e. AI specialists, with low involvement by educational technologists, and little learner-evaluation or thorough testing [Lippert 1989; Fischetti 1990]. However, some systems have been developed that allow domain experts to enter information into the KB themselves, but in a restricted class of programs.

A further basic problem is that the domain must be narrow and limited in order for the system to incorporate knowledge and intelligence at all.

Anderson et al [Anderson 1992, p. 83) acknowledge that "the work and difficulty in developing an adequate cognitive model (their term for the expert model) are the major obstacle in this approach to tutoring".

7.4.4 PROBLEMS WITH MULTIPLE SOLUTIONS

Difficulties also arise when different approaches, multiple representations, and alternative correct solutions exist for the same problem. Many ITSs (and almost all traditional CAI programs) do not provide for an approach or a skill that differs from the expert's model solution. When the student diverges from the expert's approach, control is removed from the user, and he is prompted or corrected to do as the expert would have done [Keller 1987; Ridgeway 1988; Woodroffe 1988; Boder 1990].

7.4.5 PROBLEMS WITH EVALUATION

A general problem with work in intelligent tutoring is that it has tended to progress without empirical feedback as to whether the proposed mechanisms work [Anderson 1990, Corbett 1993]. Notable exceptions are found in the efforts of Anderson and colleagues, whose tutors have been systematically tested in the classroom with satisfactory results.

7.5 EVALUATION OF ICAI

Ford [Ford 1988] provides a list of questions developed by Self to appraise the behaviour of an ICAI system. They occur in four categories, and some are given below:

- *Subject knowledge:*
  Can the system answer questions from the user?
  Can it give an explanation of a solution?
- *Student knowledge:*
  Can the system report on the user's level of understanding?
  Are explanations tailored to the user?

Does it provide informative feedback?
Are the tasks adapted to the user's needs?

- *Control:*
  Does the system actively engage the user?
  Can the user initiate a new area?
  If so, does the system monitor the change and comment if unwise?
  Does it intervene when the user has difficulty?
- *Mode of communication:*
  Can user-input be expressed in a natural manner?
  Does the system help if user-input is not understandable?
  Are the system's outputs natural?

It is clear that few systems can be positively assessed on all criteria. To achieve even a few of the above characteristics would be a highly labour-intensive process.

7.6 CURRENT VIEWS ON ICAI AND ITSs

7.6.1 REVIEW OF INITIAL INTENTIONS IN THE 1980s

In 1982 Sleeman and Brown [Sleeman 1982], editing the special landmark issue, *Intelligent Tutoring Systems,* mentioned the intention of ITSs to combine discovery-learning with tutorial guidance, objectives often in conflict. They saw the incorporation of the program's own problem-solving expertise and modelling capabilities as the way to augment open-ended, problem-solving environments with intelligent tutoring.

However, the discussion in sections 7.4 and 7.5 outlines problems that have been incurred in ITSs, despite the idealism that accompanied their advent.

7.6.2 PRAGMATIC TRENDS IN THE 1990s

More recent proponents of pure AI, Clancey & Soloway [Clancey 1990], editing the 1990 special issue, *Artificial Intelligence and Learning Environments,* point out trends and techniques that should shape the field in the 1990s. Some of these are:

- use of graphics for explanation,
- reconsidering the extent to which student modelling is possible or necessary,
- designing shells for use in multiple domains by teachers, and
- defining sequences of activities for exploration-learning.

The cognitive modelling and ACT* intelligent tutors [Anderson 1990] are compatible with content-free use in multiple domains. They incorporate student modules, pedagogical modules and the interface for student-interaction. There are three notable issues relating to these components:

- From a software engineering perspective, the various components can be developed separately, increasing the tractability and modularity of such major software projects.
- A related implication is re-use of content-free software. Student modules and interfaces contain domain knowledge, but the pedagogical strategy which controls the interaction is independent of domain material. Anderson and his colleagues hope to build different tutors using domain-free tutoring strategies.

Goodyear [Goodyear 1991] sees future research including:

- efforts to solve the problems of learner modelling and domain knowledge,
- the use of less didactic teaching styles, as collaborative learning and guided exploration play greater roles, and
- improved learner-computer communication, such as via graphic interfaces.

7.6.3 LESS OPTIMISTIC OUTLOOKS

Certain schools of thought query whether ITSs can live up to their promise. Winograd & Flores [Winograd 1986] doubt that symbolic representations can ever truly embody intelligence, and suggest that computer systems should be designed to be of practical value in a relevant context, requiring common sense and general problem-solving abilities from the user.

Cumming [Cumming 1991a] and Cumming & Self [Cumming 1991b] believe that it is extremely difficult to build effective intelligent tutors for any but restricted domains and narrow learning. Cumming's view [Cumming 1991a, p. 52) is that

> 'Under the new thinking, systems may be less ambitious and rely more on human collaboration, but they will be more practically useful than the artificial AI systems of the past".

In some quarters the acronym ICAI is now considered to stand for Interactive CAI rather than Intelligent CAI. In posing and answering the question, "What makes CAI intelligent?", Lippert [Lippert 1990] suggested the three characteristics of *adaptability, individualization* and *interactivity.*

Keller [Keller 1987] proposes Intelligent Support Systems (ISSs), which cannot solve problems themselves, but can help a student to do so by, for example, presenting a relevant rule. The learner takes an active role and the ISS acts as a support, an intelligent assistant monitoring progress, rather than a master.

7.7 APPLICATION TO CAI PRACTICE ENVIRONMENTS AND TO FRAMES IN PARTICULAR

7.7.1 AI AND THE ADULT LEARNER

The instructional context of FRAMES is tertiary-level distance education. Cumming [Cumming 1991a] discusses the preferred learning style of adult learners, which includes:

- choice between redundant diversity,
- meaningful activities, and
- availability of advice and guidance on request.

The relevance to instructional computing is that attempts to channel the adult's learning model into a rigid path and close convergence with an expert's approach are likely to frustrate the learner. An ideal is the provision of variety and "apprenticeship" situations in a context that offers a combination of learner-control with advice and guidance. Diversity should be available, and the mature learner permitted to choose between:

- practice of lower-level skills and sub-components of tasks,
- guidance by an expert leading to self-evaluation,
- increasingly complex tasks.

The learner is expected progressively to undertake more planning, control and evaluation. Note that the above range implicitly incorporates both behaviouristic and cognitive learning theories, behaviouristic in the trend towards attainment of automaticity in low-level skills, and cognitive as the learner is encouraged to modify and multiply his own mental models.

This approach blends with the philosophy of Winograd & Flores [Winograd 1986] that effective computer systems need not be inherently intelligent, but should rather be intelligently embedded within their context. Keller's *intelligent support systems* (section 7.6.3), similarly, monitor and help but do not control the learner.

7.7.2 AI AND EXAMPLE PRACTICE

Where the domain can be formalized and a "best path" through it exists, intelligent tutoring may be the most appropriate *modus operandi* [Hammond 1992a]. However the archetypal ITS model would be inefficient for a practice-environment such as FRAMES which incorporates wide diversity in domain examples. The varied examples cannot be solved by a formalized generic procedure or a sufficiently general algorithm.

Generative software, whereby examples for learner-practice are automatically generated and assessed according to a general algorithm, is described in section 7.1. The usual alternative is explicit encoding of each problem and its solution. A compromise is *courseware abstraction* [Webb 1989], where **a general parameterized segment of CAI material is applied to multiple prespecified examples**. This provides the learner with a varied series of *concrete examples*, rather than randomly generated examples, but all within the context of a single treatment structure. It is an ideal approach for the material covered in FRAMES, where the manipulations involve the same set of tests and steps, but the internal logic of each problem-relation entails rich variation. To automatically generate and process the detailed material required for the varied relations would be extremely difficult; it is far simpler to create explicit example descriptions and apply the abstracted processing structure to them. In short, **generative exercises** would not be appropriate, but a **generated structure** has high utility - it should simplify development and expedite abstraction within the student.

7.7.3 KNOWLEDGE INHERENT IN THE PROGRAM

Following the discussion of knowledge-based modules in this chapter, the optimum situation for a mathematics practice environment appears to be a program with knowledge of:

- the overall structure of the domain and its major problem-solving steps, for the purpose of structuring and sub-dividing the problem-solving process, and

- the general content of each domain element, in order to facilitate limited exploration and discovery-learning.

Due to the intense effort required and the wide variety of examples, there should not be detailed problem-solving knowledge, nor an expert model, student model or diagnostic model.

7.7.4 INTEGRATION OF AI AND GAGNé'S LEARNING THEORY

Lavoie *et al* [Lavoie 1991] express concern about the dichotomy between AI researchers on the one hand, and educational psychologists and instructional designers on the other. AI tools are developed to build knowledge-based instructional software. Those skilled in using these tools are seldom qualified educationalists, but the tools are too technical to be used by general developers of instructional software.

Basing their work on Gagné's learning theory (see section 4.2.2), they developed an authoring system for the design and construction of knowledge-based instructional software. The system architecture incorporates the usual kind of module generally found in an intelligent tutor, but also takes into account the subject matter, its comprehensibility, and the various kinds of activities entailed. Particular attention is paid to prerequisites needed before attainment of the final objectives. The content material is subdivided into the smallest possible learning situations, so as to increase flexibility. In order to achieve Gagné's different kinds of learning outcomes, the knowledge module comprises activities geared to:

- initial subject appropriation.
- strengthening of learning.
- Gagné's lower intellectual skills:
  - (a) discriminations, and
  - (b) concrete and defined concepts.
- Gagné's higher order intellectual skills:
  - (a) rules, and
  - (b) higher-order rules.
- Gagné's cognitive strategies.

These are all aspects that should be fostered in a practice environment, whether or not it is knowledge-based.

7.7.5 RE-USE OF CONTENT-FREE AND DOMAIN INDEPENDENT MODULES.

Section 5.6 advocated the use of content-free instructional strategies, a concept which re-occurred in sections 7.3.3 and 7.6.2 where shells were proposed for use in multiple domains. Section 7.6.2 also advocated the re-use of content-free modules of intelligent tutors. Trends towards development and re-use of modular content-independent structural components would optimize the efforts entailed in producing complex instructional software.

7.8 CONCLUSION

In this chapter the application of AI to CAI was investigated. Particular attention was paid to the adaptive instructional systems known as intelligent tutoring systems. Most of the power of an ITS comes from its inherent knowledge, and most of the development effort goes into formalization of the relevant knowledge. The question arises whether ICAI, with its heavy time-investment, is largely an academic research effort, or whether it is instructionally effective and accompanied by concomitant educational progress. There is a current belief that the most effective and "intelligent" role of computers in education is their use as tools.

An ITS is strong on individualization, largely by means of adaptivity and program-control, based upon its knowledge of the student and knowledge of tutorial strategies. It is possible, with current trends away from *authoritarianism and autocracy*towards *democracy*and people-empowerment, that the pendulum could be swinging towards a more flexible approach of individualization by means of learner-control, rather than individualization by program control. The next chapter takes an in-depth view of concepts such as user-control and human-computer interaction.

REFERENCES - CHAPTER SEVEN

[Anderson 1983] Anderson, J.R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.

[Anderson 1987] Anderson, J.R., Boyle, C.F., Farrell, R. & Reiser, B.J. (1987). Cognitive Principles in the Design of Computer Tutors. In: Morris, P. (Ed.) *Modelling Cognition.* Chichester: John Wiley & Sons.

[Anderson 1990] Anderson, J.R., Boyle, C.F., Corbett, A.T. & Lewis, M.W. (1990). Cognitive Modelling and Intelligent Tutoring. *Artificial Intelligence 42*, 7-49

[Anderson 1992] Anderson, J.R., Corbett, A.T., Fincham, J.M., Hoffman, D. & Pelletier, R. (1992). General Principles for an Intelligent Tutoring Architecture. In: Regian, J.W. & Shute, V.J. (Eds). *Cognitive Approaches to Automated Instruction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Boder 1990] Boder, A. & Cavallo, D. (1990). An Epistemological Approach to Intelligent Tutoring Systems. *Intelligent Tutoring Media 1* (1), 23-29.

[Carbonell 1970] Carbonell, J.R. (1970). AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction. *IEEE Transactions on Man-Machine Systems 11* (4), 190-202.

[Clancey 1981] Clancey, W.J. & Letsinger, R. (1981). Neomycin: Reconfiguring a Rule-Based Expert System for Application to Teaching. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Volume II.* Los Altos, CA: William Kaufman Inc.

[Clancey 1982] Clancey, W.J. (1982). Tutoring Rules for Guiding a Case Method Dialogue. In: Sleeman, D. & Brown, J.S. (Eds), *Intelligent Tutoring Systems.* London: Academic Press.

[Clancey 1987] Clancey, W.J. (1987). Methodology for Building an Intelligent Tutoring System. In: Kearsley, G. (Ed.), *Artificial Intelligence and Instruction: Applications and Methods.* Reading, MA: Addison-Wesley.

[Clancey 1990] Clancey, W.J. & Soloway, E. (Eds) (1990). *Artificial Intelligence and Learning Environments.* Cambridge, MA: MIT Press.

[Corbett 1991] Corbett, A.T. & Anderson, J.R. (1991). LISP Intelligent Tutoring System: Research in Skill Acquisition. In: Larkin, J.H. & Chabay R.W. (Eds), *Computer Assisted Instruction and Intelligent Tutoring Systems.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Corbett 1993] Corbett, A.T., Anderson, J.R. & O'Brien, A.T. (1993). The Predictive Validity of Student Modelling in the ACT Programming Tutor. *Proceedings of AI-ED 93, World Conference on Artificial Intelligence in Education.* Edinburgh: Association for the Advancement of Computing in Education.

[Cumming 1991a] Cumming, G. (1991). Using Artificial Intelligence to Achieve Natural Learning. In: Lewis, R. & Otsuki, S. (Eds), *Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education.* Amsterdam: North-Holland.

[Cumming 1991b] Cumming, G. & Self, J. (1991). Learner Modelling in Collaborative Intelligent Educational Systems. In: Goodyear, P. (Ed.), *Teaching Knowledge and Intelligent Tutoring.* Norwood, N.J.: Ablex.

[De Villiers 1989b] de Villiers, M.R. (1989). *Structured Knowledge Representation with Particular Reference to Frames.* Unpublished Special Topic Report, University of South Africa, Pretoria.

[Fischetti 1990] Fischetti, E. & Gisolfi, A. (1990). From Computer-Aided Instruction to Intelligent Tutoring Systems. *Educational Technology 30* (8), 7-17.

[Ford 1988] Ford, L. (1988). The Appraisal of an ICAI System. In: Self, J.A. (Ed), *Artificial Intelligence and Human Learning.* London: Chapman and Hall.

[Goodyear 1991] Goodyear, P. (Ed.) (1991). *Teaching Knowledge and Intelligent Tutoring.* Norwood, N.J.: Ablex.

[Hammond 1992a] Hammond, N.V. (1992). Learning with Hypertext: Problems, Principles and Prospects. In: McKnight, C., Dillon, A. & Richardson, J. (Eds), *Hypertext: A Psychological Perspective.* Chichester: Ellis Horwood.

[Kearsley 1987] Kearsley, G. (Ed.) (1987). *Artificial Intelligence and Instruction: Applications and Methods.* Reading, MA: Addison-Wesley.

[Keller 1987] Keller, A. (1987). *When Machines Teach: Designing Computer Courseware.* New York: Harper and Row.

[Kok 1990] Kok, W. & Poorthuis, G. (1990). The Effects of Different Teaching Strategies in Three CAI Programs within the Same Content Area. In: Pieters, J.M., Simons, P.R.J. & de Leeuw, L. (Eds), *Research on Computer-Based Instruction.* Amsterdam: Swets & Zeitlinger.

[Larkin 1991] Larkin, J.H. & Chabay, R.W. (Eds) (1991). *Computer Assisted Instruction and Intelligent Tutoring Systems.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Lavoie 1991] Lavoie, M., Gagné, M. & Jacques, A. (1991). Specifications of a Software System which assists in the Design and Construction of Knowledge-Based Instructional Software. In: Lewis, R. & Otsuki, S. (Eds), *Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education.* Amsterdam: North-Holland.

[Lewis 1991] Lewis, R. & Otsuki, S. (Eds) (1991). *Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education.* Amsterdam: North-Holland.

[Lippert 1989] Lippert, R.C. (1989). Expert Systems: Tutors, Tools, and Tutees. *Journal of Computer-Based Instruction 16* (1), 11-19.

[Lippert 1990] Lippert, R.C. (1990). *Wat maak RGO Intelligent?* Paper at the INSTRUCTA 90 Seminar, Rand Afrikaans University, Johannesburg, South Africa.

[Minsky 1975] Minsky, M.L. (1975). A Framework for Representing Knowledge. In: Winston, P.H. (Ed.), *The Psychology of Computer Vision.* New York: McGraw-Hill.

[Minsky 1985] Minsky, M.L. (1985). *The Society of Mind.* New York: Simon and Schuster.

[Morris 1987] Morris, P. (Ed) (1987). *Modelling Cognition.* Chichester: John Wiley & Sons.

[Nicholson 1988] Nicholson, R. (1988). SCALD - Towards an Intelligent Authoring System. In: Self, J.A. (Ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* London: Chapman and Hall.

[Nix 1990] Nix, D. (1990). Should Computers Know what you can do with Them? In: Nix, D. & Spiro, R. (Eds), *Cognition, Education, Multimedia: Exploring Ideas in High Technology.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[O'Shea 1983] O'Shea, T. & Self, J. (1983). *Learning and Teaching with Computers.* Brighton: The Harvester Press.

[Quillian 1968] Quillian, M.R. (1968). Semantic Memory. In: Minsky, M. (Ed), *Semantic Information Processing.* Cambridge, MA: MIT Press.

[Regian 1992] Regian, J.W. & Shute V.J. (1992). Automated Instruction as an Approach to Individualization. In: Regian, J.W. & Shute V.J. (Eds), *Cognitive Approaches to Automated Instruction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Ridgeway 1988] Ridgeway, J. (1988). Of course ICAI is Impossible... Worse though, it might be Seditious. In: Self, J.A. (Ed), *Artificial Intelligence and Human Learning.* London: Chapman and Hall.

[Self 1979] Self, J.A. (1979). Student Models and Artificial Intelligence. *Computers and Education 3,* 309-312.

[Self 1988] Self, J.A. (Ed.) (1988). *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* London: Chapman and Hall.

[Sleeman 1982] Sleeman, D. & Brown, J.S. (Eds) (1982). *Intelligent Tutoring Systems.* London: Academic Press.

[Thomas 1989] Thomas, T.A. (1989). Intelligent Tutoring Systems. *Proceedings of the First Southern African Conference on Educational Technology.* Human Sciences Research Council, Pretoria.

[Venezky 1991] Venezky, R. & Osin, L. (1991). *The Intelligent Design of Computer-Assisted Instruction.* New York: Longman.

[Webb 1989] Webb, G.I. (1989). Courseware Abstraction: Reducing Development Costs while Producing Qualitative Improvements in CAL. *Journal of Computer Assisted Learning 5* (2), 103-113.

[Winograd 1987] Winograd, T. & Flores, F. (1987). *Understanding Computers and Cognition: A New Foundation for Design.* Reading, MA: Addison-Wesley.

[Winston 1975] Winston, P.H. (Ed.) (1975). *The Psychology of Computer Vision.* New York: McGraw-Hill.

[Woodroffe 1988] Woodroffe, M.R. (1988). Plan Recognition and Intelligent Tutoring Systems. In: Self, J.A. (Ed.), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* London: Chapman and Hall.

[Woolf 1988] Woolf, B.P. (1988). Representing Complex Knowledge in an Intelligent Machine Tutor. In: Self, J.A. (Ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* London: Chapman and Hall.

CHAPTER EIGHT

USER-CONTROL, HYPERTEXT, AND USABILITY

The theme of this chapter is the interaction of the user with instructional systems. It overviews various locus of control styles and investigates hypertext with its characteristic free-browsing presentation style. The issues of human-computer interaction, usability and the nature of the user interface are also briefly addressed.

Various kinds of control are overviewed, paying particular attention to hypertext, which offers a high degree of user-control and is the epitome of a proactive design. Although there is an inherent structure and subdivision in the presentation of the material, there is no enforced sequence. The learner may browse or interact in a non-linear fashion and select text-segments in the order of his choice, moving directly from section to section, or delving deeper into a particular concept.

This study considers hypertext in the context of instruction, and suggests various extensions to basic hypertext in order to enhance learning. The locus of control in hypertext is in line with the current tendency towards learner control of instructional activities.

Attention is also paid to the discipline of human-computer interaction and the associated usability factors. Usability factors are the software design requirements which are particularly related to the needs of the human user rather than to technical aspects. The user interface of instructional software is discussed, outlining current requirements and features. The chapter concludes by applying the concepts to the design of CAI practice environments.

> The factors studied in this chapter relate to design stage of an ISD life-cycle. Once the requirements have been set out, a design must be generated to meet them. The designer must determine whether or not hypertext-style control is appropriate. With reference to the aspect of human-computer interaction, the standard of usability should be high and the user interface should be conducive to enjoyable learning.

8.1 LOCUS OF CONTROL

Locus of control in CAI refers to whether control of the sequence, content, methodology and other factors is determined by the learner, the lesson, or a combination of the two. Various kinds of control have been mentioned in sections 6.5.7 & 7.3.1, and fundamentally fall into the following, not necessarily exclusive, categories.

1. *Program control*, either
   - ♦ non-adaptive linear presentation,
   - ♦ adaptive, by branching in reaction to the learner's performance, or
   - ♦ "intelligently" adaptive, where a tutoring module makes inferences using information from a student model.

2. *Mixed-initiative strategies*, e.g. SCHOLAR [Carbonell 1970].

3. *Coactive control*, in which the learner must tackle fixed content, but selects his own sequence, pace, and possibly, presentation style.

4. *Proactive (learner) control*, where the learner chooses or omits content at will, guiding the system to respond to his own ability, preferred learning style or current need.

Hasselerharm & Leemkuil [Hasselerharm 1990] examined the effect of learner control, adaptive program control and non-adaptive program control on the achievement and attitude of secondary level scholars. They determined that learner control resulted in a positive attitude in learners, but was not effective for low achievers. Despite the individualization offered by adaptive systems, they tend to make the learner's role too passive. Adults and mature students, in particular, benefit by active decision-making. Alessi & Trollip [Alessi 1991] propose that adult learners be given more control than children.

In the classic CAI model program control leads the learner through an instructional path. McLean's "Megatrends" [McLean 1989], in their identification of major changes in educational computing, anticipate increasing learner-initiative in CAI and note the trend towards software environments that the user can explore or exploit.

8.1.1 MIDORO'S CLASSIFICATION

Midoro *et al*'s classification of system control differs from the foregoing. Their *pure adaptive* and *pure proactive* categories correspond closely to tutor- and tool-software respectively.

Pure adaptive systems

The major feature of a pure adaptive system [Midoro 1991] is a set of predetermined teaching rules based on input from the student, largely comprised of user-responses to questions, or questions from the user to the system. Parts of instruction that can be adapted are:

- amount of material,
- sequence,
- mode of instruction (expository, inquisitory, general or specific),
- feedback,
- reading or response time, and
- orientation activities.

The term, *adaptive*, traditionally refers to program control, but this definition could incorporate aspects of program control, mixed initiative, and learner control. Depending on the nature of the "questions from the user", the learner may have a degree of control.

Pure proactive systems

According to Midoro *et al*'s definition [Midoro 1991], the major feature of a pure proactive system is that learning actions are determined by the student, selecting from the available system functions. The student gives commands to activate system functions, and information from the system is the result of computations or processes activated by the student.

This classification is restricted, including microworlds, simulations, and programming environments, but excluding most software which offers practice opportunities as a learner-option, such as the type of development envisaged for FRAMES.

8.1.2 CONTROL AND SELECTION IN HYPERTEXT

Hypertext, which is described in detail in section 8.2, offers the user total freedom in selecting content from the software package. It could well be a suitable medium for offering learner-centred control over activities both active and passive, creative and responsive.

Hammond [Hammond 1992b] outlines the dichotomy whereby computer-based learning is traditionally divided into systems supporting tutoring and systems offering exploration facilities. The *tutoring side* constrains the learner's instructional path (either by branching in tutorials, or by ITSs driven by explicit learner-models and expert-models), whereas *hypertext-based free browsing* offers maximum freedom, which, its proponents believe, expedites effective learning. Figure 8.1 demonstrates this dichotomy.

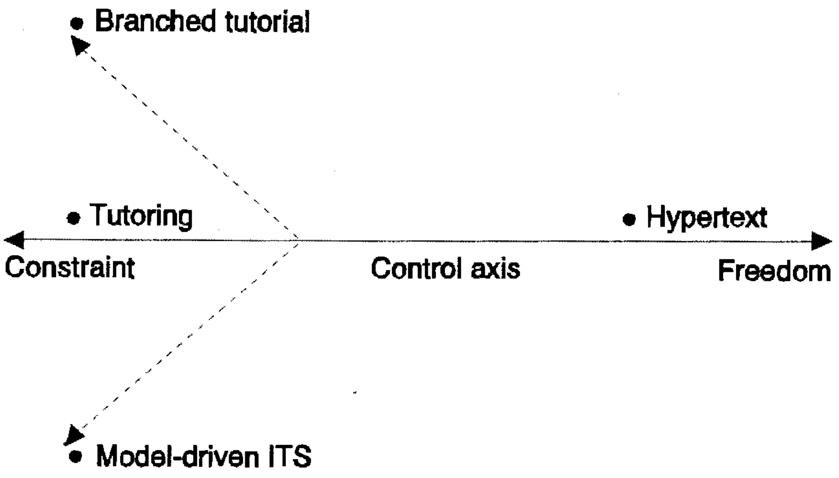


Figure 8.1 Diagrammatic Representation of Locus of Control

Recent approaches are steering between these extremes. Hypertext is strong on navigability and on presentation of material from its database. It is traditionally an information source that does not tutor the learner nor offer practice-opportunities. However, there is no reason why such should not be incorporated. Hypertext advocates acknowledge the problems, and are aiming for a happy medium by providing a range of activities - active, passive, creative, reactive and directed. Hammond argues that **hypertext can provide the framework for a range of tools supporting the multiple learning activities necessary for effective CAI.** The system can be tailored both to the generic requirements of the population of users and to the learning activities of the domain.

In the same spirit, Merrill's Component Display Theory [see section 5.4.5], pre-dating hypertext, describes a conceptual learner-controlled system, not necessarily on the computer as delivery medium, offering a **range of instructional components.** Learners control content, strategy, and the number of examples and items which they choose to do for practice of skills. The instructional environment is rich in variety and it is unlikely that an individual student would use all the material available. Section 6.7 described applications of CDT to CAI, in which a range of content units and learning styles were presented.

Spiro & Jehng [Spiro 1990] developed hypertext programs to help learners acquire knowledge in complex, ill-structured domains. A cornerstone is **multiple representation of knowledge**. They do not include typical practice of skills, but permit students to assemble material by juxtaposing given information schemas.

8.1.3 ASSESSMENT OF CONTROL IN INSTRUCTIONAL SOFTWARE, ACCORDING TO ADAPTIVITY, REACTIVITY AND NAVIGABILITY

Midoro *et al* [Midoro 1991] present a three-dimensional measuring instrument which rates interactive instructional software according to *adaptivity, reactivity* (termed proactivity by the researcher), and *navigability*. Figure 8.2 shows standard software types positioned on the axes of Midoro's interaction space.

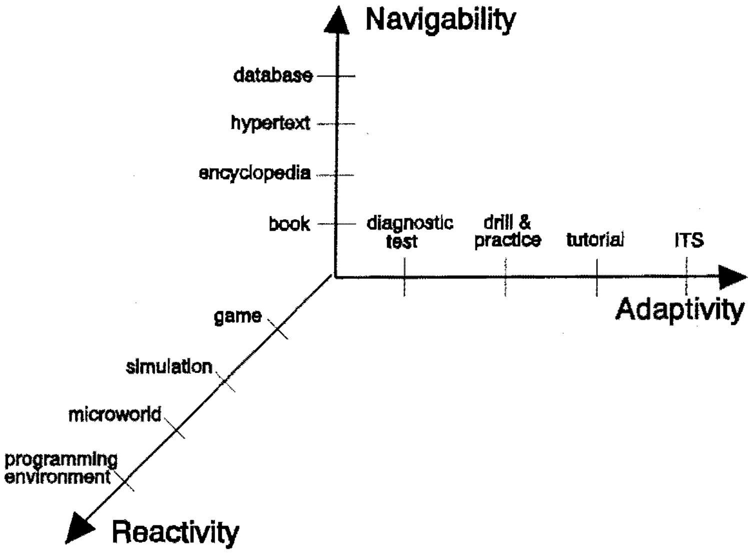


Figure 8.2 Midoro's Interaction Space

8.2 HYPERTEXT AND HYPERMEDIA

8.2.1 DEFINITION OF HYPERTEXT

Hypertext [Jonassen 1989; Nielsen 1990, Hammond 1992a] presents information to the learner in the form of a linked network of displays. Its control structure, which was outlined in section 8.1.2, is such that material is available to the learner for browsing or interaction in a non-linear fashion. He may move directly from one section to another, or while within a section, go deeper into a particular concept. Hypertext consists of *nodes*, which are chunks of text or other information, and *associative links* to interrelate the nodes. The nodes, or units, may be accessed in any sequence, serialistic or holistic, as required by an individual user, and at varying depths. Pure hypertext is thus a mechanism for creating non-sequential texts, where alternative paths potentially link any object (word, paragraph, etc) to any other object in the text [Venezky 1991]. In hypermedia objects can be linked to multi-media objects outside the text, such as audio and video. In both hypertext and hypermedia the user may browse, consult reference material, or participate in selected learning activities. Links are followed from *anchor points* to relevant elaborations, from detail to its greater context, or to re-visit the same content in different contexts (multiple representation). This resembles the associative manner in which humans think, or as Jonassen [Jonassen 1989, p.12] suggests, "reflects the user's semantic network".

More sophisticated systems include additional facilities, such as search mechanisms, tutoring information or support for problem solving. Some even permit users to generate their own materials by interlinking and synthesis.

8.2.2 ADVANTAGES OF HYPERTEXT

For the learner, the chief advantage of hypertext is his freedom to select the viewing path or the approach of his preference. He can explore and interlink materials at will. From the hypertext author's viewpoint, he is not restricted to a linear presentation of a topic [Barrow 1989b]. Supplementary items such as elaborations, glossaries, etc. can be incorporated without interfering with the flow.

8.2.3 DISADVANTAGES OF HYPERTEXT

For the user the increased flexibility may be accompanied by frustration. He may feel overwhelmed by the volume of material available, battle to ascertain his exact whereabouts, or struggle to locate information [Barrow 1989a; Tang 1991].

Hammond & Allinson [Hammond 1989] and Hammond [Hammond 1992b] list some specific problems that occur with hypertext:

1. Users get lost or disoriented in large hypertext structures.
2. Learners may find it difficult to gain an overview of the material and its interrelationships.
3. Even if a user knows specific information to be present, he may have difficulty finding it. It is possible to miss out relevant sections.
4. Learners may ramble through the knowledge base, making haphazard choices in an instructionally inefficient fashion.
5. Hypertext tends to have a complex user interface, which may hinder the learner and interfere with the primary tasks of exploration and learning the content.

From a designer's viewpoint, hypertext is intricate to create; it requires careful planning of structure and complete sets of content [Vockell 1989]. From a programming perspective complexities arise, since few fixed screen layouts exist. Hypertext, and in particular, hypermedia, have disadvantages in that some systems are **technology-driven, with little regard for learning theory**. This problem is addressed in section 8.4, which discusses the role of hypertext in learning and instruction.

8.3 INTEGRATION OF FEATURES OF AI AND HYPERTEXT

In the current climate of cross-fertilization and lateral, multi-disciplinary approaches, it is inevitable that the contrasts and comparisons between AI and hypertext should lead to research on combination of and possible synergism between these fields.

The greatest contrasts are in the areas of control and engagement, where ITSs and hypertext lie at opposite ends of the spectrum. Hammond [Hammond 1992a, p. 52]:

> "While proponents of the intelligent tutoring systems may claim that the course of learning must be driven by explicit models of learners' and experts' states of knowledge, the hypertext philosophy assumes that there is no need to model the student, and that effective learning is achieved by allowing learners maximum freedom to explore ... to discover relationships for themselves ... ".

In his discussion Hammond points out that pure ITSs fall short by failing to give the learner an active role, but that strict hypertext-based learning, despite learner-control, is unlikely to be effective if the learner views the material in a passive, unmotivated and haphazard way.

The problems of student-modelling have already been mentioned in section 7.4.1, and in general, student-modelling in a hypertext-based learning system would be considered out of place. Tang *et al* [Tang 1991], however, innovatively took the strengths of both ITS and hypertext technologies, combined with techniques to reduce their weaknesses, and developed HITS, an authoring environment usable by experts and educators. Midoro *et al* [Midoro 1991] used an ITS, based on a learner model, to guide navigation through a multimedia data base of learning material. This approach was stronger on adaptivity than on learner-control.

An alternative approach is the use of AI knowledge representation techniques to facilitate organization and retrieval in hypertext. The representations used in hypertext and hypermedia systems are compatible with frame-based representations (Minsky-type frames), which are more rigorous than the usual hypertext linking structures. AI rule-based search techniques may be used to reduce complexity in navigation and retrieval [Woodhead, 1991].

8.4 HYPERTEXT IN LEARNING AND INSTRUCTION

8.4.1 THE THEORETICAL BASIS OF HYPERTEXT

Hypertext, and in particular, hypermedia, have disadvantages in that some systems are technology-driven, with little regard for learning theory [Spiro 1990; Dick 1991]. Spiro and Jehng describe hypertext as **atheoretical**, disregarding stages of learning and the cognitive psychology of non-linear learning. It is the antithesis of traditional CAI which typically provides learners only with the choices that ensure stepwise mastery performance. Dick [Dick 1991] suggests that *constructivism* [sections 4.2.7 & 5.4.3] is an appropriate theoretical foundation for computer-based hypertext, because constructivism, as does hypertext, tends to offer the learner discretion to choose from available resources what to study and how to approach it.

8.4.2 HYPERTEXT AND HUMAN COGNITION

Despite the theories and models of human cognition, the detailed mechanisms of learning are not understood [Hammond 1992a]. Nevertheless, the more educationalists attend to the practical aspects of instructional design, the less important this ignorance becomes. Provided there is **understanding of the situations and conditions that promote effective learning**, sound instructional materials can be developed, even without a precise model of cognition. When hypertext-style software is to be used in education and training, it is important that principles of instructional design should be incorporated into the design, and care should be taken that it is not driven merely by a desire to exploit current technology.

8.4.3 BASIC HYPERTEXT AND LEARNING

Barrow [Barrow 1989a] describes two ways in which students can learn using basic hypertext. The first is free browsing, where the user follows links of interest in an associative, discovery-learning process. The second approach is more focused, in that the learner has a precise goal and aims to access specific material.

Hammond & Allinson [Hammond 1989] and Hammond [Hammond 1992a] discuss the role of hypertext in learning. Learning, particularly initial learning, is most effective when the freedom of the learner is restricted to a relevant and helpful subset of activities. Hammond also points out that clear and explicit goals on the learner's part can prevent haphazard meandering in a user-controlled situation. Conventional hypertext has certain inadequacies that mitigate against educational effectiveness. Section 8.5.2 describes how basic hypertext can be adapted or extended to enhance learning.

8.4.4 USING LEARNING NEED TO DETERMINE THE INSTRUCTIONAL APPROACH

Different types of instruction

Learning situations differ [Hammond 1992a] and the various types of CAI, such as intelligent tutoring, programmed learning, drill-and-practice and microworlds, are appropriate for different situations. Hypertext is particularly suitable when the following characteristics are required:

- flexibility in sequence,
- choice of learning activity, and
- variety of instructional approaches within a single package.

Factors which influence learning and instruction

Instruction is most effective when matched to learning need. In general, actions are learned by performing them and concepts are learned by understanding. Instructional materials can be made more meaningful by considering the following factors [Hammond 1992a[1]]:

- *Prior knowledge:* Mental *schemas*, or frameworks of existing knowledge, aid understanding and memorization by facilitating the processes of slotting new facts into existing structures, and correspondingly elaborating the knowledge structure. Simply put, if a fact is relevant to the learner, he will retain and retrieve it better than the learner who has no foundation in that material.
- *Situational dynamics:* Learning action tends to be *situated*, i.e. determined by a combination of the learner's goals and the circumstances of a particular situation. Instructional software should therefore be structured in a task-based fashion.
- *Processing depth:* This relates to the extent to which a learner processes the *meaning* of the instructional material. Cursory browsing results in shallow processing, few elaborations and poor retention. The relevance to CAI is that the instructional designer should aim for *active engagement* of the learner as well as *task relevance*.
- *Learning-by-doing:* Performing an action or participating in a process leads to far better retention of material than merely reading or observing.
- *Learning styles:* Different individuals adopt different learning styles for the same material, and a single individual may change learning styles according to the circumstances and stage of learning. For example, a learner will use the same material in totally different ways on first encounter and in examination preparation.
- *Metacognition:* Extent of learning is affected by a learner's level of metacognitive skills.

[1] Extracted from draft version of paper, not final version as in McKnight 1992.

8.5 THE IMPLEMENTATION OF HYPERTEXT-STYLE STRUCTURE

It is difficult to design and program good hypertext, as stated in section 8.2.3.

8.5.1 DEVELOPMENT CHARACTERISTICS OF HYPERTEXT

When segments of text and graphics are integrated on-screen according to pure user-control, the resultant combinations and screen structures may be unique to each user. In a windowing system users can juxtapose different items for comparative purposes [Barrow 1989b]. The ideal languages or authoring systems to implement such non-prespecified formats are those of the *object-oriented paradigm.* Such a program consists of *scripts* associated with a particular object, rather than a line-by-line or screen-by-screen sequence. User-control should be via text or graphical icons to the object to which the script is attached [Bowers 1989].

8.5.2 ADDITIONAL FEATURES THAT MAY BE INCORPORATED

It was pointed out in section 8.4 that hypertext can include facilities to enhance its instructional utility and to promote learning. Hammond [Hammond 1992a; Hammond 1992b] suggests:

Navigation Aids:
A *map* or *graphical browser* can be incorporated to provide the user with a diagrammatic representation of the overall system structure. *Indexes* of keywords and guided *tours* would ensure exposure to all the basic material. Such navigation aids prevent haphazard browsing and help the learner to control effectively, particularly on first exposure to a topic. At a later stage, a learner may exercise pure proactive control, both in the content selected and in sequence control.

Active Engagement:
Educational hypertext should go beyond the mere provision of content material to be passively imbibed; it should also embody tasks that encourage students to participate actively. Techniques that promote active engagement are exercises, interactive demonstrations and problem-solving activities. A system can incorporate editing facilities that permit learners to synthesize material, which vary from a limited set of options to a full authoring system.

Hammond & Allinson [Hammond 1989, Hammond 1992b] developed a *Learning Support Environment* which augmented hypertext by providing the additional facilities of directed access, guidance and a variety of learning activities, including quizzes and problem-solving for active engagement of the learner.

8.6 HUMAN-COMPUTER INTERACTION, USABILITY AND USER INTERFACE

8.6.1 HUMAN-COMPUTER INTERACTION (HCI)

Human-Computer Interaction (HCI) [Dix 1993; Preece 1993] is the multi-disciplinary study of the human factors in the interaction between people and computer systems, incorporating the study of the physical, psychological and theoretical aspects. It draws on disciplines such as computer science, cognitive psychology, ergonomics, engineering, graphics, design and sociology. It involves the design, development and evaluation of interactive systems in the context of the user's task. The purpose of HCI studies is to improve system design, so that it better meets user needs. In order to achieve this user-centred design, designers need precise information about the *users*, the *task*, and the *work context*, as well as technical aspects.

HCI takes cognisance of the *human information processing model* of cognitive psychology (see 4.1.4 & 4.2.1). The stages are:

- encoding information from the environment into an internal representation,
- comparing this with stored mental representations,
- determining a response,
- organizing the necessary action, and
- remembering and retrieval (which are closely related to the way information was initially encoded).

Viewing the mind in terms of this model has had many implications for HCI and has provided the theoretical basis from which design and evaluation tools have evolved.

8.6.2 USABILITY

The goals of HCI are to develop and improve systems so as to meet users' needs; i.e. the priority of HCI is usability by the human agent, rather than technical issues. *Usability* [Dix 1993; de Wet 1994] is defined as the quality of the interaction between the user and the other parts of the system, including aspects such as system design in relation to users and the environment, and also the nature of the user-support (training, manuals, etc.).

Dix *et al* [Dix 1993] subdivides the principles of usability into three main categories:

1. *Learnability:*
   This relates to the ease with which users can commence interaction and achieve optimal performance. Characteristics that contribute towards learnability are *familiarity*, namely the extent to which the user's experience with other systems is applicable to the new, and *predictability*, the support for the user to determine the effect of actions and operations. Other aspects are *generalizability*, which should help the user to apply his knowledge of specific interactions to other similar situations and the related factor of *consistency* between similar situations and objectives.

2. *Flexibility:*
   Flexibility refers to the variety of ways in which the user and the system exchange information. Once again the characteristic is supported by several principles. *Dialogue initiative* allows the user freedom from artificial constraints on the input dialogue. *Multi-threading* refers to the system's ability to support user interaction with more than one task at a time, and *substitutivity* allows the output of one task to become input of another. *Task migratability* is the ability to pass control of a specific task between user and system or to share it. Finally *customizability* is the facility which permits the user or system to modify the user interface.

3. *Robustness:*
   This entails the level of support given to the user to help him achieve his goals by means of the system. Characteristics that contribute to robustness are *observability*, the degree to which the user can determine internal status of a task or system from the perceived state, and the level of *recoverability*, or corrective action, from an error. *Responsiveness* is the user's perception of the rate of communication with the system. An important aspect is *task conformance*, referring to how well the services of the system support the user in the tasks he performs and in the way he understands them.

Preece [Preece 1993] defines usability as collectively comprising the attributes which allow users to carry out their tasks *safely* (e.g. in medical systems), *effectively*, *efficiently*, and *enjoyably*. In order to achieve the latter three characteristics, the system should have *learnability* to simplify the process of reaching a satisfactory level of user performance, and *flexibility* to expedite adaptation of the system to new ways of interaction. Other criteria of usability are the *attitude* of users to the system and the system's *throughput*, i.e. its capacity in terms of tasks accomplished and execution speed.

Measurement of usability [Bevan 1991; Chapanis 1991; de Wet 1994] is related to the system's:

- *Goal achievement* (accuracy and effectiveness),
- *Work rate* (start-up, productivity and efficiency),
- *Knowledge acquisition* (learnability and learning rate),
- *Operatability* (error-score and function usage), and
- *Versatility* (features and facilities available).

8.6.3 THE USER INTERFACE

User interface refers specifically to the communication means between the user and the computer. The aspects that affect the user's interaction with the system are [Ravden 1989; Brown 1989]:

1. *Computer-to-user functions*:
   - Information display, and
   - Presentation of options.

2. *User-to-computer functions*:
   - Control actions,
   - Manipulation of displayed information, and
   - Entry of data and other information.

There are two fundamental types of user-interface, character-based and graphical user interfaces.

Character-based user interfaces

Character-based user interfaces [Görner 1992] include low-level general purpose textual styles such as command language, menu selection (via non-device access), question-answer, and form-filling. In response to system-generated prompts, the user types in commands, responses or data. Certain command languages have been developed to deal specifically with business information systems. The user and the system communicate through a series of messages which cannot be changed or undone once sent.

Graphical user interfaces

The Graphical User Interface (GUI) [Görner 1992, Dix 1993], presenting information in visual, non-textual forms such as symbols, pictures and icons, is increasingly common. Some are purely graphical; others display the same information in both graphic and text-based forms. Particular mention must be made of the WIMP Interface, a common environment for interactive computing. WIMP stands for windows, icons, menus and pointers (alternatively, windows, icons, mouse and pull-down menus). *Windows* are screen regions, containing text or graphics, that behave as though they were independent terminals. More than one can be simultaneously on display and functional. In sophisticated systems windows can be scrolled or re-sized. An *icon* is a small picture that represents a closed window, or serves as a symbolic representation of some other aspect of the system. *Pointers*, in the form of arrows or fingers, are used to select icons, and are increasingly accessed by mouse-control. *Menus*, common to most systems, present a choice of operations, information or services available. Pull-down menus appear when called, often overlapping other areas, and can be removed when their purpose is fulfilled.

An associated concept is the *metaphor* [Dix 1993; Preece 1993]. In any interaction with a person, a process, or object, a human forms an internal mental model of that system or object. A user's mental model of a computer system, called a system image, is built up from experience of its user interface, its behaviour and its documentation. People involved with the same system, but in different ways, have different mental models due to their personal perspectives. For example, the designer's model of a system is quite different from the end-user's perception.

The relevance of this to software design, and particularly to the user interface, is the importance of "steering" the system image in a way that facilitates development of appropriate mental models. This can be done by designing explicit *metaphors* that correspond to the system, ensuring that the procedures and concepts of the familiar domain, used as a metaphor, map as closely as possible to the structures represented at the user interface.

Characteristics of a good interface

Brown & Cunningham's [Brown 1989] user interface principles define its purpose as facilitating computer use in a consistent yet flexible manner that can accommodate a range of users. The user should be constantly aware of the status and stage of the program, and should have access to help. The interface should be robust and, where possible, should be perceived to operate like other similar programs.

The appearance of screen displays is vital in user-communication [Bodker 1991; Preece 1993]. A good user interface allows the user to focus on the objects currently relevant. Information, whether relating to directions or to the task in hand, should be easy to read, comprehensible and clearly distinguishable from its background. Although it is important that screen layout should be uncluttered, the tendency is away from low-density displays, and towards increased resolution screens on which the information is well structured. Humans exhibit selective attention, filtering out certain perceptions and limiting the ability to do more than one task at a time. Screens should be designed so as to remove distraction, focus attention on important information, and to help the users in multi-tasking. Techniques to achieve these goals include screen partitioning into sub-areas or windows (which may be overlapping), and the use of visual cues and/or icons.

The user should be supported by the availability of an *undo* or *redo* facility, which makes it possible to cancel or change a command or an operation [Thimbleby 1990].

User interface in hypertext

Discussing user interfaces in the context of hypercourseware, where the many levels of interaction cause added complexity, Siviter & Brown [Siviter 1991] also advocate the use of existing standard interface techniques, rather than the creation of obscure new approaches. The main challenge in user interface design for hypertext is how to achieve consistency of style across the range of content, yet how, when appropriate, to develop innovative interface ideas which remain within the spirit of the guidelines.

8.7 APPLICATION TO CAI AND TO FRAMES IN PARTICULAR

8.7.1 HYPERTEXT-STYLE CAI

Taking cognisance of current trends in education, hypertext-style software has a useful role to play. Disadvantages of traditional programmed instruction, both as implemented on paper and in behaviouristic CAI, were outlined in section 5.3.4. Problems incurred in the development and use of ITSs were discussed in section 7.4. It has been shown that hypertext supports a mode of learning which contrasts both with programmed instruction and with model-driven intelligent tutoring. The pendulum is swinging towards more pragmatic approaches based on exploratory styles of learning, yet avoiding excessive freedom, the disadvantages of which are given in section 8.2.3.

This author's approach to the *hypertext versus intelligent tutor* debate is a decision not to embark upon labour-intensive student modelling with its associated limits on proactivity, but rather to provide an environment which facilitates learner-initiative, various usage-modes and limited exploration. It should include adequate user support in the form of help facilities and visualization [Lanzing 1994 & section 6.5.5].

Understanding domain relationships is vital in any kind of learning. Hypertext displays relationships explicitly [Barrow 1989a], in that, as a viewer moves between detail and its context, he works tangibly with these relationships. Such zooming in and out in a CAI situation, along with the various views on a topic afforded by hypertext, should improve integration and retention of material.

Section 8.4.4 investigates hypertext in the context of the relationship between learning need and instructional approach. Various aspects are considered and applications to instructional software, particularly hypertext-style software, are listed:

- *Prior knowledge* - review of prior learning and pre-requisites should be available.
- *Situational dynamics* - the content and context should be task-based.
- *Processing depth* - active engagement and task relevance promote deep processing.
- *Learning-by-doing* - hypertext for educational purposes should include exercises and other user-activities.
- *Learning styles* - the multi-perspective selection features of hypertext facilitate foster the user's ability to choose material and mode according to personal preference and stage of study.
- *Metacognition* - hypertext facilitates metacognition by allowing the learner to plan his own instructional experience.

The goal of the FRAMES practice-environment is to offer the learner a variety of activities and interaction modes, in keeping with the above instructional approaches. Although the prototype would not qualify as a hypertext system, its desired navigation structure is **in harmony with the hypertext style.** Hammond [Hammond 1992a, p. 56] advises:

> "Hypertext seems particularly suited to learning situations where flexibility in sequencing of exposure to materials and in the choice of learning activity is required".

FRAMES should meet these criteria. Flexibility in learning style is also required, in the form of different modes; different users should find them appropriate for their personal learning style, and the same user may change mode for different occasions and stages of learning.

Hammond [Hammond 1992a; Hammond 1992b] proposes a cubic framework for analyzing hypertext-based learning systems. Following his observation that more is known about providing appropriate environments for learning than about the details of learning processes, he presents a three-dimensional framework of principles that contribute towards effective learning. *Control* refers to the degree of learner-control versus program-control. If the kind of learning requires thought, then a measure of restriction and guidance may well be integrated into the freedom of action. *Engagement* relates to the nature of processing done by the learner - active or passive. Unless a learner possesses the internal intrinsic motivation to apply himself assiduously to the task, some form of external motivation, or else an inbuilt learning challenge could be offered. This occurs automatically when the system provides a range of learning activities outside of strict hypertext, such as demonstrations and problems to solve. The level of *synthesis* indicates whether the learner creates material and relationships, or merely observes them. Allowing a novice learner access to a full authoring system, however, is unlikely to be productive and would also create problems in assessment.

Figure 8.3 shows Hammond's framework.

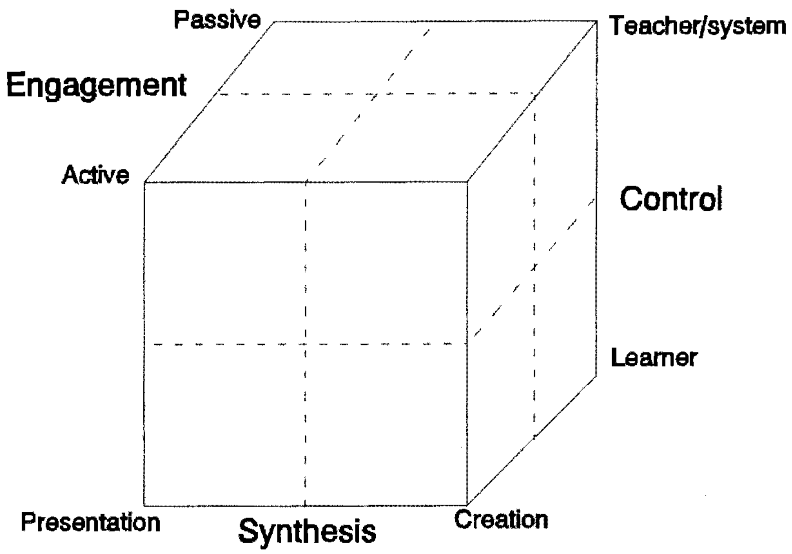


Figure 8.3 Hammond's Assessment Framework
[Hammond 1992a, p. 65]

8.7.2 USABILITY AND USER INTERFACE IN CAI

Section 8.6 overviews human-computer interaction, usability criteria, and the kinds and requirements of a user interface.

To achieve a high **usability level**, educational software should be learnable so as not to frustrate the student, effective, efficient, and robust to perform its tasks accurately on a continuing basis. Learnability can be enhanced by the provision of training, manuals, on-line and off-line help. In a situation of distance-learning, training courses as such are impossible, thus placing the onus on written and on-line training materials. The functionality and versatility should be such that it meets the learner's needs, both explicit and implicit, accurately and effectively. Robustness is of great importance in a distance-education situation, where a system-crash would cause severe frustration.

Application of the principles for a good **user interface** would ensure a simply functioning user interface with commands and interaction modes that are consistent across the program, effective system-response, aesthetic screen presentation, a crashproof system, and availability of help facilities.

Segregation of the screen into functional areas or windows [Bodker 1991; Preece 1993] fosters an environment which presents multiple facets or differing aspects of a topic. It enables presentation of large quantities of information without the appearance of screen clutter, particularly when a small, yet legible font is used. In the instructional context, screen subdivision is of great utility, since it permits form to follow function, as the various commands, components and utilities acquire their own consistent locations.

For reasons given in section 9.4.1, the name "FRAMES" was selected for the accompanying CAI prototype. Subsequently the researcher found that Jonassen in defining hypertext units [Jonassen 1989, p. 7], used the same term: "these nodes, cards, or units may also be referred to as *frames*". This was an apt confirmation, since the researcher had been concerned about possible confusion with the term as used in knowledge representation (section 7.2.2), and its use in conventional CAI, where it refers to a fixed screen layout. The *frames* to be used in FRAMES are not full-screen displays, but juxtaposed components, combining on-screen to give a window-based appearance.

In the highly visual environment of current software, it is clear that graphics should be used both in instructional and in control aspects. Similarities to standard interface techniques are to be recommended, particularly where the user can be assumed to be familiar with current software systems.

8.8 CONCLUSION

Chapter Seven stated that the strength of an ITS lies in its **individualization by adaptivity,** but that its weakness is lack of learner participation in the selection of instructional material. Designers of ITSs should reconsider the roles of expert- and student-models, to what extent the system should generate new examples, and how and when it should intervene [Keller 1987]. Hypertext, on the other hand, gives control of the learning process to the learner, **individualization by learner-control.** In the hands of an academically mature learner, one who can independently plan and organize his own learning, hypertext-style control can be a powerful tool.

The ultimate question is whether hypertext is effective in meeting instructional goals. The most natural mode of hypertext use is browsing. Proponents of hypertext assume that the freedom to select material, exploration within a conducive environment and the self-discovery of relationships, with the consequent formation of integrated structures, result in effective learning. However it is uncertain whether unconstrained browsing and mere information retrieval can support deep processing and result in meaningful learning [Jonassen 1992; Hammond 1992a; Hammond 1992b]. While pure hypertext and hypermedia have an inimicable role to play as reference material, a solely read-and-look format is inadequate for full and formal learning. Active **learner-control** alone is insufficient; there must be **learner-participation** in the activities or components presented.

A general problem with most CAI software systems is that their approach supports only a subset of the desirable learning activities for that particular domain. Under varying circumstances it may be appropriate for learners to browse, to interact, to explore a microworld, to follow a fixed route through a linear domain, to be guided by an intelligent tutor, to practice subskills, to solve problems, to synthesize materials, or to assess their knowledge. The open philosophy and navigation structure of hypertext makes it a suitable medium to "support a range of learning activities" [Hammond 1992a, p. 55].

Regardless of how appropriate and instructionally beneficial the activities in a CAI package may be, the system cannot realize its potential unless it meets the various usability criteria. It should also have an interactive user-interface which facilitates use of the system by a range of users, and has a structure compatible with the functions of the software.

REFERENCES - CHAPTER EIGHT

[Alessi 1991] Alessi, S.M. & Trollip, S.R. (1991). *Computer-Based Instruction: Methods and Development.* Englewood Cliffs, N.J.: Prentice Hall.

[Barrow 1989a] Barrow, J. (1989). Hypertext as an Educational Medium. *Proceedings of the First Southern African Conference on Educational Technology.* Pretoria: Human Sciences Research Council.

[Barrow 1989b] Barrow, J. (1989). Structuring Hypertext. *Proceedings of the Fifth South African Computer Symposium.* Johannesburg, South Africa.

[Bevan 1991] Bevan, N., Kirakovski, J. & Maissel, J. (1991). What is Usability? In: Bullinger, H.J. (Ed), *Human Aspects in Computing: Design and Use of Interactive Systems and Work with Terminals.* Amsterdam: Elsevier Science Publishers B.V..

[Bodker 1991] Bodker, S. (1991). *Through the Interface: A Human Activity Approach to User Interface Design.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Bowers 1989] Bowers, D. (1989). The Software Design Document: More than User's Manual. *Educational Technology 29* (12), 15-18.

[Brown 1989] Brown, J.R. & Cunningham, S. (1989). *Programming the User Interface: Principles and Examples.* New York: John Wiley and Sons, Inc.

[Carbonell 1970] Carbonell, J.R. (1970). AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction. *IEEE Transactions on Man-Machine Systems 11* (4), 190-202.

[Chapanis 1991] Chapanis, A. (1991). Evaluating Usability. In: Shackel, B. & Richardson, S.J. (Eds), *Human Factors for Informatics Usability.* Cambridge: Cambridge University Press.

[De Wet 1994] De Wet, L. (1994). *A Comparison of the Usability Properties of Character-based and Graphical-based User Interfaces.* Unpublished MSc thesis, University of South Africa, Pretoria.

[Dick 1991] Dick, W. (1991). An Instructional Designer's View of Construction. *Educational Technology 31* (5), 41-44.

[Dix 1993] Dix, A., Finlay, J., Abowd, G. & Beale, R. (1993). *Human-Computer Interaction.* Hemel Hempstead: Prentice Hall International (UK).

[Görner 1992] Görner, C., Vossen, P. & Ziegler, J. (1992). Direct Manipulation User Interface. In: Galer, M., Harker, S. & Ziegler, J. (Eds), *Methods and Tools in User-Centred Design for Information Technology.* Amsterdam: Elsevier Science Publishers B.V.

[Hammond 1989] Hammond, N.V. & Allinson, L.J. (1989). Extending Hypertext for Learning: An Investigation of Access and Guidance Tools. In: Sutciffe, A. & Macauly, L. (Eds), *People and Computers V.* Cambridge: Cambridge University Press.

[Hammond 1992a] Hammond, N.V. (1992). Learning with Hypertext: Problems, Principles and Prospects. In: McKnight, C., Dillon, A. & Richardson, J. (Eds), *Hypertext: A Psychological Perspective.* Chichester: Ellis Horwood.

[Hammond 1992b] Hammond, N.V. (1992). Tailoring Hypertext for the Learner. In: Kommers, P.A.M., Jonassen, D.H. & Mayes, J.T. (Eds), *Cognitive Tools for Learning.* Berlin: Springer-Verlag.

[Hardman 1988] Hardman, L. (1988). Hypertext Tips: Experiences in Developing a Hypertext Tutorial. In: Jones, D.M. & Winder, R. (Eds), *People and Computers IV.* Cambridge University Press.

[Hasselerharm 1990] Hasselerharm, E. & Leemkuil, H. (1990). The Relation between Instructional Control Strategies and Performance and Attitudes in Computer-Based Instruction. In: Pieters, J.M., Simons, P.R.J. & de Leeuw, L. (Eds), *Research on Computer-Based Instruction.* Amsterdam: Swets and Zeitlinger B.V.

[Jonassen 1989] Jonassen, D.H. (1989). *Hypertext / Hypermedia.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Jonassen 1992] Jonassen, D.H. (1992). Effects of Semantically Structured Hypertext Knowledge Bases on Users' Knowledge Structures. In: McKnight, C., Dillon, A. & Richardson, J. (Eds), *Hypertext: a Psychological Perspective.* Chichester: Ellis Horwood.

[Keller 1987] Keller, A. (1987). *When Machines Teach: Designing Computer Courseware.* New York: Harper and Row.

[Lanzing 1994] Lanzing, J.W.A. & Stanchev, I. (1994). Visual Aspects of Courseware Engineering. *Journal of Computer Assisted Learning 10*, 69-80.

[McKnight 1992] McKnight, C., Dillon, A. & Richardson, J. (Eds) (1992). *Hypertext: a Psychological Perspective.* Chichester: Ellis Horwood.

[Midoro 1991] Midoro, V., Olimpo, G., Persico, D. & Sarti, L. (1991). Multimedia Navigable Systems and Artificial Intelligence. In: Lewis, R. & Otsuki, S. (Eds), *Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education.* Amsterdam: North-Holland.

[Nielsen 1990] Nielsen, J. (1990). *Hypertext and Hypermedia.* Boston: Academic Press.

[Nix 1990] Nix, D. & Spiro, R. (Eds) (1990). *Cognition, Education, Multimedia: Exploring Ideas in High Technology.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Pieters 1990] Pieters, J.M., Simons, P.R.J. & de Leeuw, L. (Eds) (1990). *Research on Computer-Based Instruction.* Amsterdam: Swets and Zeitlinger B.V.

[Preece 1993] Preece, J. (Ed.) (1993). *A Guide to Usability: Human Factors in Computing.* Wokingham: Addison Wesley.

[Ravden 1989] Ravden, S.J. & Johnson, G.I. (1989). *Evaluating Usability of Human-computer Interfaces: A Practical Method.* Chichester: Ellis Horwood.

[Regian 1992] Regian, J.W. & Shute V.J. (1992). Automated Instruction as an Approach to Individualization. In: Regian, J.W. & Shute V.J. (Eds), *Cognitive Approaches to Automated Instruction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Siviter 1992] Siviter, D. & Brown, K. (1992). Hypercourseware. In: Kibby, M.R. & Hartley, J.R. (Eds). *Computer Assisted Learning: Selected Contributions from the CAL91 Symposium.* Oxford: Pergamon Press.

[Spiro 1990] Spiro, R.J. & Jehng, J. (1990). Cognitive Flexibility and Hypertext: Theory and Technology for the Nonlinear and Multidimensional Traversal of Complex Subject Matter. In: Nix, D. & Spiro, R.J. (Eds), *Cognition, Education and Multimedia: Exploring Ideas in High Technology.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Tang 1991] Tang, H., Barden, R. & Clifton, C. (1991). A New Learning Environment Based on Hypertext and its Techniques. In: Lewis, R. & Otsuki, S. (Eds), *Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education.* Amsterdam: North-Holland.

[Thimbleby 1990] Thimbleby, H. (1990). *User Interface Design.* Wokingham: Addison Wesley.

[Venezky 1991] Venezky, R. & Osin, L. (1991). *The Intelligent Design of Computer-Assisted Instruction.* New York: Longman.

[Vockell 1989] Vockell, E. & van Deusen, R.M. (1989). *The Computer and Higher-Order-Thinking Skills.* Watsonville, CA: Mitchell Publishing, Inc.

[Woodhead 1991] Woodhead, N. (1991). *Hypertext and Hypermedia: Theory and Applications.* Wokingham: Addison-Wesley.

CHAPTER NINE

THE FRAMES PRACTICE ENVIRONMENT

Various broad factors which determine the basic characteristics of a piece of instructional software, have been investigated and applied to the prototype implementation, FRAMES. Each factor was considered with relation to the task in hand, and the most appropriate approach/es selected to optimise that factor for a practice environment. Chapters Two to Eight addressed the issues factor by factor and Figure 9.1 shows which aspect/s of CAI each chapter addresses.

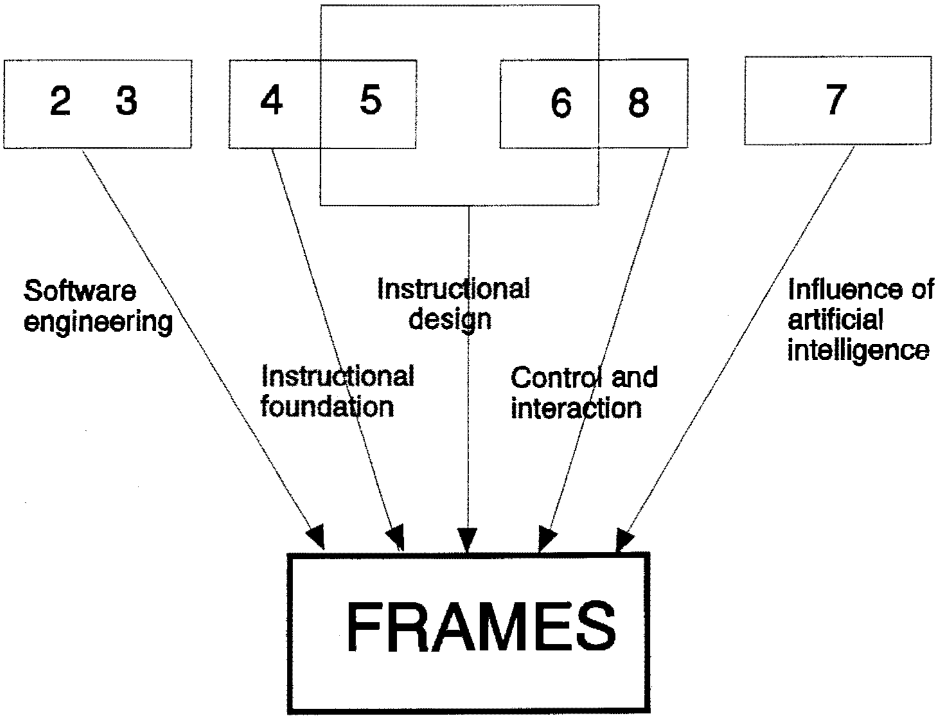


Figure 9.1 The Impact of each Chapter on FRAMES

The subdivision of this chapter, however, is:

- not factor by factor, emphasizing the characteristics of the *product*,
- but step by step through the *process*, according to the phases of an SDLC model.

It shows how the various issues, instructional and computer-oriented, can be integrated into the appropriate phases of a life-cycle model.

9.1 LIFE-CYCLE MODELS

Every software product has a life-cycle spanning the period from its earliest inception to its retirement. Sections 2.1 and 3.4 overviewed various SDLCs in the contexts of different software development methodologies.

9.1.1 A LIFE-CYCLE MODEL FOR CAI

A suitable choice for the life-cycle factor in instructional software was found to be a prototyping model. Lantz advocated rapid prototyping (section 3.4.2) for CAI, in order to enhance communication and to facilitate refinement of requirements and design. Section 2.1.2 listed situations where prototyping is particularly beneficial, four cases of which are most relevant:

- highly interactive programs,
- cases where requirements cannot be precisely specified in advance,
- occasions when the system goes beyond the users experience, and
- situations where the utility of software needs to be demonstrated hands-on.

For ISD *evolutionary prototyping* (section 2.1.2) appears to be more cost-effective than *throw-away*. The initial portions of the envisaged system, once satisfactorily complete, can be used as part of the final product.

9.1.2 THE FRAMES PROTOTYPING LIFE-CYCLE MODEL

FRAMES breaks new ground in the type of software developed by CENSE at Unisa. A highly interactive practice environment entails innovative instructional strategies, which call for new programming techniques. Initially, even the technical feasibility of the desired features had to be assessed. The logical requirements were imprecise, requiring testing. With the premise of *form follows function*, screen layouts need to evolve, rather than adhere to precise specifications. A prototyping life-cycle is the ideal route, since it is conducive to modification of the approach, the strategies, and even of the objectives. Evaluation of a prototype prior to final development of a production model can only enhance the products of analysis and design. Prototyping is much more appropriate for interactive ISD than a development methodology which uses rigid paper-based specifications and design. In the development of FRAMES, a prototype is needed to:

- discover unanticipated problems and misconceptions,
- determine an optimum control strategy,
- consolidate the combined roles of keyboard and mouse in user-control and response,
- develop consistent standards for each mode of doing exercises,
- validate visual and perception aspects in screen layouts,
- check interaction between help facilities and RAM control,
- take cognisance of possible user-input, and
- ensure instructional utility.

In Chapters Two and Three various ISD life-cycle models were investigated. The Booch model for object-oriented design was introduced in section 2.3 (see Figure 2.4) and, in section 2.5.3, was shown to be suitable for ISD. Another candidate is the comprehensive Chen & Shen life-cycle model for ISD (see Figure 3.1 in section 3.4.2). It tends to emphasize specific ID procedures and activities, not all of which are relevant to FRAMES. The most suitable life-cycle model for the development of FRAMES appears to be the Wong prototyping life-cycle (see Figure 3.2 in section 3.4.2). Slightly modified and tuned, it is reproduced as Figure 9.2. There is, however, a major difference between Wong's suggested approach and the methodology used in FRAMES. Wong refers (Chapter Three, section 3.4.2) to "quick prototypes by non-computing laymen", whereas the FRAMES prototype is designed and programmed by professionals.

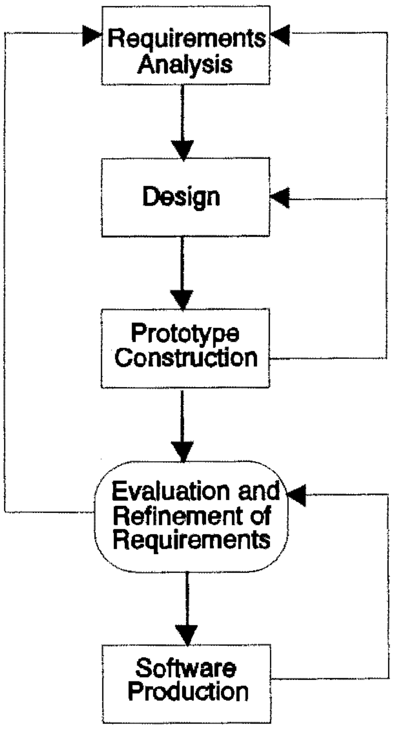


Figure 9.2 The FRAMES Prototyping Life-Cycle Model

The life-cycle is similar to the Booch Model, but includes prototyping explicitly. It emphasizes *evaluation and refinement of requirements* to such an extent that it is perceived as an additional process (appearing in a round-cornered rectangle to distinguish it from the true phases), rather than just being represented by feedback links into basic *requirements analysis.*

The next three sections discuss FRAMES under headings that correspond to the first three steps in Wong's model, namely *requirements analysis*, *design* and *the prototype* respectively. The final step, namely, *software production*, is beyond the scope of this dissertation.

9.2 REQUIREMENTS ANALYSIS

Conventional requirements analysis for instructional software incorporates analysis of the topic and the target group's performance, tending to focus on aspects such as:

- identification of a problem-area in a curriculum,
- definition of aims and objectives of a course and its lessons,
- definition of course structure,
- resources,
- needs analysis of the target group and their performance in the identified area, and
- instructional activities.

This dissertation proposes requirements analysis, but of a different kind. It assumes the above **needs-identification to be complete**, and uses its results as a fixed parameter for the starting point. The type of premise on which a study such as this is founded could be, for example:

- A bank, as the client, has commissioned an educational software developer to produce CBT courseware to train bank staff in the use of a new online teller system.

- A school teacher points out a commonly occurring spelling problem in the use of English as a second language.

- A certain area in a Unisa distance-education module creates problems, evidenced by an excessive number of student-enquiries and by poor examination performance.

Thus the need is identified and delimited, and the target population is fixed. **How then does the designer decide on the most appropriate CAI or CBT to address the problem?**

In section 3.4.2 it was noted how Lantz compared the logical definitions in SE to the instructional objectives in CAI. The instructional objectives and requirements addressed here are not categorical statements of intended learner performance over specific subject matter. The requirements analysis phase for ISD is far broader, embracing the **general characteristics, both instructional and computing-oriented, of a desired instructional experience** to address the specific problem area. Chapters Two to Eight discussed various issues that impact on instructional software. In the requirements analysis step of FRAMES these issues are investigated and the most suitable approaches selected.

9.2.1 THE PROBLEM DOMAIN AND THE PROBLEM STATEMENT

The problem domain, as outlined in Chapter One, is the section on relations in the Unisa module COS101-S, Theoretical Computer Science 1. The target group comprises first year BSc students, studying by distance education, who, in general, experience problems in the analysis of relations with an infinite number of members. The subject matter of FRAMES is relations, their properties, and special kinds of relations. A need is evident for practice exercises that firstly increase the learner's overall familiarity with the domain, and secondly help him gain experience in the mathematical proofs required. Familiarity can be enhanced by a system that facilitates visualization and exposure to examples within the domain. The proofs entail several steps; students require both the experience of studying worked examples and the opportunity to practice. In the realm of mathematical proofs, student-responses cannot be assessed by the conventional techniques used in judging short answers and multiple choice options.

> **The goal is to produce a practice-environment, a kind of *activity box* providing a variety of useful instructional activities, and to design and develop this instructional software system using a software engineering approach.**

9.2.2 MODELLING THE DOMAIN WITH AN ERA DIAGRAM

ERA diagrams (section 2.4.3) are used to show the properties, or attributes, of domain entities, and to model the relationships between entities. ERA Diagrams are used in database design, but are generally applicable to any domain that incorporates inter-relationships. Figure 9.3 shows the most important entities, relationships, and attributes (properties) to be learned and practiced using FRAMES. It sets out the six mathematical properties of the entity *relation*, as taught in COS101-S, and shows other entities relating to the study of relations. The inheritance relationship demonstrates the existence of various special *kinds of relations*, as a specialized subclass of the class, *relation*. The values of the six *property attributes* of *relation* are boolean, i.e. true or false.

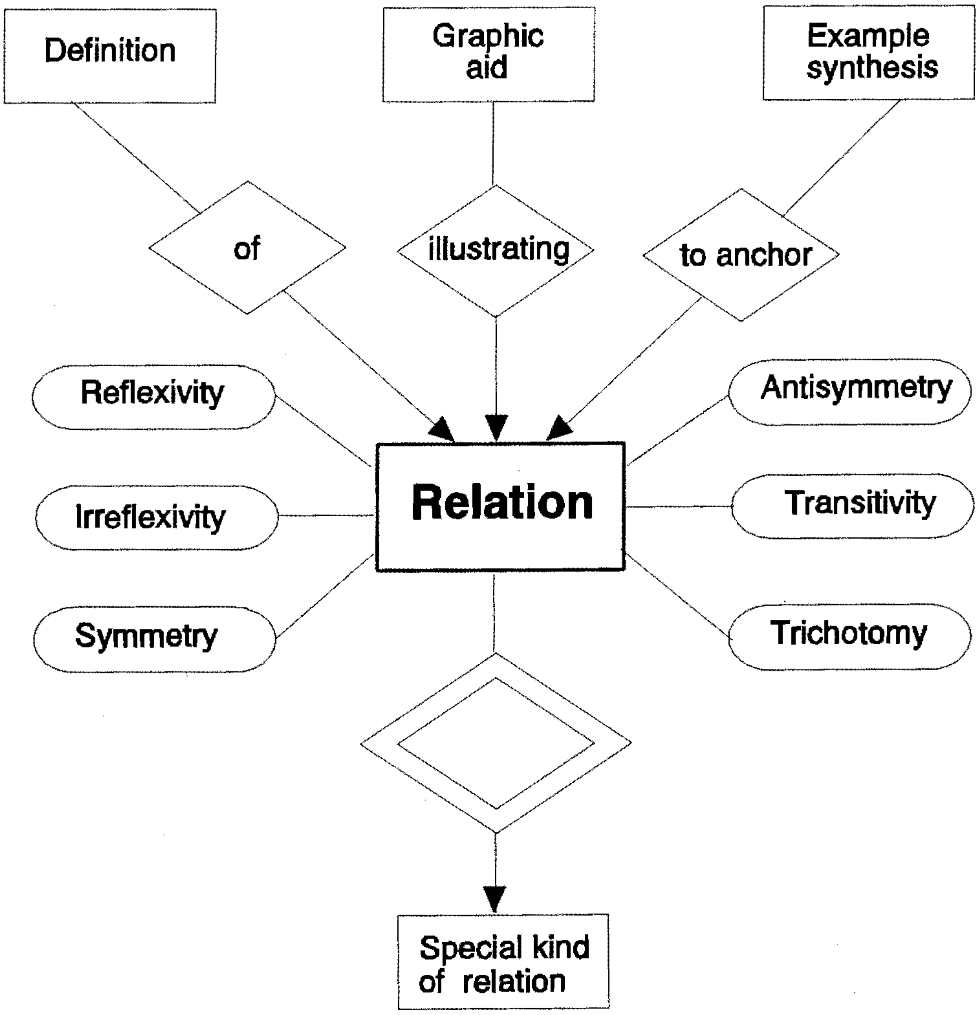


Figure 9.3
Entities and Relationships in FRAMES

9.2.3 INSTRUCTIONAL THEORY AND FRAMES

Chapters Four and Five investigated the factors of cognition and learning and their impact on instruction. This section applies them to FRAMES and extracts appropriate approaches. It outlines the chosen premises of FRAMES and does not go into design details, but, in appropriate places, includes additional information in parentheses to relate the required feature to its implementation in FRAMES, or illustrates salient points by means of screen displays.

Cognition and Learning

Certain aspects of Schiever's Thinking Model (section 4.1.1) could be applied directly to FRAMES. *Concept development* in learners can be fostered by example-synthesis exercises, where entities are identified as members or non-members of a class. The *making of generalizations by projecting experience* should be a fundamental goal of FRAMES, achieved by choosing as examples relations that differ markedly, so that no single algorithm could be used to tackle the same aspect of each problem. Instead, re-application of the same principles would be required, but in varying contexts. The *repetitive encounters* emphasized by Schiever should be inherent in FRAMES, and can be achieved by various modes of doing each exercise and by approaching a topic or aspect from multi-perspectives.

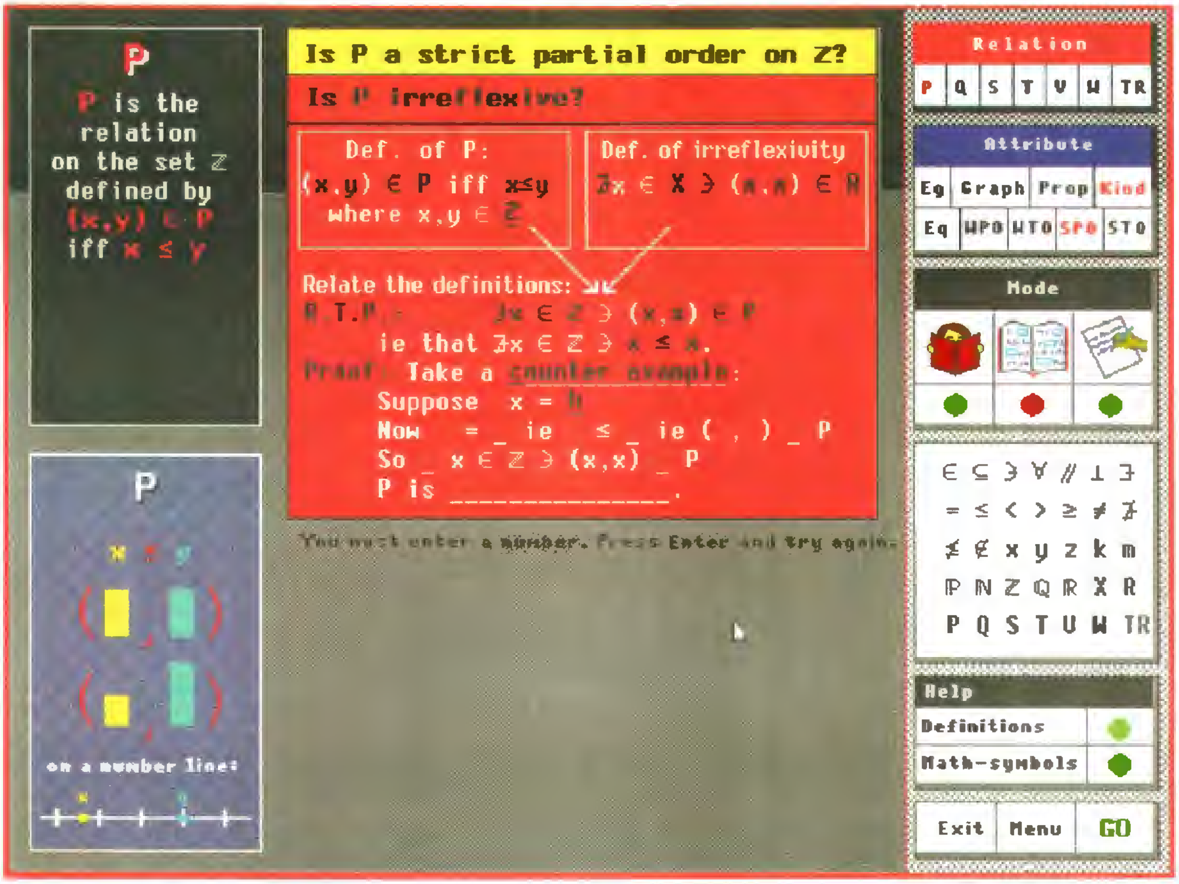
Although not explicitly teaching them, FRAMES should develop Higher Order Thinking Skills (sections 4.1.3 & 4.2.8) within the learner, in particular *metacognition,* by providing an environment conducive to planning his own learning experience. *Problem-solving strategies* should be implicit in the exercises. This can be done by including complete worked-examples, i.e. read-only proofs. *Critical thinking* is to be encouraged by giving the learner the opportunity to consider properties of the relation and develop proofs, both by filling in blanks and by supplying entire steps.

Behaviourism and Mastery Learning

Sections 4.2.4 & 4.2.5 introduced behaviourism and mastery learning, and section 5.3 follows on by relating them directly to instruction, ID and CAI. Although the instructional approach of FRAMES is to be fundamentally cognitive, certain behaviouristic principles are valuable. Fleming's behaviouristic *perception and learning principles* were examined and several are appropriate as instructional requirements. The learner should:

- Attain such *familiarity with basic messages* like definitions that automaticity is achieved (done by the regular presentation of property definitions in each Mode 2 exercise).
- Develop *familiarity in problem-solving techniques* (achieved by Mode 1 which presents complete worked-proofs, and by repetitive encounters due to the three modes of proof presentation).
- Human information processing capacity is limited, so *step-wise strategies* should be used. The command structure and navigation system must be simple, avoiding cognitive overload.
- *Spatial and temporal arrangements influence perception of relationship.* The FRAMES screen layout should permit simultaneous viewing of the definition and visual aid along with a problem-solving exercise. The material is to be presented in a well-structured manner that supports knowledge acquisition.
- It is important to *indicate relationship* between associated objects. For example, arrows can be used to facilitate the contextualization of a generic definition into the current relation (done in Mode 2).
- *Change in stimulation* is necessary for sustained attention. This is up to the learner who should be able to change the content or approach any time he likes. Another aspect of change is variety in the examples and exercises.
- *Situational support* for problem-solving is necessary to emphasize crucial definitions and reveal important relationships (limited cuing occurs due to the line lengths supplied in Mode 2 exercises and the proof structure supplied; cues are faded in Mode 3, where the learner works more independently).
- The interactive question-response-judgment process should be characterized by *contiguity* - having filled in a blank or typed a complete line, the learner should receive immediate feedback. In the case of incorrect or inappropriate responses a second chance ought to be given before the remedial feedback appears.
- *Examples and non-examples* should be demonstrated (since the tests for all six properties are to be done on every relation, whether or not the property holds true, a good selection of examples and non-examples will be shown).
- However, out of congruence with the fundamental stance of behaviourism, there are to be no scores, rewards or reinforcement other than feedback. Pre-tests and post-tests are deliberately excluded.

Several of these features are clearly illustrated in Screen Displays 9.1 and 9.2.



P

P is the relation on the set ℤ defined by (x,y) ∈ P iff x ≤ y

Is P a strict partial order on Z?

Is P irreflexive?

Def. of P:
(x,y) ∈ P iff x≤y
where x,y ∈ ℤ

Def. of irreflexivity
∄x ∈ X ∋ (x,x) ∈ R

Relate the definitions:
R.T.P.: ∄x ∈ ℤ ∋ (x,x) ∈ P
ie that ∃x ∈ ℤ ∋ x ≤ x.
Proof: Take a counter example:
Suppose x = h
Now = _ ie ≤ _ ie (,) _ P
So _ x ∈ ℤ ∋ (x,x) _ P
P is ______________.

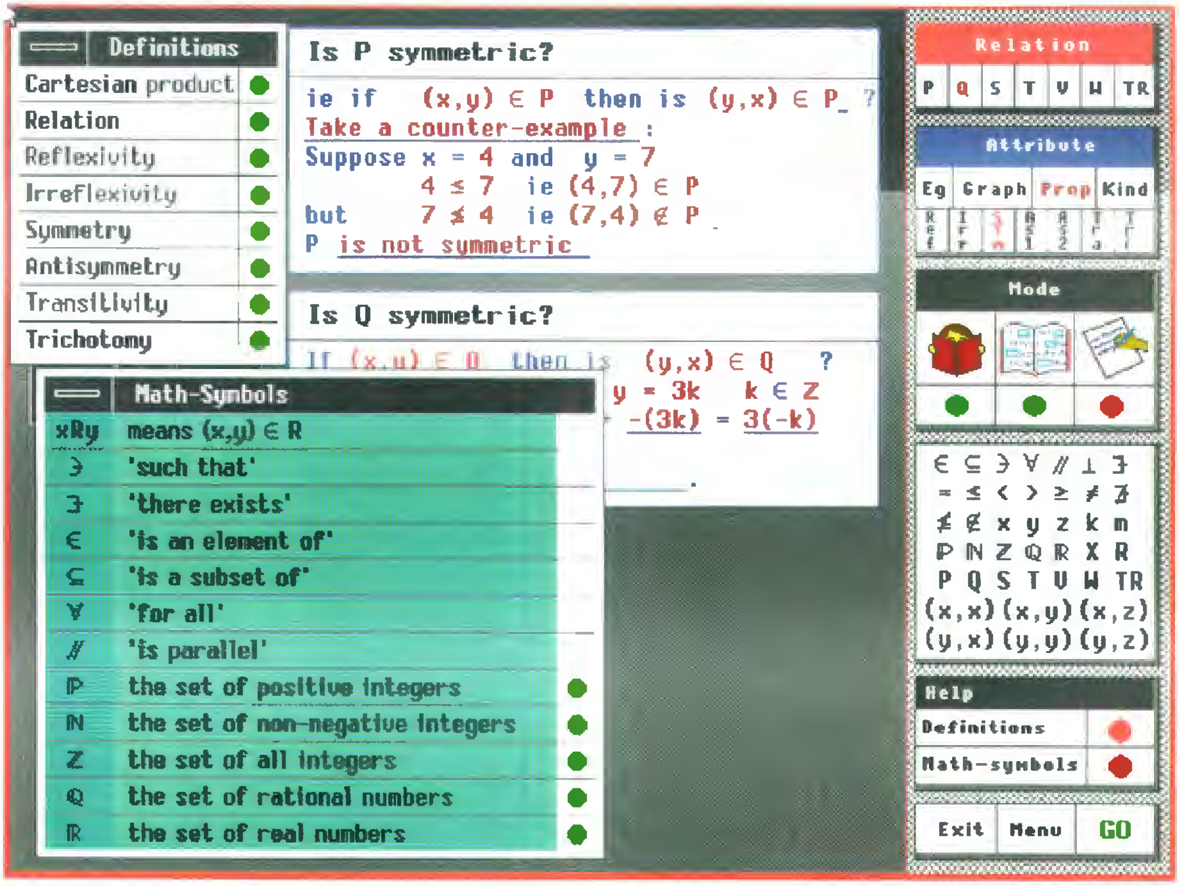
You must enter a number. Press Enter and try again.

Screen Display 9.1

The FRAMES screen layout permits *simultaneous viewing* of the definition and visual aid along with the component currently selected. The material is chunked, for example, the definitions of “P” and “irreflexivity” at the head of this Mode 2 exercise appeared in a *step-wise* manner before the appearance of the actual fill-in-the-blank proof.

Relationship is emphasized by arrows and by the instruction, “Relate the definitions”, which encourages the student to *contextualize* the general definition of irreflexivity.

Feedback is geared to handle inappropriate, as well as incorrect, entries. In this case an alphabetic character is rejected as being illegal; the learner is told, “You must enter a number”. The colour of feedback has subsequently been changed to white.



Is P symmetric?

ie if (x,y) ∈ P then is (y,x) ∈ P_ ?
Take a counter-example :
Suppose x = 4 and y = 7
4 ≤ 7 ie (4,7) ∈ P
but 7 ≰ 4 ie (7,4) ∉ P
P is not symmetric

Is Q symmetric?

If (x,y) ∈ Q then is (y,x) ∈ Q ?
y = 3k k ∈ Z
-(3k) = 3(-k)

Math-Symbols	
xRy	means (x,y) ∈ R
∋	'such that'
∃	'there exists'
∈	'is an element of'
⊆	'is a subset of'
∀	'for all'
//	'is parallel'
ℙ	the set of positive integers ●
ℕ	the set of non-negative integers ●
ℤ	the set of all integers ●
ℚ	the set of rational numbers ●
ℝ	the set of real numbers ●

Screen Display 9.2

Situational support for problem-solving is available from:

1. *Help Components*
   Mouse-clicking on 'Definitions" and "Math-symbols" on the right produces overlay-menus at top and bottom left respectively. Selecting an option on the right highlights the appropriate switch or character/s in red - compare with Screen Display 9.1. Further clicking on these submenus provides overlay elaboration of the selection (shown later in Screen Displays 9.8 & 9.16).

2. *Cuing*
   Mode 2 exercises *cue* by providing a blank with the appropriate number of spaces for the correct answer (see Screen Display 9.1). Mode 3, shown in this Screen Display, fades the cue by presenting continuous lines for the student's answers.

Non-examples play an important role. In both Screen Displays 9.1 and 9.2, counter-example proofs are necessary.

Cognitive learning and constructivism

Sections 4.2.6 & 4.2.7 introduced cognitive learning and constructivism, and section 5.4 followed on by relating them directly to instruction, ID and CAI. With reference to learning and instructional factors, the better approach for FRAMES seems to be cognitive science, implemented with a constructivist flavour, thus viewing thinking and learning as human information processing (sections 4.1.4 & 4.2.1). Important aspects are the *integration of new information with old*, and the *limitations on STM*. The structural templates for problem solving and the help facilities in FRAMES should facilitate the former. STM limits can be addressed by chunking and by a simple command structure. Desirable cognitive features are:

- *Active participation* - the learner must play a highly active role, both in control of his practice experience, and in participation in the exercises.

- *Integration of new with prior knowledge* - the learner's perception of the new should occur in a context of ancillary support. This should foster construction of internal schemata to facilitate comprehension and recall.

- *Accretion, tuning and restructuring* - these should occur as the learner works through FRAMES (moving between modes when his coaching needs vary, changing from one problem relation to another, and tackling the composite problems presented under the option *kinds of relations*).

- *Transfer of skills from one setting to another* - is likely to occur as a consequence of the three processes above.

- *Integrated testing* - there is to be no formal testing or scoring, but the student ought to apply his knowledge and skills continuously in problem-solving.

- *Anchored instruction* - to relate abstraction to reality (can be achieved by the permanent on-screen appearance of the current relation's definition and by graphic aids to promote visual perception of a problem domain).

- *Learning by experience* - there should be visual aids and integration of examples (as stated above there are graphic aids; there are also exercises involving synthesis of concrete examples).

- *Strategies to develop general intellectual skills* - should be fostered by associational and mapping techniques explicitly and implicitly incorporated in FRAMES.

- *Real world relationship* - should be clearly set out (done by the visual aid and by example synthesis).

- *Multiple presentation of information* - to expedite comprehension and recall (by different instructional modes).

- *West's cognitive strategies* (section 5.4.2) - four of these, in particular, should be included in FRAMES:

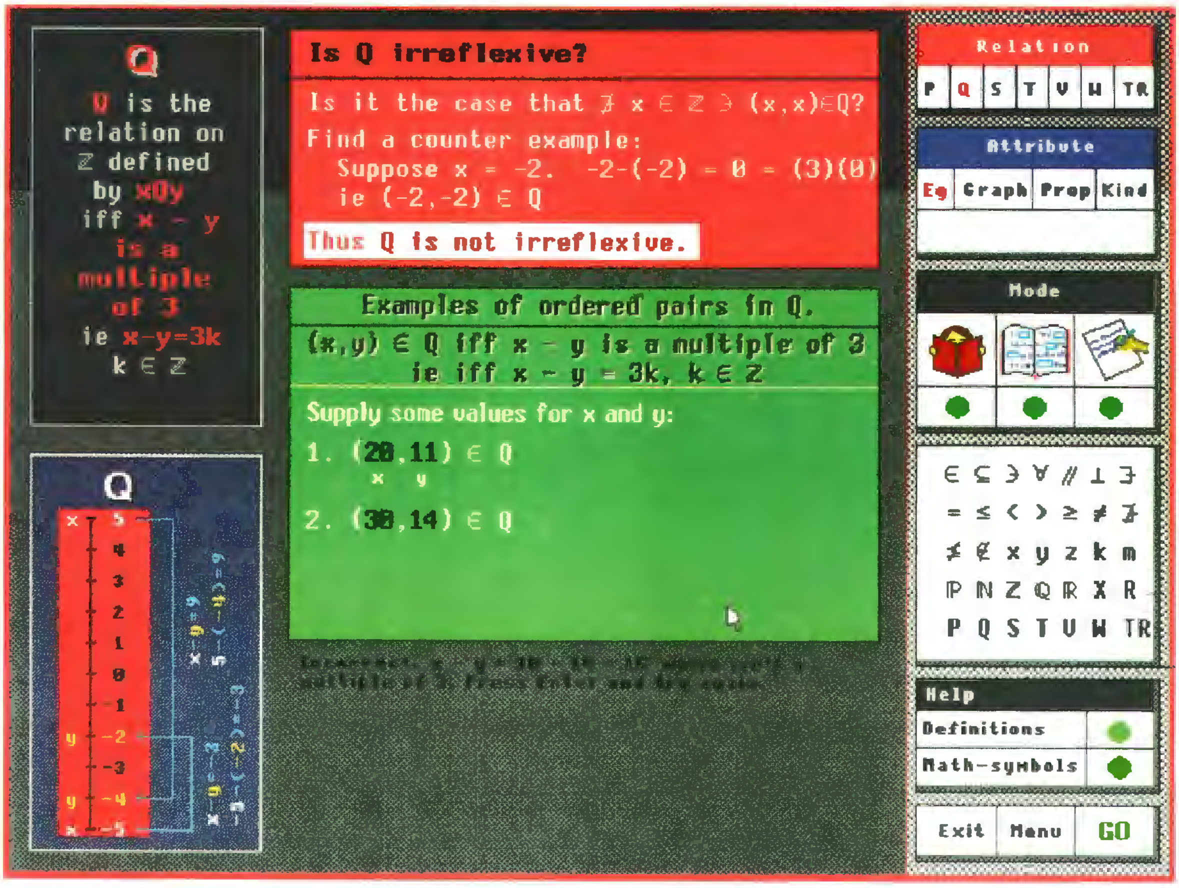
    1. *Rehearsal*, as the same material is approached in various ways and from different perspectives.

    2. *Concept mapping*, to show relationships.

    3. *Imagery*, in diagrams and graphic aids to illustrate the concepts.

    4. *Mnemonics*, to aid recall and simplify usage (occurs in the letters **R A M,** used in the control structure and described in section 9.4.1).

Several of these features are clearly illustrated in Screen Displays 9.3 and 9.4.



Q

Q is the relation on ℤ defined by xQy iff x − y is a multiple of 3 ie x−y=3k k ∈ ℤ

Is Q irreflexive?

Is it the case that ∄ x ∈ ℤ ∋ (x,x)∈Q?

Find a counter example:

Suppose x = −2. −2−(−2) = 0 = (3)(0)

ie (−2,−2) ∈ Q

Thus Q is not irreflexive.

Examples of ordered pairs in Q.

(x,y) ∈ Q iff x − y is a multiple of 3 ie iff x − y = 3k, k ∈ Z

Supply some values for x and y:

1. (20,11) ∈ Q

2. (30,14) ∈ Q

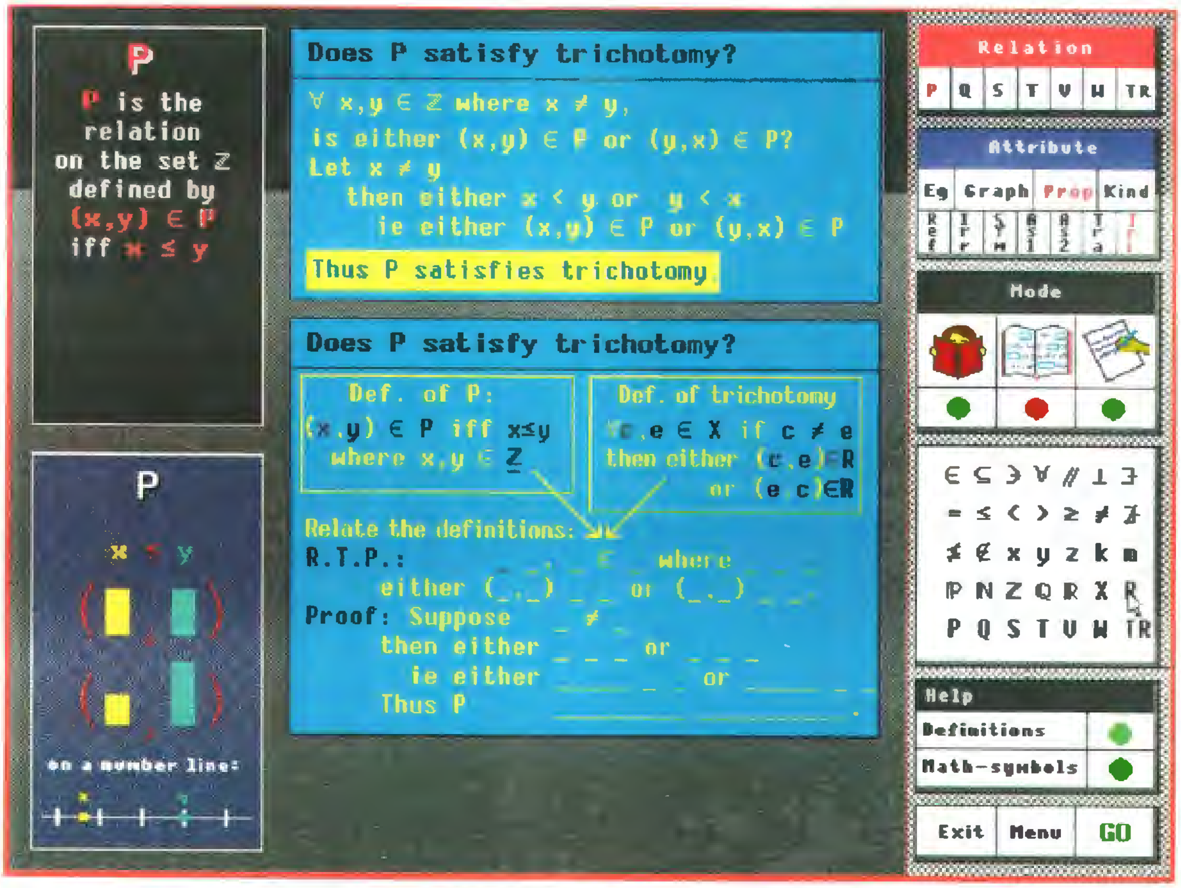
Incorrect. x − y = 30 − 14 = 16 which isn't a multiple of 3. Press Enter and try again.

Screen Display 9.3

The learner plays an *active role* both in control and in participative problem-solving. The screen above shows how, empowered by user-control, the learner selected three components relating to **Relation Q.** He chose the **Attributes**:

- visual aid *Graph*, which along with the definition, *anchors* the relation by relating it to reality.
- a *Prop* (property), in this case *irreflexivity*, and
- *Eg*, the opportunity of synthesizing concrete examples of Q.

The most recent selection set is highlighted in red in the right-hand control area. A **Mode** selection is not highlighted, because mode is not applicable to a graphic aid or an example. As the user *applies knowledge* and *learns by experience* by setting up ordered pairs for Q, where the first and second components differ by a multiple of three, his efforts are met by appropriate feedback. Example synthesis helps a learner to establish a *real world relationship*.



P

P is the relation on the set ℤ defined by (x,y) ∈ P iff x ≤ y

Does P satisfy trichotomy?

∀ x,y ∈ ℤ where x ≠ y,
is either (x,y) ∈ P or (y,x) ∈ P?
Let x ≠ y
then either x < y or y < x
ie either (x,y) ∈ P or (y,x) ∈ P
Thus P satisfies trichotomy

Does P satisfy trichotomy?

Def. of P:
(x,y) ∈ P iff x≤y
where x,y ∈ ℤ

Def. of trichotomy
∀c,e ∈ X if c ≠ e
then either (c,e)∈R
or (e,c)∈R

Relate the definitions:
R.T.P.: _ , _ ∈ _ where _ _ _
either (_,_) _ _ or (_,_) _ _
Proof: Suppose _ ≠ _
then either _ _ _ or _ _ _
ie either _____ _ _ or _____ _ _
Thus P _____ .

Screen Display 9.4

Tuning and restructuring occur as the learner tackles the same problem, changing from **Mode 1**, where a complete proof is demonstrated step by step, to **Mode 2**, where a skeleton is provided and he fills in the blanks. This also illustrates *multiple presentation* of the same proofs.

Transfer occurs when the learner can successfully handle the proof for the same property test in a different setting, i.e. with a different relation.

This display illustrates two further points:

1. The student decided to be individualistic, and used **c** and **e** in his definition of trichotomy instead of the usual **x** and **y**. His alternative was, however, internally consistent and was acceptable to the judgement procedure.

2. The Mode 2 proof skeleton is shown at an early stage.

All four of West's *cognitive strategies* listed under the heading "Cognitive learning and constructivism" in section 9.2.3 are used in this screen display.

9.2.4 ARTIFICIAL INTELLIGENCE AND FRAMES

This section discusses the approach to FRAMES with relation to some of the factors considered in Chapter Seven on Artificial Intelligence. FRAMES is in no way intended to be an ITS or ICAI system, but it should embody a limited theoretical knowledge base. In section 7.6.3 Lippert's suggested characteristics, namely, adaptability, individualization and interactivity, were mentioned. FRAMES is not adaptable, but it is highly interactive and strong on individualization. It will not be inherently intelligent, but can be perceived by the learner as a **source of intelligent support.**

Example practice

The idea is to expose the learner to wide diversity by judicious selection of the seven relations presented as learner-exercises in FRAMES. Relations are based on a mathematical formulae or relationships, which should differ to such an extent that the property tests cannot be solved by a general algorithm. This works against possible incorporation of a problem-solving expert model, and necessitates explicit encoding of each proof or disproof in all three modes, although a **generic problem-solving format can be supplied**. The situation is a little different for the tests for kinds of relations where *courseware abstraction*, as described in section 7.7.2, should be applied to the composite skills. Each test for a kind can be handled within the context of a general parameterized structure that "knows" and automates the overall process, accessing the concrete exercises already coded as single property tests.

The adult learner and FRAMES

Section 7.7.1 discusses the case of the adult learner. FRAMES, with its audience of tertiary-level students, should offer the user meaningful activities and choice between redundant diversity, diverse both in terms of content and strategy. Help should be available on request, but no advice or explicit guidance. (The cuing and prompting that occurs in Mode 2 exercises, in the form of arrows and line-lengths, can be considered a form of implicit guidance.) The adult learner may not appreciate a form of instruction that guides his problem-solving strategy into close convergence with a fixed model, preferring rather to develop his own methods. Alternative correct responses to questions and entries within proofs must be acceptable. This calls for considerable effort in designing judgement and feedback modules.

In general the adult learner does not appreciate gimmicks and graphics other than those related to elaboration of the material. FRAMES should have a purely functional approach, with no name personalization or theme characters.

9.2.5 SUMMARY OF REQUIREMENTS ANALYSIS

Contribution of SE:

Use of an ERA model to represent domain entities, their attributes, and relationships between them.

Stimulation of cognitive thinking:

An environment that

- develops concept awareness,
- fosters generalization,
- presents repetitive encounters,
- instils metacognitive skills.

Aspects of behaviourism:

Learner should

- achieve automaticity in the basics.

System should

- use step by step strategies,
- set visual aid alongside text,
- ensure change in stimulation,
- provide situational support,
- use examples and non-examples.

Aspects of cognitive science:

System should promote

- active participation,
- integration of new with old,
- accretion, tuning and restructuring,
- generalization and transfer,
- integrated testing,
- anchored instruction,
- experiential learning,
- multi-perspective presentation,
- imagery,
- mnemonics.

Aspects of artificial intelligence:

Environment should include

- limited knowledge base,
- intelligent support for learners.

9.3 DESIGN

9.3.1 DESIGN GOALS IN FRAMES

FRAMES breaks new ground in the development of instructional software at Unisa:

- It is not a tutorial. It assumes a subject-matter grounding from the textbook, the study guide, and the CAI tutorial RELATIONS, and in no way sets out to re-teach the basic definitions and principles.

- Help facilities are, however, accessible in the form of on-line reference to relevant definitions, rules, and elaboration of mathematical symbols.

- There is no underlying sequence either explicitly or implicitly.

- There is an inherent screen layout in terms of consistent functional areas for certain objects, such as the control buttons and the definition of the current relation, but there are no fixed combinations of on-screen units; a variety of objects are available as options, and the student makes his own selection in his own sequence, thus constructing a screen comprising his own chosen set of components. The selected exercises appear in the central portion and a step-wise scrolling mechanism moves the topmost exercise off-screen, as new exercises move in underneath.

- Responses, particularly mathematical symbols, may be clicked in via a mouse.

FRAMES is a **practice environment**, making available to the student a variety of features and problems from the *relations* section of COS101-S. A student may select components from a variety of instructional transactions:

1. Synthesis and keying in *examples of ordered pairs* as members of the relation selected as the current problem. This is an exploratory facility, whereby the student can check whether his intuitive understanding of the relation is correct.

2. Viewing a *graphic aid* as a visual representation of the relation.

3. Viewing or doing exercises, in the form of mathematical proofs, to determine whether a relation satisfies a certain selected *property*.

4. Similar subsections of several varying relations (i.e. the same property-test on different relations). Sequential analysis of the same property across different relations consolidates the learner's grasp of that property.

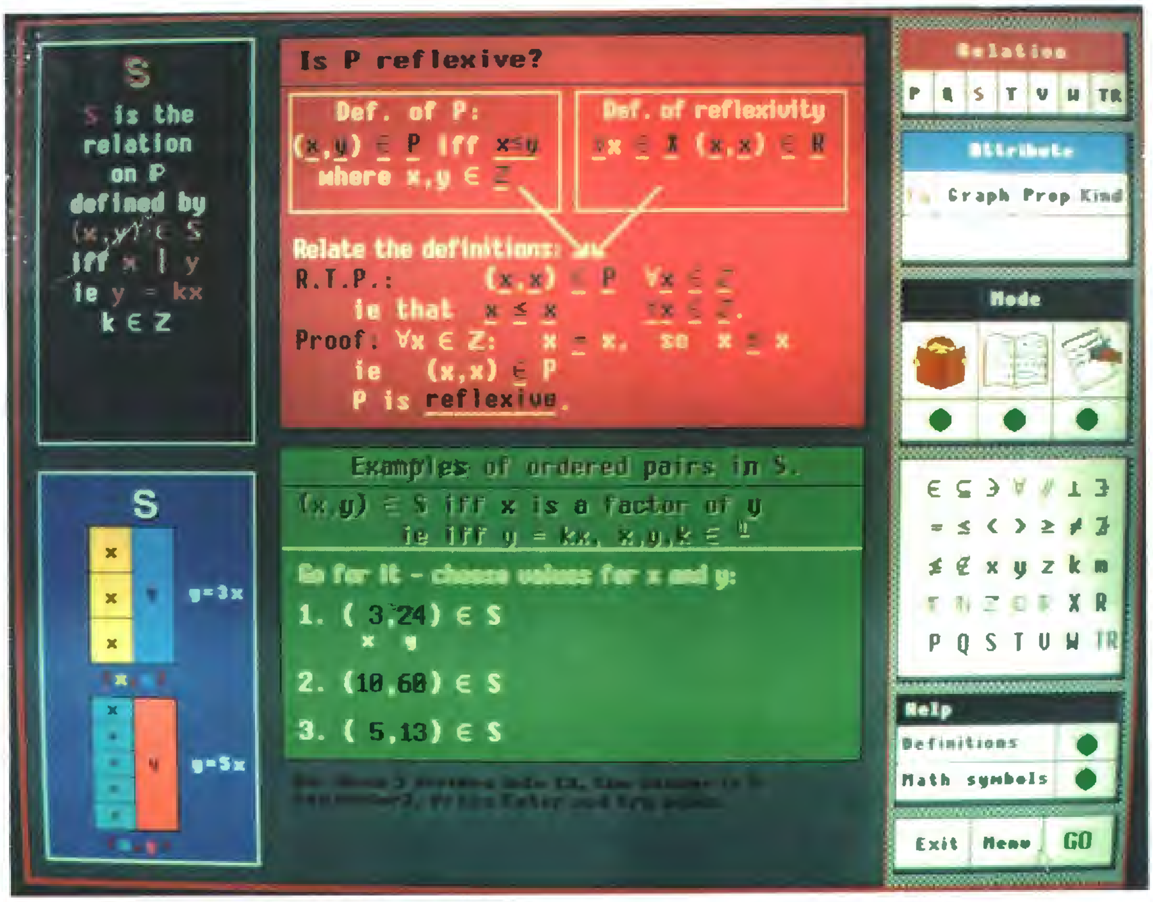
5. Several subsections of one relation (i.e. different property-tests on the same relation). Comprehensive analysis of a relation deepens understanding of that relation.

6. Similar subsections of one relation from multi-perspectives (i.e. the same property of the same relation, done in different modes). When a practice mode follows directly after a presentation mode, the former can be used as a model.

7. Composite analysis of a relation, determining whether it is of a particular *kind* by testing a set of specific properties.

   Thus the student can select whether to practice single components of tasks or more complex integrated problems. The exercises mentioned in points 3 - 7 may be done in three different modes of approach, offering varying degrees of support. It is even possible to change modes during the composite test for a *kind*, described in point 6. In section 3.4.2 it was noted how Lantz compared the physical definitions in SE to the instructional strategies in CAI. Modes, or instructional strategies were investigated in sections 4.2.10, 6.6.2 and 6.6.3. The three modes of FRAMES cover the same content, but in different ways.

The approaches listed in points 1 - 7 above are implicitly portrayed by previous Screen Displays, and are explicitly illustrated in Screen Displays 9.5 - 9.8.



S

S is the relation on P defined by $(x,y) \in S$ iff $x \mid y$ ie $y = kx$ $k \in Z$

Is P reflexive?

Def. of P: $(x,y) \in P$ iff $x \leq y$ where $x,y \in Z$

Def. of reflexivity $\forall x \in X \ (x,x) \in R$

Relate the definitions:

R.T.P.: $(x,x) \in P \ \forall x \in Z$ ie that $x \leq x \ \forall x \in Z$.

Proof: $\forall x \in Z$: $x = x$, so $x \leq x$ ie $(x,x) \in P$ P is reflexive.

Examples of ordered pairs in S.

$(x,y) \in S$ iff x is a factor of y ie iff $y = kx$, $x,y,k \in Z$

Go for it - choose values for x and y:

1. $(3,24) \in S$
2. $(10,60) \in S$
3. $(5,13) \in S$

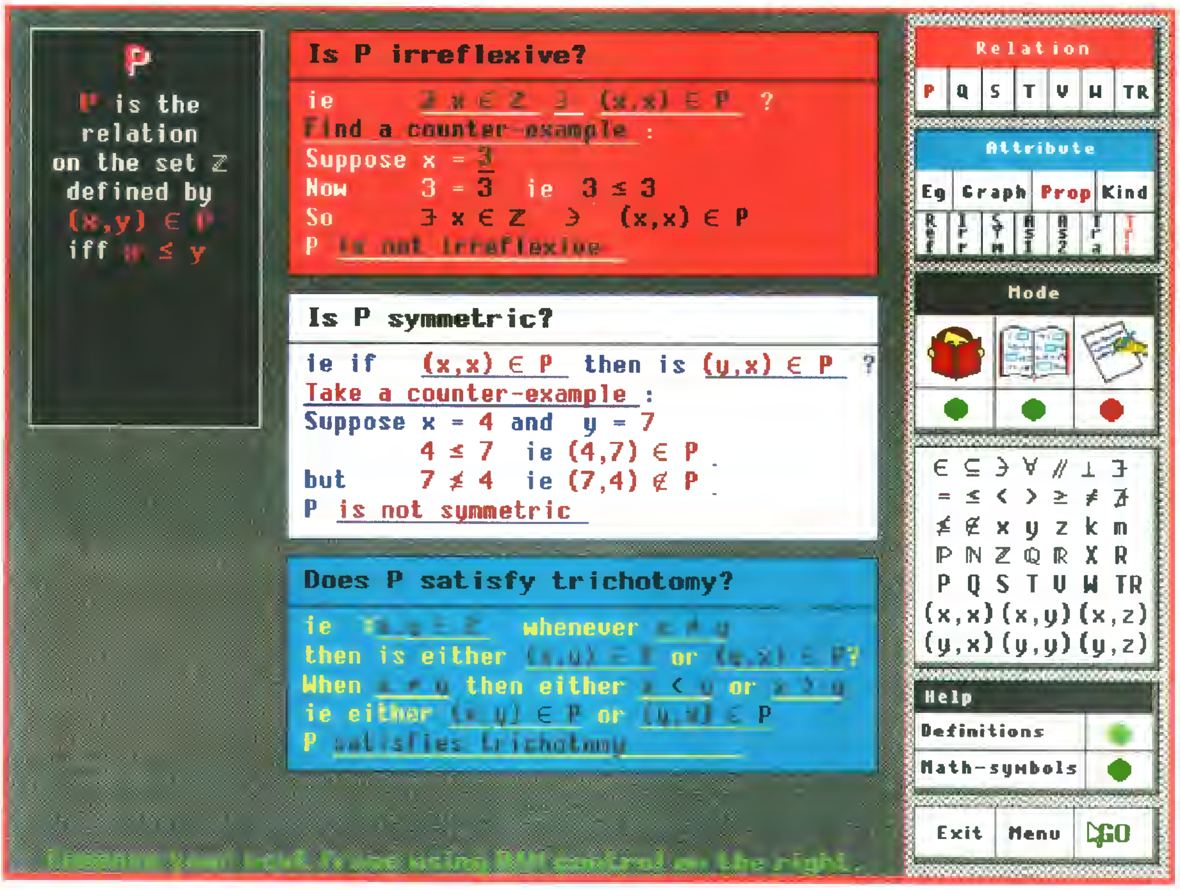
Screen Display 9.5

This screen shows several interesting features:

- Once again the user is tackling example synthesis (point 1, section 9.3.1), this time on relation **S**, where y is a multiple of x. The system processes the student's entry by doing the appropriate calculation itself, and providing *intelligent support* by well-designed feedback.

- The graphic aid (point 2, section 9.3.1) helps to *anchor* the abstract formula in reality by *visual portrayal* of the concepts of multiple and factor.

- The first instructional transaction shown on the screen is the property test to determine whether Relation **P** satisfies reflexivity (point 3, section 9.3.1). Thus this screen depicts frames relating to two different relations, although not analysis ofthe same property across them, as advised in point 4 of section 9.3.1 and shown in Screen Display 9.2.



```
P
P is the
relation
on the set Z
defined by
(x,y) ∈ P
iff x ≤ y
```

```
Is P irreflexive?
ie        ∄ x ∈ Z ∋ (x,x) ∈ P ?
Find a counter-example :
Suppose x = 3
Now      3 = 3  ie  3 ≤ 3
So       ∃ x ∈ Z  ∋  (x,x) ∈ P
P is not irreflexive
```

```
Is P symmetric?
ie if   (x,x) ∈ P  then is (y,x) ∈ P ?
Take a counter-example :
Suppose x = 4 and  y = 7
          4 ≤ 7  ie (4,7) ∈ P
but       7 ≰ 4  ie (7,4) ∉ P
P is not symmetric
```

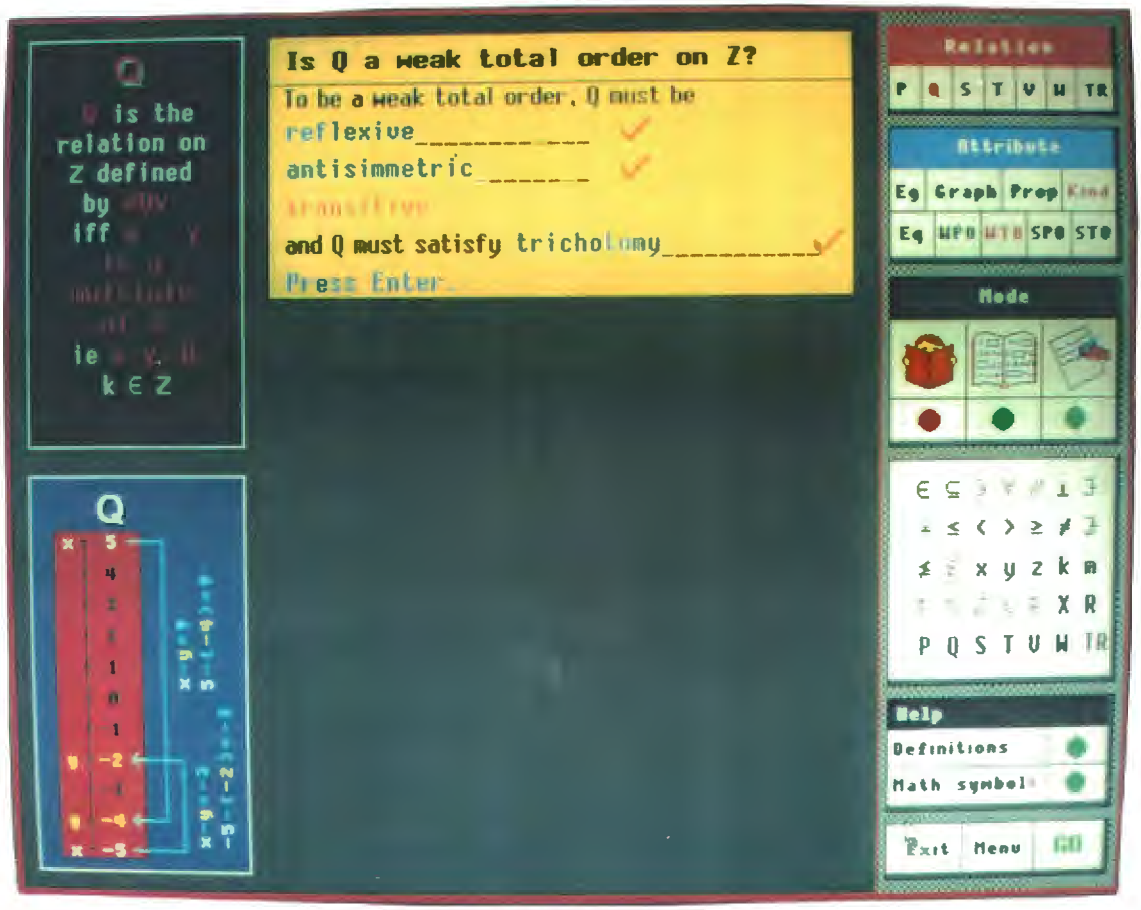
```
Does P satisfy trichotomy?
ie  ∀x,y ∈ Z  whenever x ≠ y
then is either (x,y) ∈ P or (y,x) ∈ P?
When x ≠ y then either x < y or x > y
ie either (x,y) ∈ P or (y,x) ∈ P
P satisfies trichotomy
```

Continue your task. Prove using RAM control on the right.

Screen Display 9.6

Point 5 in section 9.3.1 refers to analysis of several subsections of one relation, i.e. different property tests on the same relation. This is shown on relation **P** above, where the Attribute *Prop* is selected in the right-hand control area. From the property submenu which appears below, three different property tests were chosen in Mode 3. Red highlighting in the control area shows the set currently selected, i.e. which Relation, which Attribute, and which Mode. It is possible to change modes between proofs (point 6 in section 9.3.1) as was shown in Screen Display 9.4.

Notice that the user opted to do these exercises without the support of a graphic aid.



Is Q a weak total order on Z?

To be a weak total order, Q must be

reflexive__________ ✓

antisimmetric_______ ✓

transitive

and Q must satisfy tricholomy____________✓

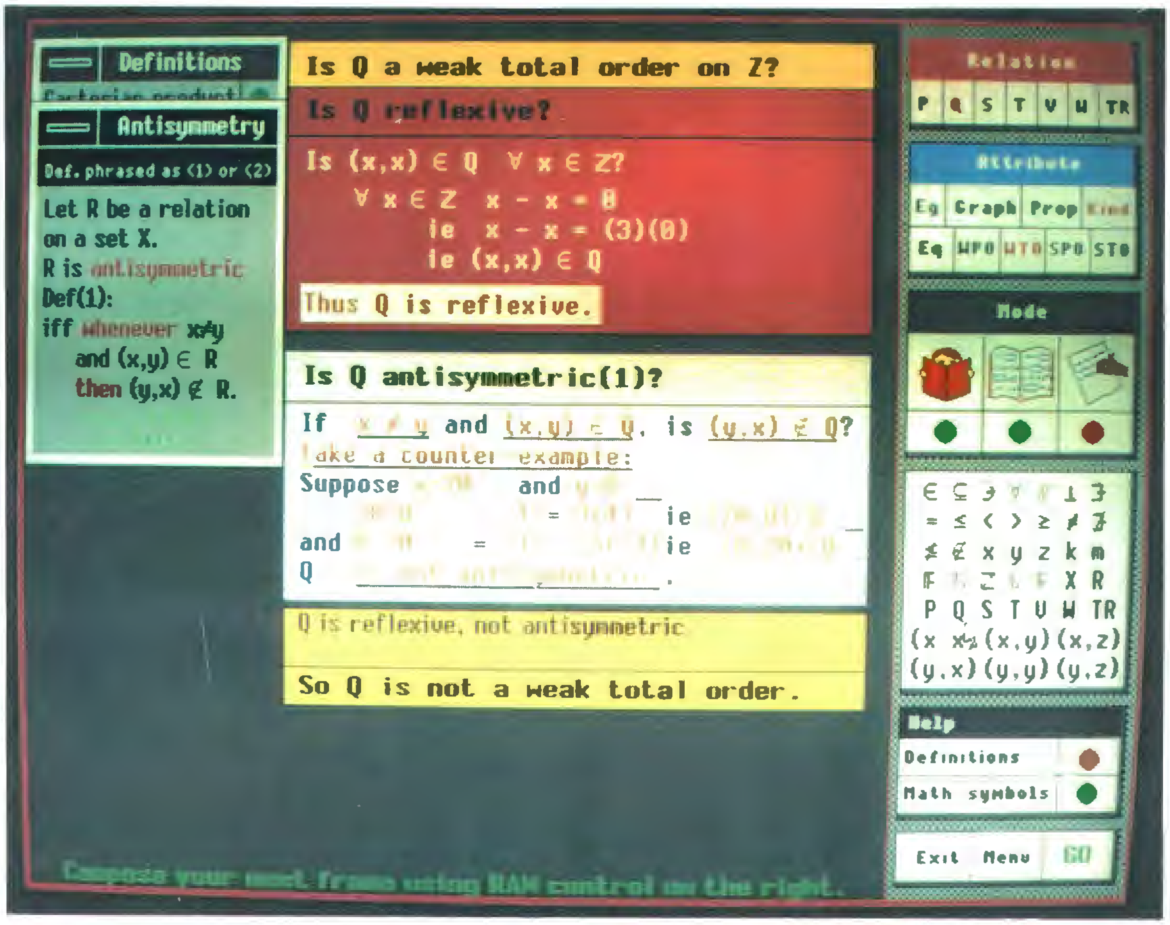
Press Enter.

Screen Display 9.7

FRAMES is equipped to test for a particular *Kind of relation* (point 7 in section 9.3.1) by analyzing a specific set of properties. If the relation satisfies each of them, then it is a relation of that particular kind.

Selection of the attribute *kind* produces a submenu for kind (see Screen Display 9.10). On selecting a particular kind, in this case, *WTO* (a weak total order analysis for **Q**), a question component appears, which asks the user to name the necessary properties. The user-response is judged:

- Certain spelling errors, e.g. "antisimmetric", are accommodated
- Wrong answers are replaced by correct answers, e.g. "transitive", in red to focus attention.



Def. phrased as (1) or (2)

Let R be a relation on a set X.
R is antisymmetric
Def(1):
iff whenever x≠y
  and (x,y) ∈ R
  then (y,x) ∉ R.

Is Q a weak total order on Z?

Is Q reflexive?

Is (x,x) ∈ Q ∀ x ∈ Z?
  ∀ x ∈ Z x - x = 0
    ie x - x = (3)(0)
    ie (x,x) ∈ Q

Thus Q is reflexive.

Is Q antisymmetric(1)?

If x ≠ y and (x,y) ∈ Q, is (y,x) ∉ Q?
Take a counter example:
Suppose [illegible] and [illegible]
[illegible] = [illegible] ie [illegible]
and [illegible] = [illegible] ie [illegible]
Q [illegible].

Q is reflexive, not antisymmetric

So Q is not a weak total order.

Compose your next frame using RAM control on the right.

Screen Display 9.8

This follows on from Screen Display 9.7. The system "knows" the set of specific properties for each *kind*, and automates presentation of the relevant property tests. Mode can be changed between tests by the user, as was done here, where the reflexivity test was done in Mode 1 and the antisymmetry test in Mode 3.

The first proof component overlays the question component shown in Screen Display 9.7, except for the top line which remains visible as a heading.

Should any property test turn out negative, it is clear that the relation is not of that kind, and the composite analysis is terminated immediately with an apt conclusion.

In doing the test for antisymmetry, the user accessed *Help* for *definitions* and clicked on the *antisymmetry* elaboration, which appeared as an overlay.

9.3.2 CONTENT OF FRAMES

Mathematical set notation (see section 2.4.2) is used to list the required subject-matter of FRAMES, i.e. the relations, attributes and special kinds of relations shown in Figure 9.3. Seven relations are introduced, but only two are comprehensively treated in the prototype. A mnemonic occurs in the name, **R A M** control, given to the control structure, reminding the user to select each exercise by choosing a **R**elation, an **A**ttribute and a **M**ode. The **R A M** control structure is illustrated in Screen Displays 9.9 and 9.10.

1. The **Relations** selected for exercises, each one a situational problem, comprise the set:

{P, Q, S, T, V, W, TR} where:

P is the set on$\mathbb{Z}$ of all **(x,y)** pairs such that $\mathbf{x \leq y}$
(the domain$\mathbb{Z}$ is the set of integers, the set { ... ,-2, -1, 0, 1, 2, ... }).

Q is the set on$\mathbb{Z}$ of all **(x,y)** pairs such that $\mathbf{x - y = 3k}$
i.e. the difference between x and y is a multiple of 3.

S is the set on$\mathbb{P}$ of all **(x,y)** pairs such that $\mathbf{x \mid y}$
i.e. x is a factor of y, and y is a multiple of x
(the domain$\mathbb{P}$ is the set of positive integers, the set {1, 2, 3, ... }).

T is the set on $\mathcal{P}$U of all **(A,B)** pairs such that $\mathbf{A \subseteq B}$
i.e. A is a subset of B
(the domain $\mathcal{P}$U is the universal set, the set of all sets).

V is the set on $\mathbb{Q}$ of all ($\frac{a}{b}$, $\frac{c}{d}$) pairs such that $\mathbf{ad - bc = 0}$
i.e. x and y are equivalent fractions
(the domain $\mathbb{Q}$ is the set of rational numbers, the set of numbers that can be written in the form $\frac{a}{b}$ where a,b $\in$ Z).

W is the set on$\mathbb{Z}$ x$\mathbb{Z}$, or on the set of two-letter words, of all **((a,b),(c,d))** pairs
such that either $\mathbf{a < c}$
or $\mathbf{a = c}$ and $\mathbf{b \leq d}$
i.e. numeric order of two-digit numbers, or alphabetic order of two-letter words.

TR (Tennis Rankings) is the set on TP, the set of tennis players, of all **(x,y)** pairs such that **x is ranked higher than y**.

2. The **Attributes** available for learners to "VIEW or DO" are:

{Examples, Graphic, Property, Kind} abbreviated to **{Eg, Graph, Prop, Kind}** where:

The **Example** attribute gives the learner the opportunity to synthesize example members of the current relation.

The **Graphic** attribute shows a visual representation of the relation:

The **Property** attribute presents proofs of:
{reflexivity, irreflexivity, symmetry, antisymmetry, transitivity, trichotomy}, abbreviated in the right hand control area to **{Ref, Irr, Sym, As1, As2, Tra, Tri}**, where As1 and As2 are tests for antisymmetry, according to its first and second definitions respectively.

The special **Kinds of Relations** occurring are:
{equivalence, weak-partial-order, weak-total-order, strict-partial-order, strict-total-order},
abbreviated to **{Eq, WPO, WTO, SPO, STO}** respectively.

(Screen Display 9.10 shows the submenu that presents these options)

A special kind of relation is one for which certain properties hold, i.e. a specific subset of the properties should be satisfied. For each **kind** the necessary properties are listed:

Kind of Relation	Necessary Properties
Eq	{Ref, Sym, Tra}
WPO	{Ref, Asm, Tra}
WTO	{Ref, Asm, Tra, Tri}
SPO	{Irr, Asm, Tra}
STO	{Irr, Asm, Tra, Tri}

(Asm could be As1 or As2)

The knowledge of these five definitions is inherent in the system's knowledge base (see section 9.3.6), so that it can check the student's response and select the relevant set of exercises for the composite analysis.

3. The **Modes** for doing property and kind exercises are **{1, 2, 3}** where:

> **Mode 1** presents the user with a *read-only* proof,
>
> **Mode 2** is *guided practice* presenting a structure in which the learner completes blanks,
>
> **Mode 3** intersperses linking-structure with blank lines on which the learner inputs a *D.I.Y.* (do-it-yourself) proof.

Figure 9.4 extends the general structure chart of Figure 6.2 to the particular design of FRAMES, showing the components that comprise its instructional content.

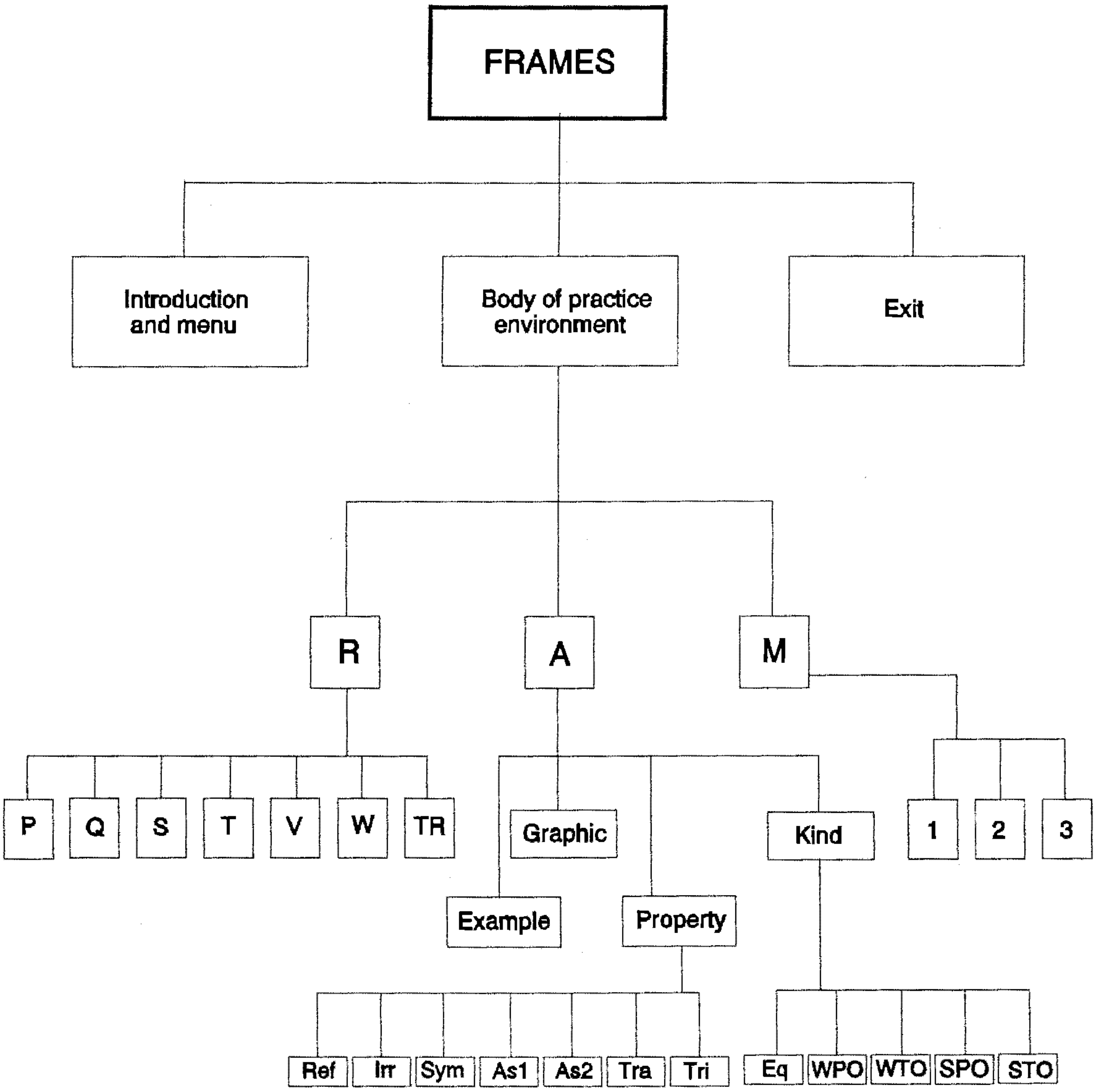
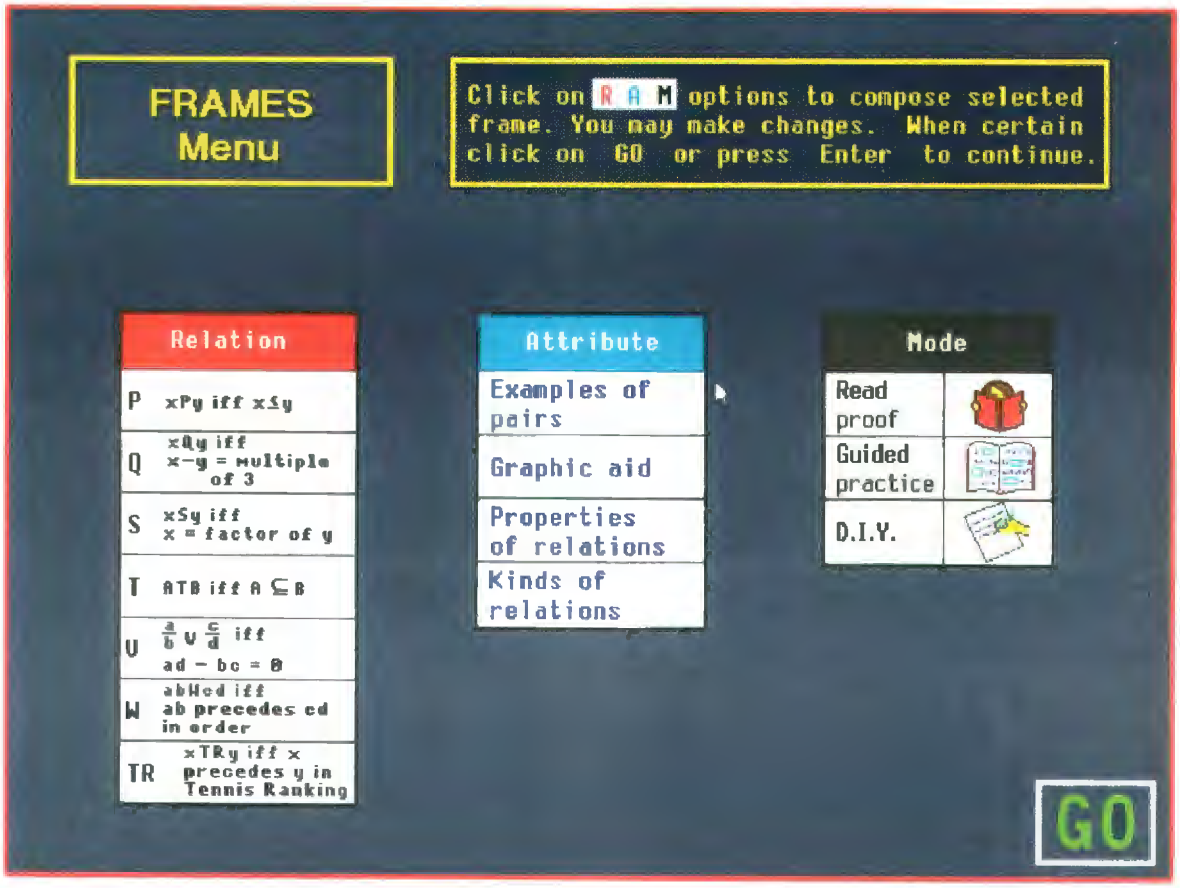


Figure 9.4 Structure Chart of FRAMES



FRAMES Menu

Click on R A M options to compose selected frame. You may make changes. When certain click on GO or press Enter to continue.

Relation
P xPy iff $x \le y$
Q xQy iff x−y = multiple of 3
S xSy iff x = factor of y
T ATB iff $A \subseteq B$
U $\frac{a}{b}$ U $\frac{c}{d}$ iff ad − bc = 0
W abWcd iff ab precedes cd in order
TR xTRy iff x precedes y in Tennis Ranking
Attribute
Examples of pairs
Graphic aid
Properties of relations
Kinds of relations
Mode
Read proof
Guided practice
D.I.Y.
GO

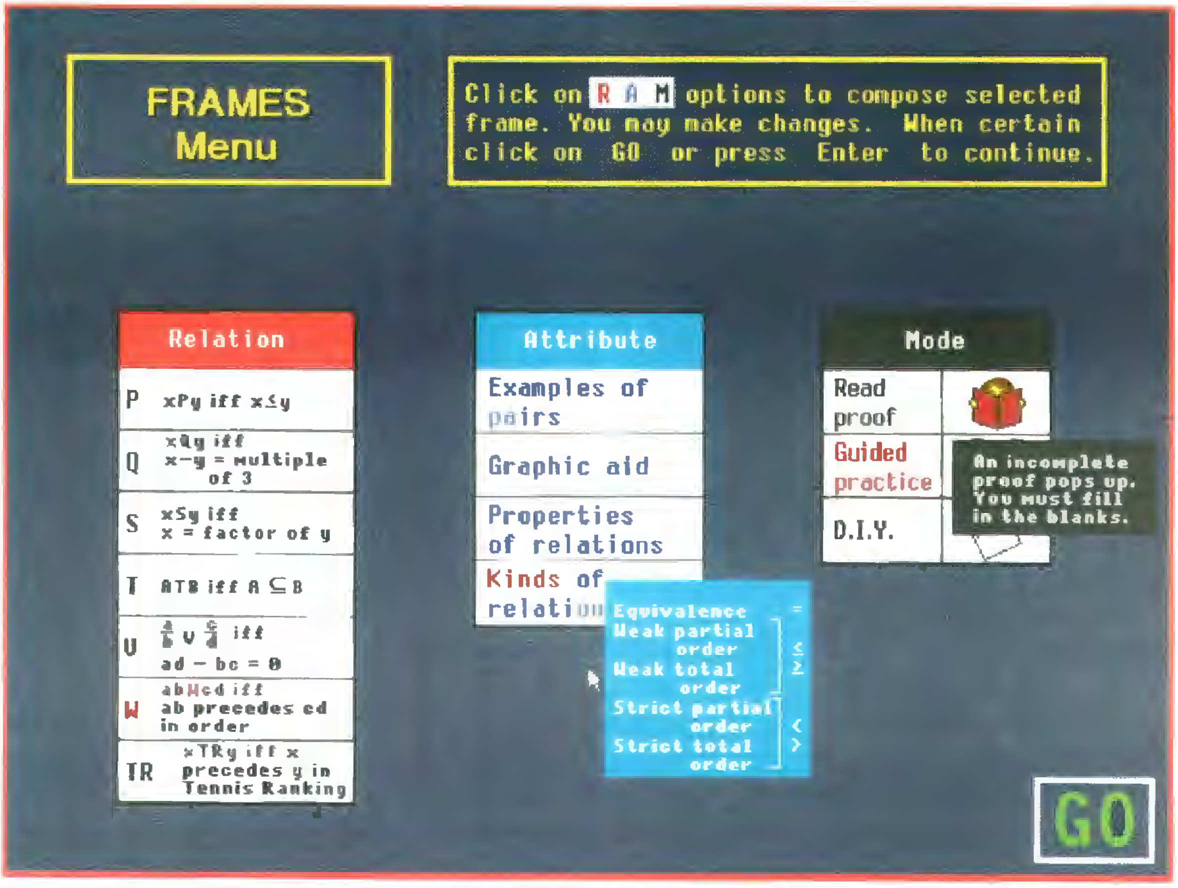
Screen Display 9.9

This is the FRAMES main menu screen, listing the relations, attributes and modes described in section 9.3.2.

The user composes a frame/component by clicking on:

- a **Relation**
- an **Attribute**
- a **Mode**.
- <**GO**>, to activate the selection.

Each use of a component is referred to as an instructional transaction. Once into the system, further selections are executed using the reduced menu in the right-hand control area and shown in the preceding Screen Displays.



Click on R A M options to compose selected frame. You may make changes. When certain click on GO or press Enter to continue.

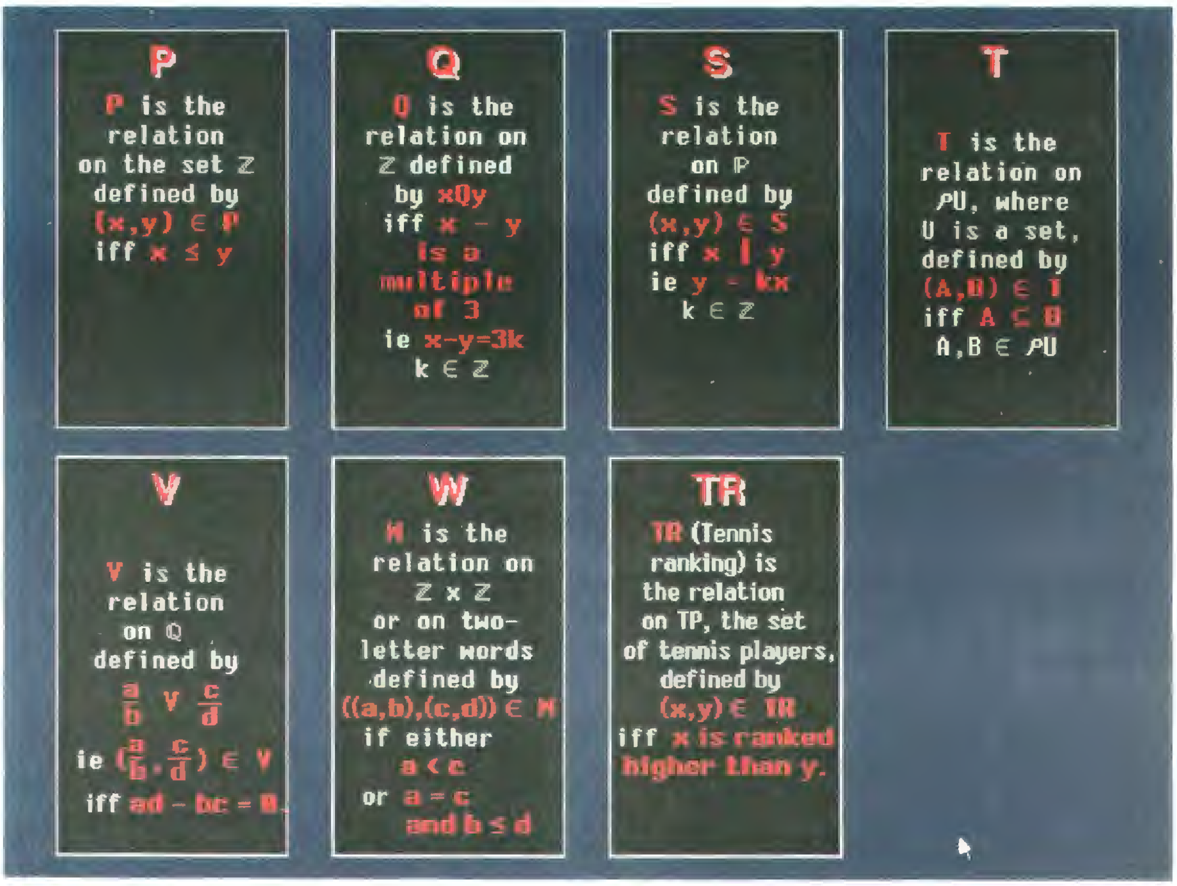
An incomplete proof pops up. You must fill in the blanks.

Screen Display 9.10

Here is the main menu after composition of the selected frame, but before activation of <GO>.

Clicking on **Attributes** produces an elaboration of the selection. Selection of *Properties of relations* and *Kinds of relations* produces lower-level submenus, showing the expansions set out on page 178 and portrayed in Figure 9.4. Clicking on a **Mode**, in this case Mode 2, sets out an elaboration of the strategy used by that mode.

Selected options are highlighted in red.



P

P is the
relation
on the set $\mathbb{Z}$
defined by
$(x,y) \in P$
iff $x \leq y$

Q

Q is the
relation on
$\mathbb{Z}$ defined
by xQy
iff $x - y$
is a
multiple
of 3
ie $x-y=3k$
$k \in \mathbb{Z}$

S

S is the
relation
on $\mathbb{P}$
defined by
$(x,y) \in S$
iff $x \mid y$
ie $y = kx$
$k \in \mathbb{Z}$

T

T is the
relation on
$\mathcal{P}U$, where
U is a set,
defined by
$(A,B) \in T$
iff $A \subseteq B$
$A,B \in \mathcal{P}U$

V

V is the
relation
on $\mathbb{Q}$
defined by
$\frac{a}{b}$ V $\frac{c}{d}$
ie $(\frac{a}{b}, \frac{c}{d}) \in V$
iff $ad - bc = 0$

W

W is the
relation on
$\mathbb{Z} \times \mathbb{Z}$
or on two-
letter words
defined by
$((a,b),(c,d)) \in W$
if either
$a < c$
or $a = c$
and $b \leq d$

TR

TR (Tennis
ranking) is
the relation
on TP, the set
of tennis players,
defined by
$(x,y) \in TR$
iff x is ranked
higher than y.

Screen Display 9.11

Whenever the user's selection calls in a component of a new/changed relation, a "definition blackboard", describing the selected relation appears at the top left. The relation definitions are similar to those on page 177.

This Screen Display is not an official screen in the FRAMES practice environment. It was created to collate all the "blackboards" for interest's sake, and has been left in place as part of the <Exit> path.

9.3.3 THE COMPONENT-BASED STRUCTURE OF FRAMES

Frames is designed in the spirit of Component Display Theory (sections 5.4.5 & 6.7). In Chapter Six (sections 6.8.7, 6.9.1 & 6.9.2) the researcher's decision to design FRAMES as a component-based system was outlined. The three modes, *read, guided-practice, and DIY*, are similar to Merrill's *remember, use and find* performances respectively. Other performances required of students are the contextualization of definitions and the synthesis of examples. The type of content to be learned includes Merrill's *facts* and *concepts* (for example the definitions), *procedures* (the proof- and disproof-techniques used in tests for properties), and *principles* (the rules used in individual tests for properties, and in the classification of the overall relation as a particular kind). The final envisaged FRAMES system will incorporate an extensive library of components:

7 relation definitions	=	7 components
7 visual aids	=	7 components
7 example-synthesis exercises	=	7 components
7 relations x 6 properties x 3 exercise-modes	=	126 components
Total	=	**147 components**

There are actually 21 more, making a total of **168**, since the antisymmetry property has two alternate definitions and the test can be done in two different ways, termed antisymmetry (1) and (2). Since antisymmetry(2) is also to be done for seven relations in three different modes, it adds 21 extra components.

The prototype includes all seven relation definitions, seven visual aids, and six example-synthesis components. The property exercise components have been designed for three of the seven relations and programmed for two. There are also ancillary components, such as the nine definitions of properties and the twelve mathematical terms available as <Help> facilities. In total the prototype FRAMES comprises 69 components.

By supplying a test for each property for every relation, a rich environment of examples and non-examples is provided. It is unlikely that a single student would, at one sitting, tackle all the 126 property components in the final package, but whatever the preferred learning style or stage of learning, material would be available to meet his current needs.

User-control permits the learner to select his own components, both with respect to content and instructional strategy. Figure 9.5 categorizes the components of FRAMES according to Merrill's performance-content matrix.

LEVEL OF PERFORMANCE

	fact	concept	procedure	principle
find			Property - mode 3	Example synthesis
use	Definition blackboard	Help - maths, Graphic aid	Property - mode 2	Help - definitions, Example synthesis, Kind of relation
remember	Definition blackboard	Graphic aid	Property - mode 1	Kind of relation

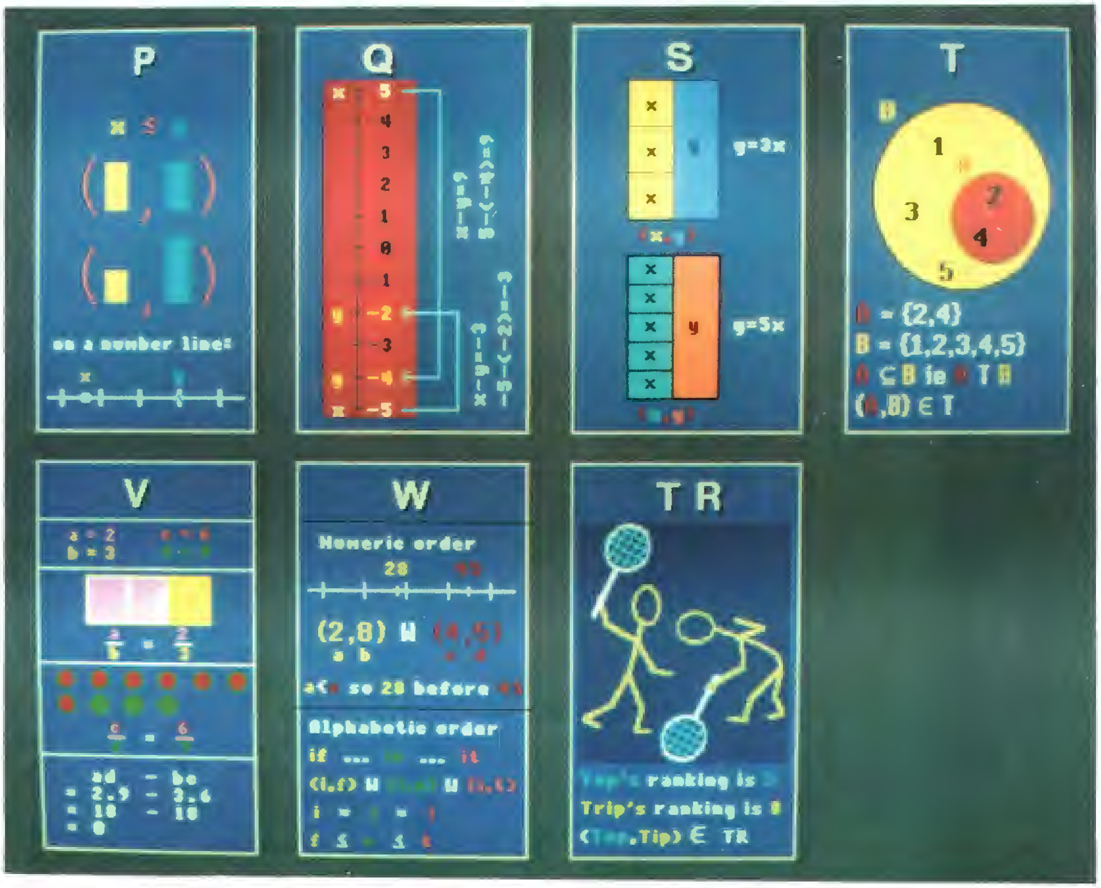
TYPE OF CONTENT

Figure 9.5 A Performance-Content Matrix for the Components of FRAMES

Use of a component by a user is called an *instructional transaction.*

The goals of the practice environment are general domain familiarity, exposure to worked examples, and experience in independent exercises. To achieve these, the environment should incorporate rich variety in instructional transactions, and the broad coverage of the matrix in Figure 9.5 shows that this is the case.

Many of these components have been illustrated in previous Screen Displays, for example, "definition blackboards" (Screen Display 9.11), example synthesis (Screen Displays 9.3 and 9.5), Modes 1, 2 & 3 (Screen Displays 9.1, 9.2, 9.4, 9.6 & 9.8), and kind of relation (Screen Displays 9.7 & 9.8). Screen Displays 9.12 and 9.13 focus on graphic aids and <Help> elaborations.

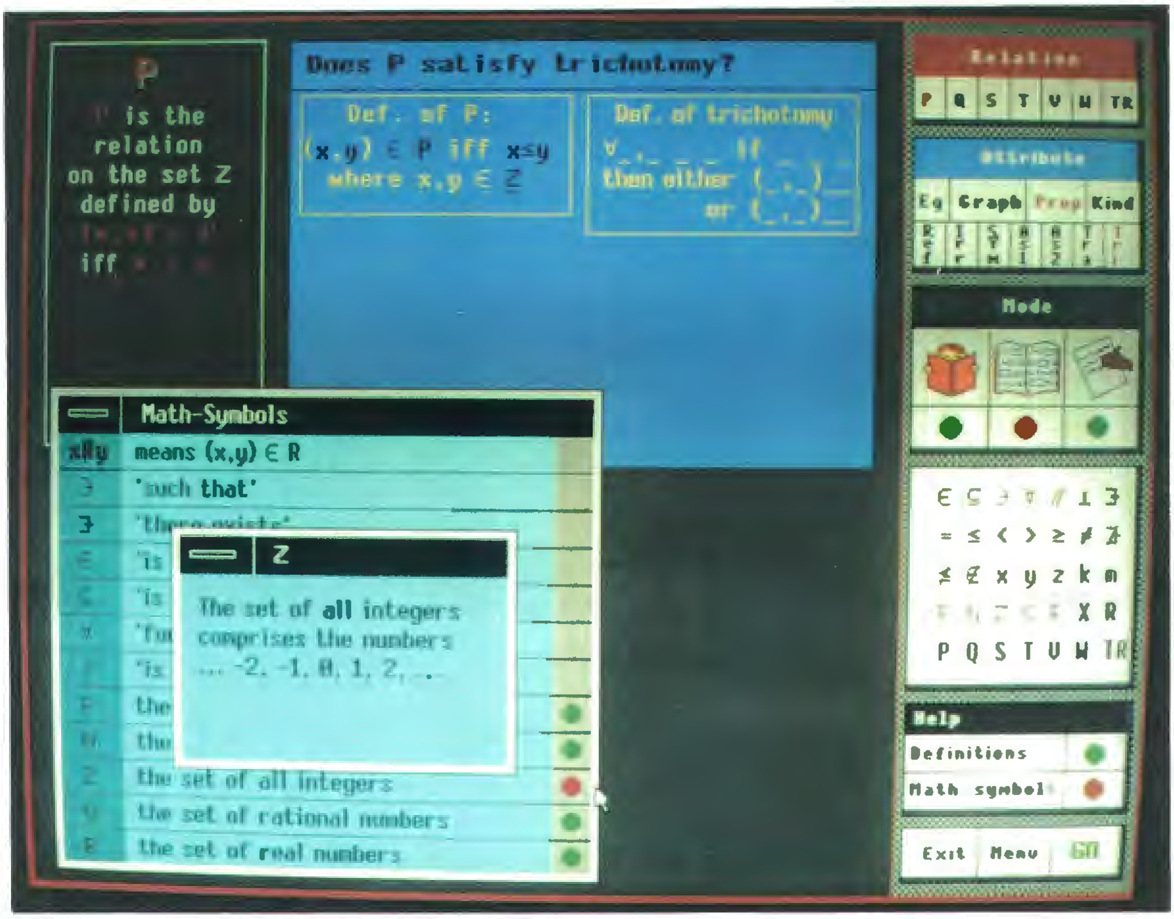


Screen Display 9.12

Each relation is illustrated by a graphic aid. Visualization plays an important part in facilitating perception, application and retention of a concept or principle (see Figure 9.5).

As with the definitions, a screen was compiled comprising all seven graphic aids. For interest's sake, it has also been left in place on the <Exit> path.

Notice the colour coding in each graphic aid, relating the text or mathematical representation to the diagram.



P

P is the relation on the set Z defined by (x,y) ∈ P iff x ≤ y

Does P satisfy trichotomy?

Def. of P:
(x,y) ∈ P iff x≤y
where x,y ∈ Z

Def. of trichotomy
∀_,_ _ _ if _ _
then either (_,_)_
or (_,_)_

Math-Symbols

xRy	means (x,y) ∈ R
∋	'such that'
∃	'there exists'
∈	'is
⊆	'is
∀	'for
⊥	'is
P	the
N	the
Z	the set of all integers
Q	the set of rational numbers
R	the set of real numbers

Z

The set of all integers comprises the numbers
..., -2, -1, 0, 1, 2, ...

Screen Display 9.13

In Figure 9.5 *Help* components are positioned in the cells relating to the *use* or application, level of performance. In FRAMES they provide situational support for problem solving.

Screen Display 9.2 shows the two kinds of Help available, namely, *Definitions* and *Math-symbols*. Screen Display 9.8 shows, as elaboration, a definition of the antisymmetry property, and the screen above demonstrates an expansion of a mathematical notation, namely, the set ℤ of integers. Should the user require a different elaboration, he clicks on the close slot at top left of ℤ to clear the current elaboration then accesses another. Clicking on the close slot at top left of a *Help* menu clears the entire overlay menu.

Also of interest in this screen is the portrayal of the first stage in a Mode 2 property test, presenting the two definition blocks which must subsequently be related to one another.

9.3.4 SOFTWARE ENGINEERING METHODOLOGY APPLIED TO FRAMES

Software engineering entails the development methodology of a system. Analysis and design in conventional SE includes extensive attention to data- and process flows. In the CAI milieu, there is little data to flow, and in the FRAMES environment, most processes (instructional transactions) are complete in themselves. The major processing activities are the calling of components (usually by the learner; occasionally by another component) and the assessment of learner-responses. The selected life-cycle model, namely prototyping, was discussed in section 9.1. The SE methodology followed is the object-oriented design paradigm.

The team approach

The team approach advocated in section 3.3, although desirable in courseware engineering, was not used because of the nature of this project as part of a post-graduate study course. A programmer coded the prototype, but otherwise it is the unaided work of the researcher, and appropriate for submission as partial fulfilment of the requirements for the MSc degree.

The object-oriented design paradigm

The object-oriented paradigm, viewing a domain as being comprised of entities and inter-relationships, was discussed in section 2.2.2 and identified as being appropriate for CAI. The OOP projects beyond the pure software components and incorporates features from the environment. Objects play a unifying role as top-down analysis and bottom-up program development occur simultaneously, with the activities of analysis, design and implementation merging into one another. It is highly compatible with prototyping. As was stated in section 2.5.2 and in the conclusion to Chapter Two, FRAMES, with its innovative structure and content, lends itself to an object-oriented approach.

Software engineering representations and tools

SE representations and tools facilitate the analysis process. Various analysis and design tools introduced in section 2.4 can be used to represent the structure and objects of FRAMES. Since the domain objects in ISD are quite different from those in conventional Information Systems (ISs), the application of SE tools tends to be somewhat unconventional.

Booch Diagrams (see section 2.4.1) depict the problem-domain objects, their operations, and messages between them. Objects are shown in vertical rectangles, carrying an inset oval to identify the object and small horizontal rectangles to identify its operations. Service objects appear in vertical rectangles without inset ovals. Lines between objects signify message connections. Figure 9.6 represents the objects, operations and messages of FRAMES.

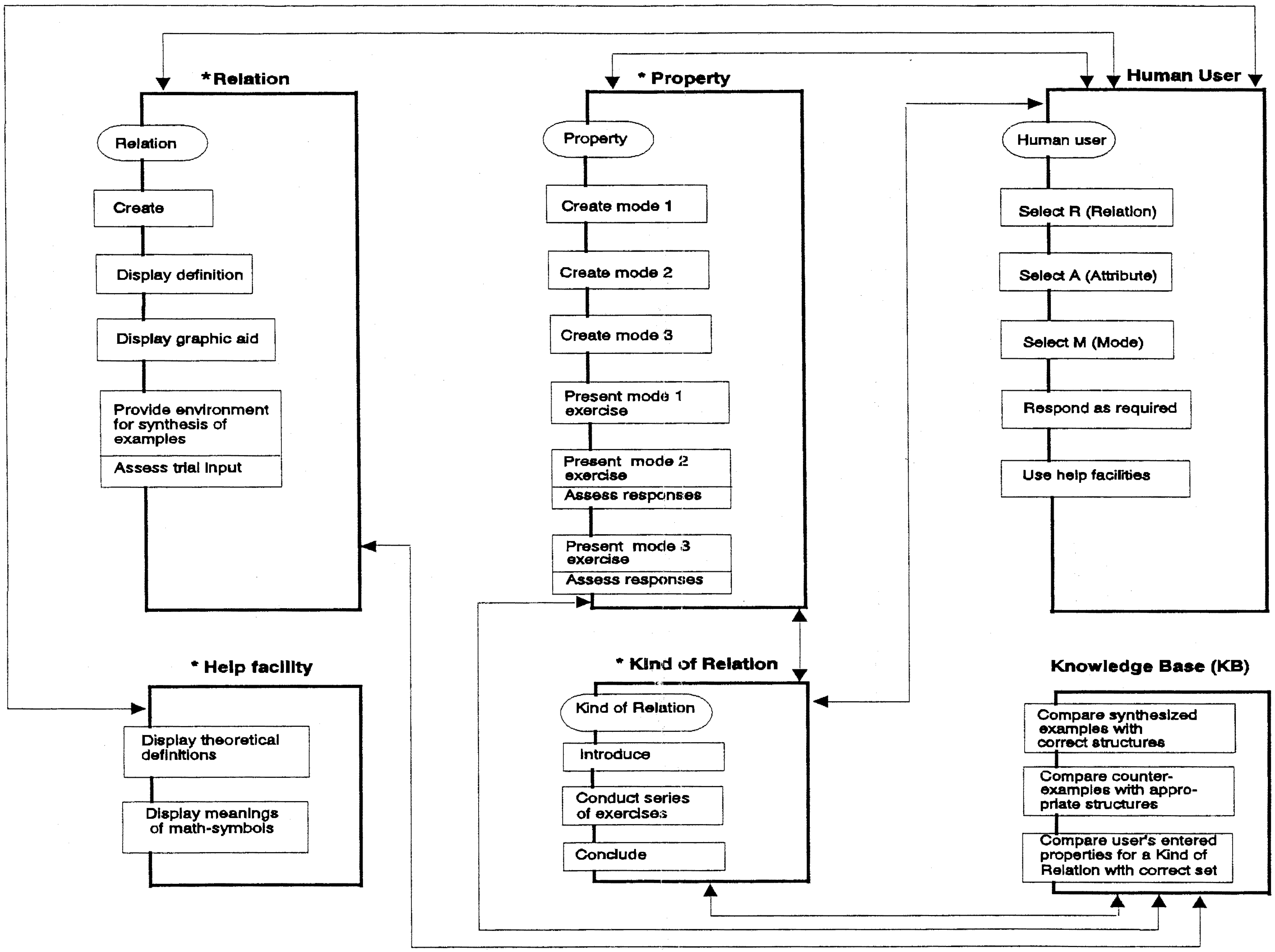


Figure 9.6 The Objects, Operations and Messages within FRAMES, illustrated by a Booch Diagram

The problem domain objects are:

1. *Relation:*
   The relations available to the student for problem-solving exercises were defined in section 9.3.2. The relevant operations include the necessary step/s by the designer, as well as the its use by the learner.

2. *Property:*
   The properties for which the relation must be tested are listed in section 9.3.2. Assessment of the learner's response and determination of appropriate feedback accounts for a major share of the effort in producing FRAMES.

3. *Human User:*
   The learner participates actively, calling components by R A M control. The calls are shown as messages. In this respect the system differs notably from conventional data processing where modules call each other.

4. *Kind of relation:*
   These are outlined in section 9.3.2. The object *kind* communicates with the object *property*, calling in the necessary property tests.

The service objects are:

1. *Help facility:*
   This comprises a library of theoretical definitions and mathematical elaborations to be displayed on demand. *Help* is illustrated in Screen Displays 9.2, 9.8 and 9.13.

2. *Knowledge base (KB):*
   The required knowledge, described in section 9.3.6, is knowledge of the composition of each *kind* and of the internal structure (relationship between first co-ordinate and second co-ordinate) of each relation. Messages are passed between the KB and other objects in the response-judging task. The manipulations involved in checking mathematical interrelationships are the closest FRAMES comes to conventional data processing.

9.3.5 INSTRUCTIONAL DESIGN AND FRAMES

Classical ID emphasises assessment of needs and goals, explicit objectives, and a sequential, step-wise instructional strategy geared specifically towards the achievement of those objectives. Branching is used to facilitate differentiation between individual learners. Testing, both pre-testing of prior knowledge, and assessment-testing after the instruction, is of prime importance.

FRAMES is more in line with the current shift in emphasis towards flexible educational software. It is essentially intended for the presentation of instructional components that offer students the options of perusal, application of principles and skill-practice. However the classic ID principles for CAI apply to its design. The ten design features outlined in section 6.5 are listed one by one, followed by a brief description of their application in FRAMES.

1. **Instructional approach**

This was discussed and illustrated in section 9.2.3 where it was stated that the instructional philosophy of FRAMES was to be primarily cognitive, but that aspects of behaviourism play important roles. Some further **practical design implications** of *cognitive theory*, as applied in FRAMES, are:

♦ The load on working-memory is variable, in that the learner can reference relevant definitions and rules via the <Help> facility shown in several Screen Displays.

♦ Cognitive theory aims for learner-comprehension of a concept by integration with existing knowledge structures or schemata. This is often achieved when a learner can backward-chain from the goal. When asked to demonstrate or prove something, the learner should be able to determine the "second-last step", i.e. a necessary and sufficient condition, which, if true, would make the overall goal true too. Mode 2 encourages this kind of reasoning:

(a) The proof forces retrieval of prior knowledge by commencing with a fill-in definition of the appropriate property. This definition is the general case for a relation *R* on a set *X*, so as to encourage the learner to move from the general to the particular, as he applies the abstract definition of the property to the concrete definition of the relation (see Screen Displays 9.13 and 9.14).

(b) The next step is the *R.T.P.* (required-to-prove) step, i.e. the learner's aim, the subgoal/s to be proved in order to achieve the main proof. A completed **R.T.P.** step is shown in Screen Display 9.14.

(c) This is followed by the actual *Proof*, which is done step by step. In each line the learner completes several blanks and, in order to keep the procedure on track, judgement occurs at the end of each major step. The student is given two chances for each step; should the student-response still be incorrect after the second try, the system supplies the correct values, terms, etc. In such cases the underscore place marker remains in place (see Screen Display 9.14) to denote steps where misconceptions or errors occurred, and to distinguish them from those which the learner got right independently.

A further practical illustration of implementation of *behaviourism* is:

♦ Cuing and fading:

(a) The line lengths for fill-in responses in Mode 2 serve as cues to the type of response required; this is shown in Screen Display 9.14.

(b) Mode 3 provides long lines for each step, deliberate fading of the cue, which can be seen in Screen Display 9.6.

2. Menus and directions

FRAMES commences with a series of introductory screens:

1. *Title screen*

2. *A VIEW of what YOU can DO with FRAMES* (Screen Display A.1 in Appendix A): Outlines user-control and briefly introduces the **R A M** system. It serves as an advance organiser for the subsequent screens.

3. *What YOU can DO with FRAMES* (Screen Display 9.14):
   - Expands on previous screen, defining the seven relations and elaborating on each attribute and mode, and
   - Offers a branching option to:

4. *Demo screens* (Screen Display 9.18 and Screen Display 9.19): Demonstrate the three modes of doing exercises (eliciting user-interaction in Modes 2 and 3), encourage practice with <Help>, and require the entering of mathematics characters by mouse-clicking on the symbol pad. The demo returns control to the calling screen.

5. *Menu* (Screen Displays 9.9 and 9.10):
   User clicks on R A M, then on <GO> to call a component. Should the selected attribute be *property* or *kind*, a lower level menu appears.

The two kinds of *Help* components, definitions and mathematical aid, which are called in from the right hand control area, appear as overlay menus on the left and are shown in Screen Display 9.2. Using these menus, the user can click on the definition or mathematical symbol he would like elaborated. Elaborations of definitions and mathematical notations are shown in Screen Displays 9.8 and 9.13 respectively.

3. Layout and text

The screen is subdivided, with the functional area on the right dedicated to user-control. Having called the first component via the main menu, the user can access further components using R A M control (Relation, Attribute, Mode) in the mini-menu on the right, followed by a click on <GO>. The upper-left region is devoted to the definition "blackboard" of the most recently called relation, and the space below it is reserved for the corresponding graphic aid component, when called. Central screen is the working area where the main action happens. All other interaction and exercise components appear here and a step-scroll mechanism moves each component up as the next is called. Depending on their depth, up to three components can be simultaneously on-screen. Elaborations called by <Help> overlay the components on the left, and can be cleared by clicking on the close slot at their top left (see Screen Display 9.13).

4. Colour

Consistent colour standards are used in the components:

♦ *Definitions* - a "blackboard" appearance, with red and white "chalk".
♦ *Ref and Irr properties* - black and white fonts on red background.
♦ *Sym and Asm properties* - blue and red fonts on white background.
♦ *Tra property* - black and white fonts on dark blue background.
♦ *Tri property* - black and yellow fonts on sky-blue background.
♦ *Kind of relation* - black font on yellow background for introduction and conclusion.
♦ *Example synthesis* - black and white fonts on green background.

The question heading each property test is in bold black font. Reverse video is used at the end of each Mode 1 proof to emphasize the conclusion (Screen Display 9.16).

Colour coding is used in the graphic aids to relate each diagram to the associated formula or text.

5. Graphics and animation

Visualizing the relationship between the first and second co-ordinates of a relation helps the learner gain a feel for the domain. The graphic aid components depicted in Screen Displays 9.12 and others portray diagrammatic representations of the relation's definition.

Graphic icons are used to indicate the types of interaction in Modes 1, 2 and 3.

Student participation in Mode 2 and 3 proofs entails the entry of mathematical symbols and characters not available on the keyboard. Both these modes provide a symbol pad, technically termed a *soft keyboard*, in the right-hand functional area. This is shown in Screen Displays 9.1 and 9.2 respectively. Mode 3 requires an extended pad including objects such as (x,y) pairs. These characters are imported into the proof by mouse-clicking.

6. Interactivity

The first aspect of interactivity is control. The learner holds total control of his practice experience, and may select from the available components. He may choose to re-visit the same problem, or an aspect of that problem (e.g. a specific attribute) in various modes to enhance cognitive processing. Conversely, the learner may work on different problems, tackling, for example, the same property of different relations to obtain multiple perspectives on that property.

The second aspect is active participation. FRAMES permits interactivity at a level far surpassing yes/no, multiple choice, and simple fill-in-the-blank answers. Mode 2 requires the user to fill in several blanks in each step of the proof, and in Mode 3 the user supplies the entire proof, one step at a time.

7. Individualization

FRAMES incorporates individualization, but not personalization (i.e. addressing the user by first name). The user-control system allows the learner to plan his own learning and practice experience, in terms of content, sequence, quantity and instructional style.

8. Question/Answer/Feedback process

There are very few questions as such in FRAMES. Instead, the learner responds by participating in example synthesis and in mathematical proofs. As described, he completes the proofs in Mode 2 by filling in the blanks, and in mode 3 by supplying entire steps. Each step taken by the learner is monitored, and each student-response followed by a system-response in the form of appropriate feedback. Forethought has been given to typical errors, and remedial feedback prepared, as shown in various Screen Displays.

In a multiple fill-in-the-blank situation, the backspace key permits a user to backtrack over fixed text to rewrite in previous blanks. Once an answer has been accepted as correct, however, it is protected and a user cannot backspace over it. A response judged incorrect is treated by appropriate feedback, and the wrong answer replaced by the correct version, so that the user can continue.

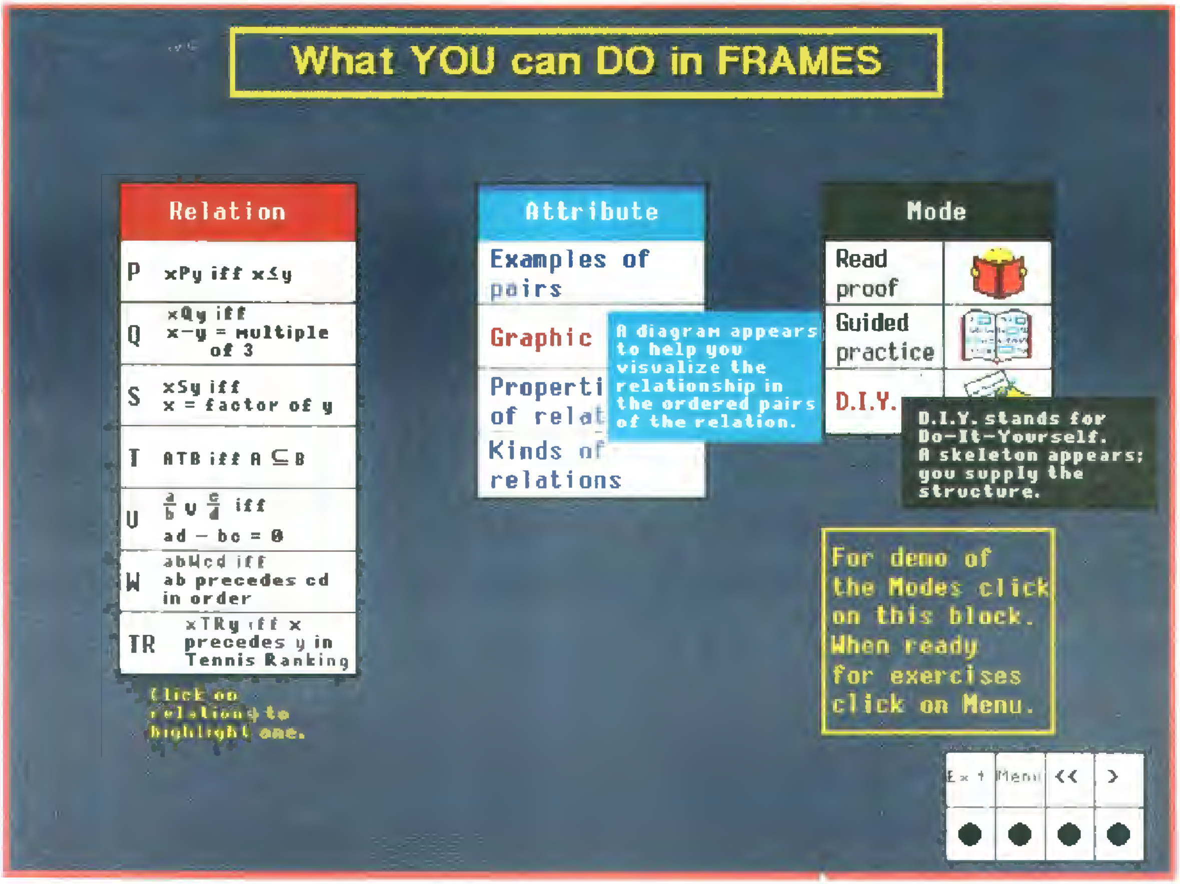
Although a mature learner appreciates total freedom in determining his proof strategy, there is no alternative in the design of FRAMES other than to work according to a relatively fixed solution path, because the assessment of open-ended answers is a complex task, and the proof strategies entailed are fairly rigid, permitting only minor deviations in terms of sequence and structure. Nevertheless, response judging is one of the major processing events in FRAMES, as the learner's input is assessed. Provision is made for correct alternative answers and alternative spellings.

9. Motivation

FRAMES engenders intrinsic motivation. The learner should obtain satisfaction from organizing his own learning experience and gain increased self-confidence as his performance improves. There are no "bells and whistles", such as themes or rewards.

10. Complementary and supplementary media

FRAMES complements the text book and the study guide for COS101-S. To facilitate use, it is accompanied by a *how-to-use* card.



Relation	
P	xPy iff $x \leq y$
Q	xQy iff $x-y$ = multiple of 3
S	xSy iff x = factor of y
T	ATB iff $A \subseteq B$
U	$\frac{a}{b}\ U\ \frac{c}{d}$ iff $ad - bc = 0$
W	$abWcd$ iff ab precedes cd in order
TR	$xTRy$ iff x precedes y in Tennis Ranking

Click on relations to highlight one.

A diagram appears to help you visualize the relationship in the ordered pairs of the relation.

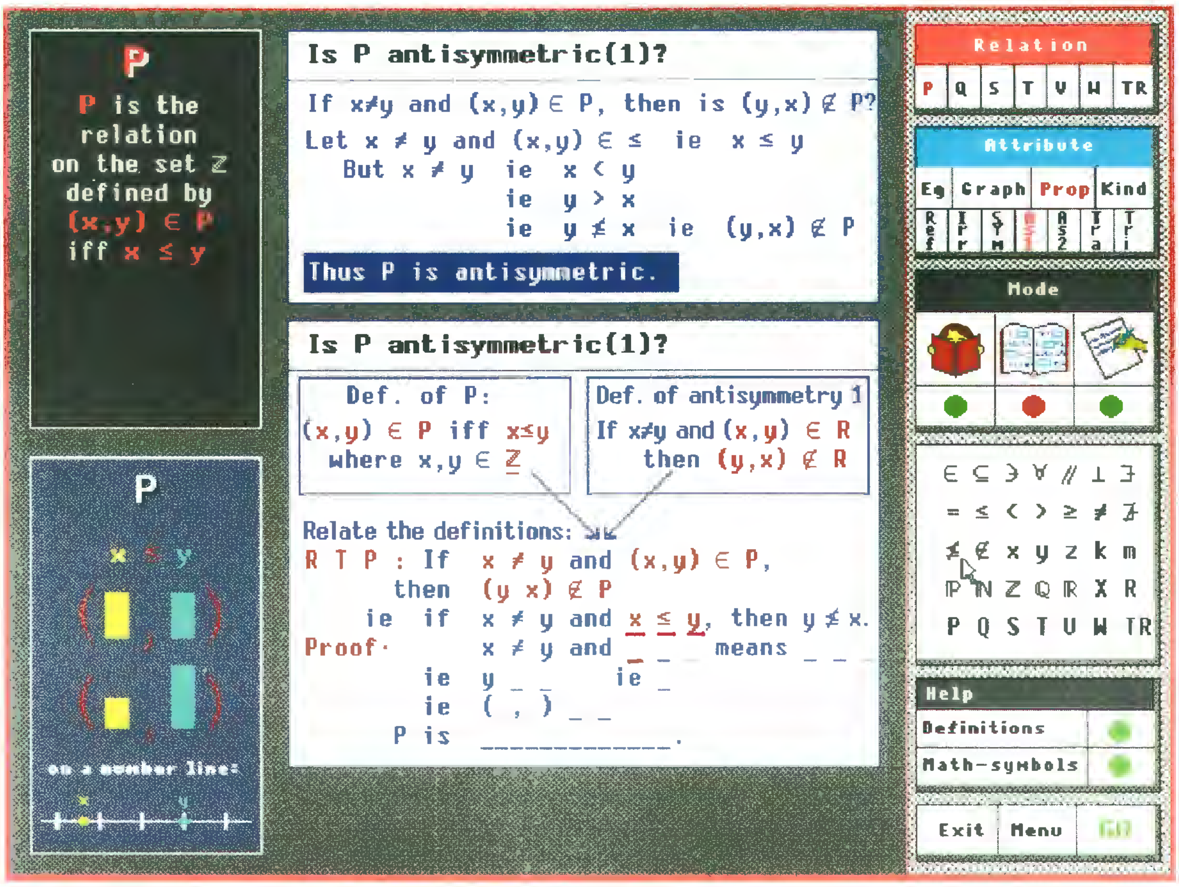
D.I.Y. stands for Do-It-Yourself. A skeleton appears; you supply the structure.

For demo of the Modes click on this block. When ready for exercises click on Menu.

Screen Display 9.14

On starting up the FRAMES practice environment, the user is presented with the introductory screens described in point 2 of section 9.3.5. The third screen, shown above, appears before the main menu, and sets the scene for it due to its analogous appearance and elaboration facilities. Its other main purpose is to offer access to a *demo* of the three instructional modes. Screen Displays 9.18 & 9.19 describe the demo. After using the demo, the user returns to this screen.

Note the elaboration of a graphic aid and the explanation of D.I.Y.



P

P is the relation on the set ℤ defined by (x,y) ∈ P iff x ≤ y

Is P antisymmetric(1)?

If x≠y and (x,y) ∈ P, then is (y,x) ∉ P?

```
Let x ≠ y and (x,y) ∈ ≤   ie   x ≤ y
   But x ≠ y   ie   x < y
               ie   y > x
               ie   y ≰ x   ie   (y,x) ∉ P
```

Thus P is antisymmetric.

Is P antisymmetric(1)?

Def. of P:
(x,y) ∈ P iff x≤y
where x,y ∈ ℤ

Def. of antisymmetry 1
If x≠y and (x,y) ∈ R
then (y,x) ∉ R

Relate the definitions:

```
R T P : If   x ≠ y and (x,y) ∈ P,
        then (y x) ∉ P
     ie  if  x ≠ y and x ≤ y, then y ≰ x.
Proof·       x ≠ y and _ _   means _ _
         ie  y _ _      ie _
         ie  ( , ) _ _
        P is ______________.
```

Screen Display 9.15

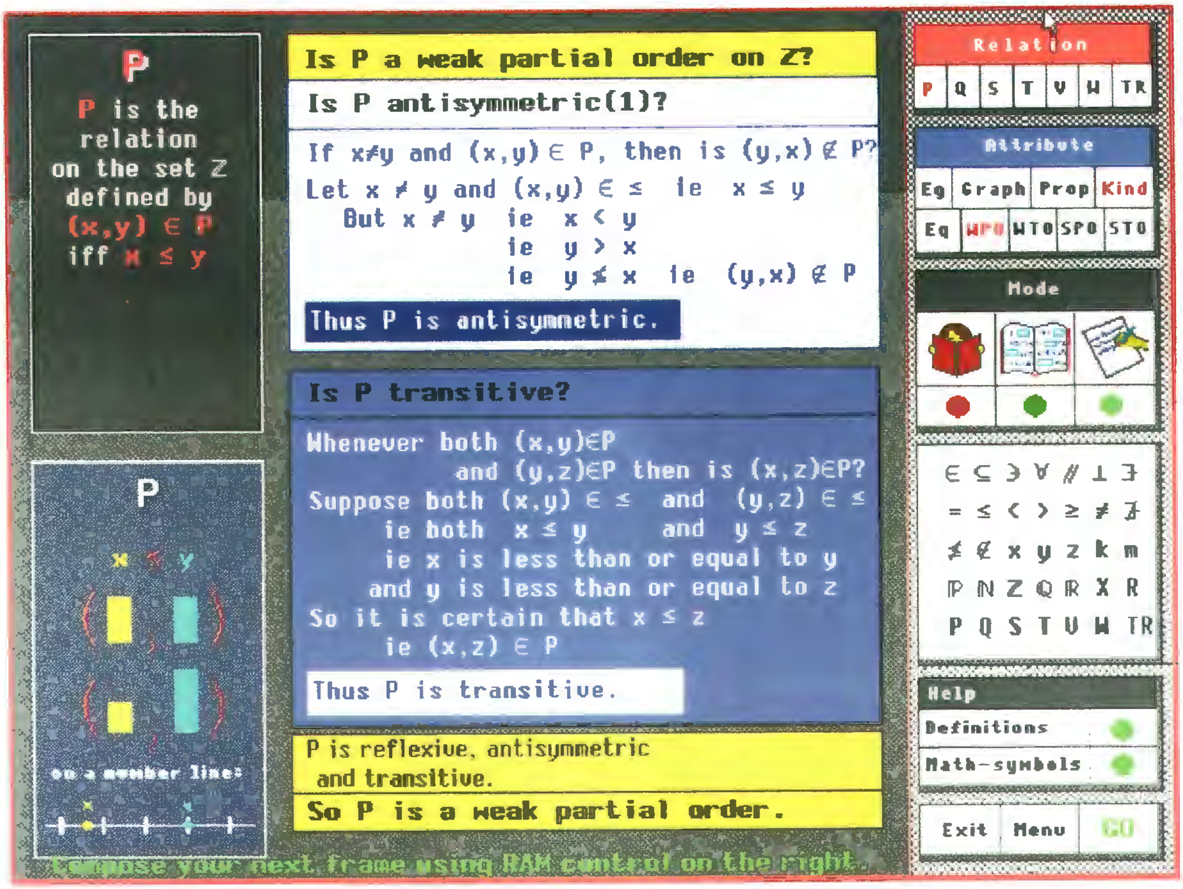
Mode 1 *perusal* is followed by Mode 2 *practice* of the same problem.

The first stage of a Mode 2 proof presents two definitions to be related to each other as described in Screen Display 9.13. The second stage, the **R.T.P.** (required to prove), entails supplying the content for an infrastructure. This forces the student to consider his overall aim, so that he can work in a goal-directed manner. The proof is shown, completed up to the end of the **R.T.P.** stage.

The actual proof remains to be done. Prior context-setting and goal-definition put the learner in a position to work in an efficient and focused fashion. Note that the number of spaces cue the user as to the exact length of the answer. After every step the student has a second chance, if his first answer is wrong. Should the second answer also be wrong, the correct response is supplied, so that he can continue, and the underscore place-marker remains in place to denote that an error occurred. See $\underline{x} \underline{\leq} \underline{y}$ in the third line of the **R.T.P**.

Two further points about proof-completion:

- The red place-marker is a cursor.
- Use of the symbol pad (soft keyboard) for entry of mathematical characters has been mentioned. This screen shows the mouse-arrow in position at ≰, the most recently clicked-in character (second-last character in line above **Proof**).



```
P
P is the
relation
on the set Z
defined by
(x,y) ∈ P
iff x ≤ y
```

```
Is P a weak partial order on Z?
Is P antisymmetric(1)?
If x≠y and (x,y) ∈ P, then is (y,x) ∉ P?
Let x ≠ y and (x,y) ∈ ≤  ie  x ≤ y
   But x ≠ y  ie  x < y
                  ie  y > x
                  ie  y ≰ x  ie  (y,x) ∉ P
Thus P is antisymmetric.

Is P transitive?
Whenever both (x,y)∈P
            and (y,z)∈P then is (x,z)∈P?
Suppose both (x,y) ∈ ≤  and  (y,z) ∈ ≤
      ie both  x ≤ y        and  y ≤ z
      ie x is less than or equal to y
    and y is less than or equal to z
So it is certain that x ≤ z
      ie (x,z) ∈ P
Thus P is transitive.

P is reflexive, antisymmetric
 and transitive.
So P is a weak partial order.
```

Compose your next frame using RAM control on the right.

Screen Display 9.16

Colour standards are described in point 4 of section 9.3.5. All the components have appeared in previous Screen Displays, except the *transitivity* property shown above. This screen also displays Mode 1 conclusions in reverse video.

A further aspect of note is the treatment of a *Kind* analysis. Screen Displays 9.7 and 9.8 showed a *WTO* analysis of relation **Q**, which was found not to be of that kind, and terminated immediately a test turned out negative. The *WPO* analysis of **P** in this Screen Display shows that **P** meets all the requirements and is a WPO. The analysis involved tests for reflexivity, antisymmetry and transitivity. The reflexivity test has been moved off-screen, but the others are shown, along with the introductory question and conclusion for a *Kind*. As has been pointed out, conclusions are not pre-specified or hard-coded, but determined by FRAMES during its composite analysis of multi-properties when testing for a particular kind.

FRAMES has a step-scroll mechanism, which builds up the central screen-display as new instructional transactions are initiated by the student. It moves components off-screen when full. Centre screen can generally accommodate two components, and sometimes three (see Screen Display 9.6). Selection of the *Kind* option clears the screen and starts at the top. Should a user wish, at any time, to start with a clear screen, he can do so by using the main menu, which can be clicked-in from bottom right.

9.3.6 THE FRAMES KNOWLEDGE BASE

FRAMES needs both macro-knowledge and micro-knowledge.

- The macro-knowledge relates to the characteristics of the five special *kinds of relation*, knowing which set of properties must test positive for each. It judges the learner's version of these properties, and then guides him step by step through the composite proof.

- The micro-knowledge relates to the mathematical relationship within each domain element, i.e. knowledge of the internal structure of the ordered pairs in each of the seven relations. This knowledge is used when the system judges the learner's *examples*, entered in the example-synthesis exercise, and also when it judges his *counter-examples* in a proof. It knows the correct relationship between the first and second co-ordinates of the ordered pairs, and tests the user's entries against this structure.

Screen Displays 9.7, 9.8 and 9.16 show the system's knowledge of the characteristics of the WTO *kind*. The introductory question and conclusion are not hard-coded; the text, reasoning and conclusion of each composite analysis for a kind are **not pre-specified**, but are inserted at use-time into a parameterized structure, using underlying knowledge of the requirements for that kind. This would facilitate inclusion of new relations. The intelligent feedback to incorrect calculations in Screen Displays 9.3 and 9.5 demonstrates knowledge of the mathematical inter-relationships inherent in each relation.

9.3.7 SUMMARY OF DESIGN ACTIVITIES

Type of software:

Component-based practice environment.

Subject-matter:

Relations - domain perception, their properties and special kinds.

Nature of instructional transactions:

- some for passive perusal,
- others facilitating perception, and most
- ensuring active participation.

Basic components for each relation:

- definition,
- visual aid,
- example-synthesis,
- seven properties, each in three modes,
- five analyses for kind.

Help components:

- nine definitions,
- twelve math-symbol elaborations.

Courseware Engineering using SE principles:

- object-oriented design paradigm,
- appropriate representations.

9.4 THE FRAMES PROTOTYPE

Some prototypes are implemented by an author-developer actually **designing on-screen.** This methodology was not followed in FRAMES, where the project was too complex and comprehensive for omission of traditional design. Several conceptual ideas were tested for feasibility and aesthetics by the author-designer and programmer sitting together at a computer, but all components, instructional transactions, feedback and control strategies were designed on paper by the researcher before implementation by the programmer.

This section outlines the features of FRAMES not incorporated in previous sections. It describes the prototype, refers to installation instructions, suggests a walk-through path, and discusses control and usability factors. The prototype is used to implement a "design-and-refine" paradigm, and the section goes on to describe its role in the refinement of requirements and design.

9.4.1 THE NAME "FRAMES"

A FRAMES user makes three decisions each time he selects a component to "view or to do":

1. Which **R**elation to choose: When the user selects a relation as the current problem, its comprehensive definition component appears on-screen in a "blackboard" form.

2. Which **A**ttribute to view, apply or test: The attributes options include viewing the *graphic aid*, *synthesizing example members* for the relation, doing an exercise to test a *property of the relation*, or testing several properties to determine the *kind of relation.* Should a learner opt to test a *property* of the relation, he is offered six from which to choose; should he select the option to test for a particular *kind*, he is shown five kinds from which to choose and is then presented with a composite problem, integrating the appropriate property tests for that kind.

3. In which **M**ode to do the exercise: Mode 1 presents a read-only proof; Mode 2 offers guided practice, giving the learner the opportunity to fill in blanks in the definition, in an R.T.P and in final proofs; Mode 3 (DIY) encourages a more independent approach where the learner supplies complete steps of a proof for line-by-line assessment.

In CAI terminology a *frame* refers to a screen presentation. The acronym above, **R A M,** was set within *frame* to name the system, **"F<u>RAM</u>ES"**. Its *frames*, however, are not full-screen presentations, but window-style components, combining to form a screen display.

9.4.2 THE SCREEN OBJECTS OF FRAMES

Coad & Yourdon notation sets out the attributes and services of an object class. It is particularly useful in modelling both concrete and abstract objects, and not just direct classes of software objects. Figure 9.7 represents and models interaction (messages) between, the user, the functional area of the screen, the concept of exercise practice, and certain features of the relation description. None of these correspond precisely to previously identified objects.

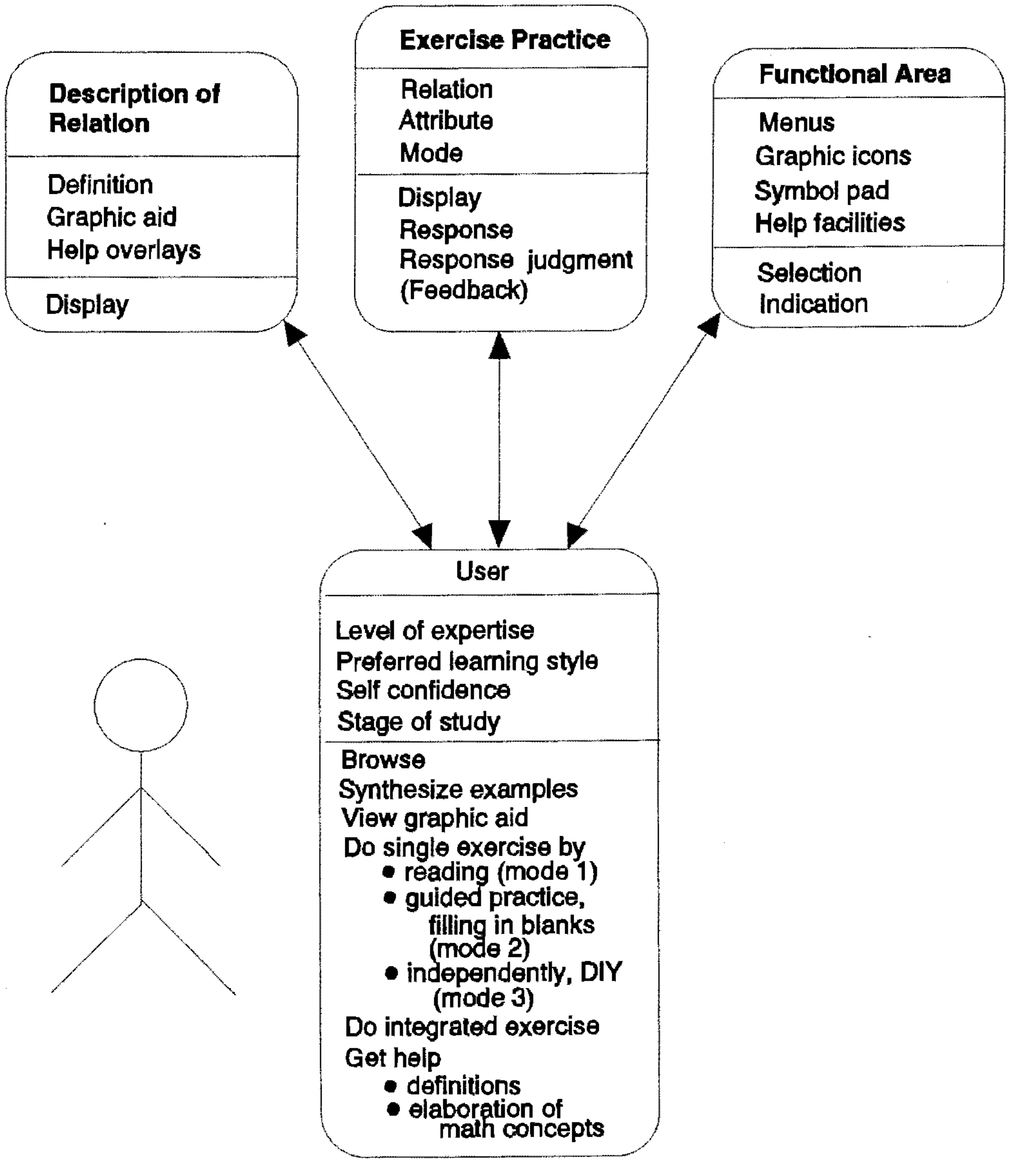


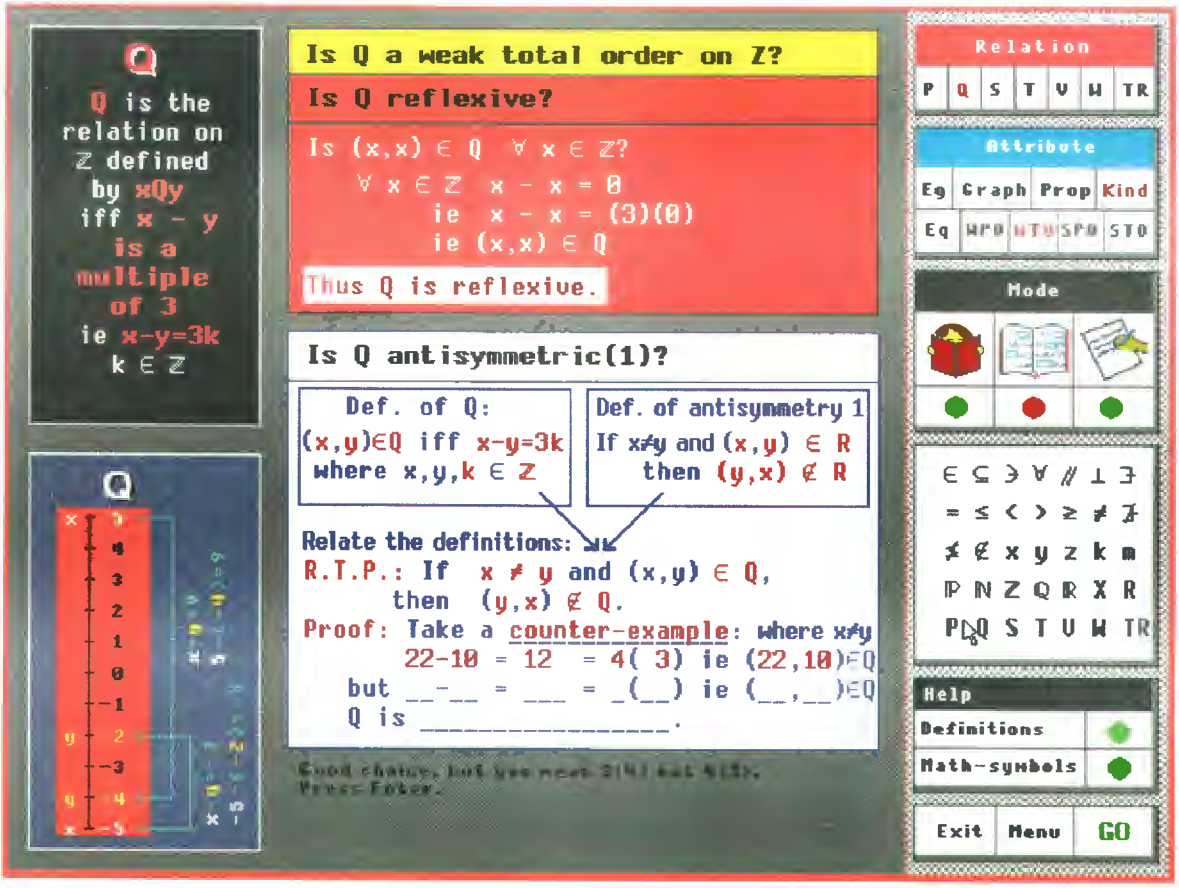
Figure 9.7
The Screen-related Objects of FRAMES

FRAMES screen layouts have clean lines and a well-structured aesthetic appearance. The **right hand side** is a functional area dedicated to control options and utilities. The R A M options are available as a mini-menu in a curtailed control area. The proofs in modes 2 and 3 require mathematical manipulations, so the functional area includes a symbol pad of mathematical operators and notations on which the user can click to enter them in his proof. Access to the <Help> components is also controlled from this right hand functional area. At the extreme bottom right are the <GO>, <Menu> and <Exit> buttons.

The **left hand region** of the screen is devoted to description of the current relation, displaying the definition 'blackboard" and, if called up by the user, the graphic aid which portrays that relation visually. Further facilities available are <Help> utilities that present concept definitions and mathematical definitions on request. These appear as overlay menus and elaborations on the left, and can easily be cleared.

The **central region** presents the user's requested components. He can select several relating to the same relation, in which case the left region remains static, while the chosen exercises scroll up the central region. Selecting the *kind of relation* option automatically produces the necessary series of tests. Changing the relation produces a new screen configuration as the new definition appears. The D.I.Y. (do-it-yourself) exercises in Mode 3 demand considerable independent manipulation from the student, and are accompanied by a more extensive symbol pad on the right. The example-synthesis exercise occurs in a standard mode only.

Screen Display 9.17 shows how Figure 9.7 is in congruence with the actual FRAMES screen design, subdivided into distinct left hand, central and right hand functional areas.



Q

Q is the relation on $\mathbb{Z}$ defined by xQy iff $x - y$ is a multiple of 3 ie $x-y=3k$ $k \in \mathbb{Z}$

Is Q a weak total order on Z?

Is Q reflexive?

Is $(x,x) \in Q \ \forall x \in \mathbb{Z}$?
$\forall x \in \mathbb{Z} \quad x - x = 0$
ie $x - x = (3)(0)$
ie $(x,x) \in Q$

Thus Q is reflexive.

Is Q antisymmetric(1)?

Def. of Q:
$(x,y) \in Q$ iff $x-y=3k$
where $x,y,k \in \mathbb{Z}$

Def. of antisymmetry 1
If $x \neq y$ and $(x,y) \in R$
then $(y,x) \notin R$

Relate the definitions:

R.T.P.: If $x \neq y$ and $(x,y) \in Q$,
then $(y,x) \notin Q$.

Proof: Take a counter-example: where $x \neq y$
$22-10 = 12 = 4(\ 3)$ ie $(22,10) \in Q$
but __-__ = ___ = _(__) ie (__,__)$\in$Q
Q is __________________.

Good choice, but you need 3(4) not 4(3).
Press Enter.

Screen Display 9.17

Most of the screen objects listed in Figure 9.7 are encountered in this Screen Display.

The left-hand functional area, dedicated to *Description of the Relation*, shows a graphic aid and the definition "blackboard". Use of <Help> is not depicted here, but is shown in Screen Displays 9.2, 9.8 & 9.13.

The central region, devoted to *Exercise components*, shows two instructional transactions. The second is incomplete, showing a user-response to a step in the proof and the feedback that resulted from its judgement. This screen demonstrates two different kinds of mathematical proof techniques:

1. A proof for the *general case*, used to prove that something **is** true.
   See the proof that **Q** is reflexive.

2. A counter-example disproof, used to prove something **is not** true.
   See the uncompleted proof to show that **Q** is not antisymmetric.

3. The right-hand **functional area** is a compact control area, offering all the functionality of the main menu, but in reduced space. In addition it presents a symbol pad (or soft keyboard) of mathematical symbols not available on the hard keyboard. Help facilities are also accessed here, although they appear on the left.

The final object, the external user, must be imagined.

9.4.3 THE PROGRAMMING CODE OF FRAMES

The major modules, programmed in TenCORE 5.0, are listed below. The body of the program comprises interactive modules, all simultaneously active:

- *Initialization* - presents introductory screens; sets up physical screen structures.

- *R A M executive control* - Accepts user's input from main menu or from abbreviated right hand control area and accesses module/s for selected component; the control module has submodules for:
  - highlighting and de-highlighting selected options
  - step-scrolling mechanism for use when centre-screen is full.

- *Help modules* - to access, elaborate and clear Help components.

- *Symbol pad control* - to import math symbols into cursor position, by mouse-clicking on the symbol pad.

- *Modules for relation definitions* (top left components).

- *Modules for relation graphic aids* (bottom left components).

- *Modules for relation example synthesis* (a central component).

- *Modules for "property" components in Modes 1, 2 and 3* -

  - A single module for each Mode 1 proof.

  - Two modules for each Mode 2 proof:
    - proof framework
    - student-interaction handler.

  - Two modules for each Mode 3 proof:
    - proof framework
    - student-interaction handler.

  - Two further modules for each Mode 2 proof:
    - relation-definition-block handler
    - property-definition-block handler.

- *Judgement and feedback modules -*
  - Module to assess student-response in Mode 2
  - Module to provide feedback to student-response in Mode 2
  - Module to assess student-response in Mode 3
  - Module to provide feedback to student-response in Mode 3.

- *Modules for "kind of relation" components -*
  (The *kinds* are five composite components, each calling a set of property components)
  - Module to assess and provide feedback to student-response in regard to definition of the kind
  - Module to call the appropriate property test modules.

- *The FRAMES program has knowledge (see section 9.3.6) of:*
  - composition characteristics of each special kind of relation,
  - structure of the ordered pairs of each relation, and
  - physical screen size of each component (knowledge necessary for scrolling mechanism).

9.4.4 A WALKTHROUGH OF FRAMES

Installation and usage instructions are supplied with the FRAMES diskette in the back cover of this dissertation.

In Appendix A the same information is provided, along with a suggested walkthrough path of FRAMES, designed to give the reader a representative overview of the content of the prototype.

9.4.5 ASSESSMENT OF FRAMES' CONTROL STRUCTURE

The philosophy underlying navigation through the FRAMES environment and the selection of sequence, content and style is **non-sequential user-control**. The only instance of program-control occurs when a user selects the attribute *kind*, opting to test whether the relation is an equivalence relation, a weak partial order, a strict partial order, a weak total order or a strict total order. In these cases the respective sets of property tests are presented automatically, although the user reserves the right to change mode between tests.

FRAMES is assessed according to Midoro's measuring grid (see section 8.1.3) which rates the criteria of *navigability, adaptivity and reactivity*, and positioned within Hammond's framework (see section 8.7.1) which assesses the dimensions of *control, engagement and synthesis*. Due to the practical implications of effectively portraying a three-dimensional representation on a two-dimensional medium, the control- and navigation-related factors of FRAMES are assessed by linear ratings in Figure 9.8.

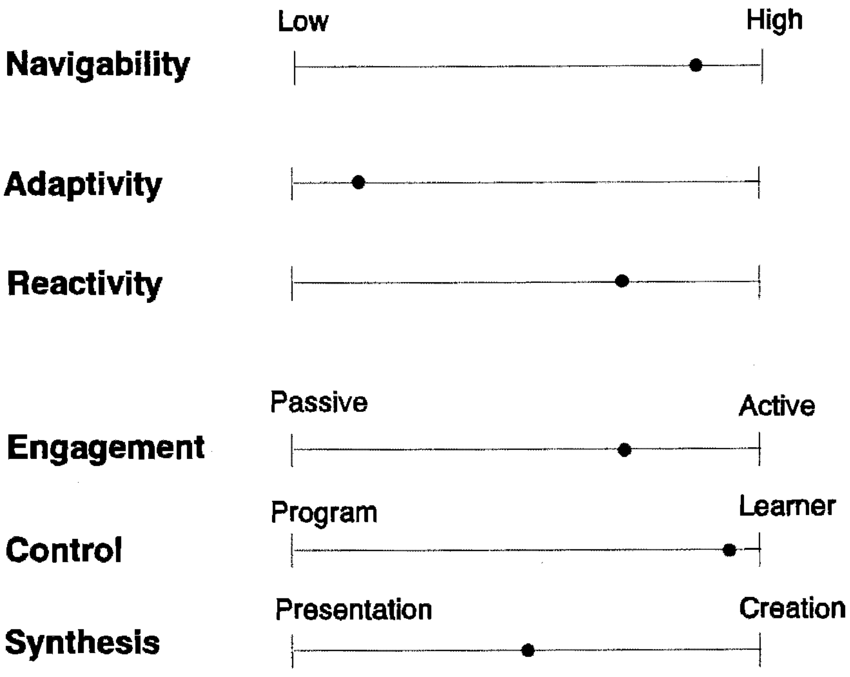


Figure 9.8
FRAMES assessed according to Midoro's Grid and Hammond's Framework

It rates high on navigability, fairly high on reactivity, and low on adaptivity, due to the complete absence of program control. There is total learner-control and highly active engagement. The system also offers limited opportunities for creative synthesis.

9.4.6 USABILITY FACTORS OF FRAMES

Usability relates to the quality of interaction between the user and the system. Several of the criteria given in section 8.6.2 apply to conventional information systems processing data-streams and are not relevant to FRAMES, but others are desirable in instructional software.

The system portrays a certain *familiarity*, in that the <Help> facilities have some resemblance to a windowing environment. The command structure and operation of FRAMES are *predictable* and consistent. Should unanticipated key actions on the user's part cause error conditions or interrupt flow of a proof, *recoverability* is attained by clicking on <Menu> and recommencing the component. A user who is "bogged down" in a proof and does not wish to complete it can click on <Menu> at any time to quit that component and start another.

With regard to *learnability*, the system is easy to operate:

♦ An interactive "demo" introduces the user to the three modes of operation (Screen Display 9.19).

♦ The first component, or *frame,* is accessed via the main menu by **R A M** selection, activated by a click on <**GO**> when the user is certain of his choice.

♦ Further components are selected from the curtailed control area, or mini-menu, on the right, again using **R A M** plus <**GO**>. They are added cumulatively to the screen.

9.4.7 USER INTERFACE OF FRAMES

FRAMES lies on the spectrum between a character-based and a graphically-based user interface (see section 8.6.3), but closer to the latter. Where possible, visual icons are used, but most concepts are better represented by concise text. Navigation is by mouse control. FRAMES has similarities to a WIMP environment, with its full, yet sub-divided screens, its icons, mouse-control and clicked-in menus. The exit techniques from Help facilities, by means of mouse-clicks on a slot, are similar to those in the Microsoft Windows interface. Computer Science students should have no difficulty in running it. A read-and-write metaphor is used in the icons for the three modes and depicted in Screen Display 9.18:

MODE	USER-ACTION	PICTORIAL ICON	INDICATING
1	Read-only	Open book	Straight reading of a proof
2	Guided practice	Written page with blanks	Blanks to be filled in by user
3	D.I.Y	Pen to paper	Largely independent action

Even a Mode 1 "read-only" perusal requires a measure of interactivity - proofs are presented step-wise, and the student must mouse-click or press <Enter> to move on. Student responses in Mode 2 and 3 exercises are input by a combined strategy, using the keyboard for standard text and mouse-clicks on a symbol pad to import mathematical symbols. The learner adjusts easily to this joint action of key strokes with the left hand and mouse-clicks on the right.

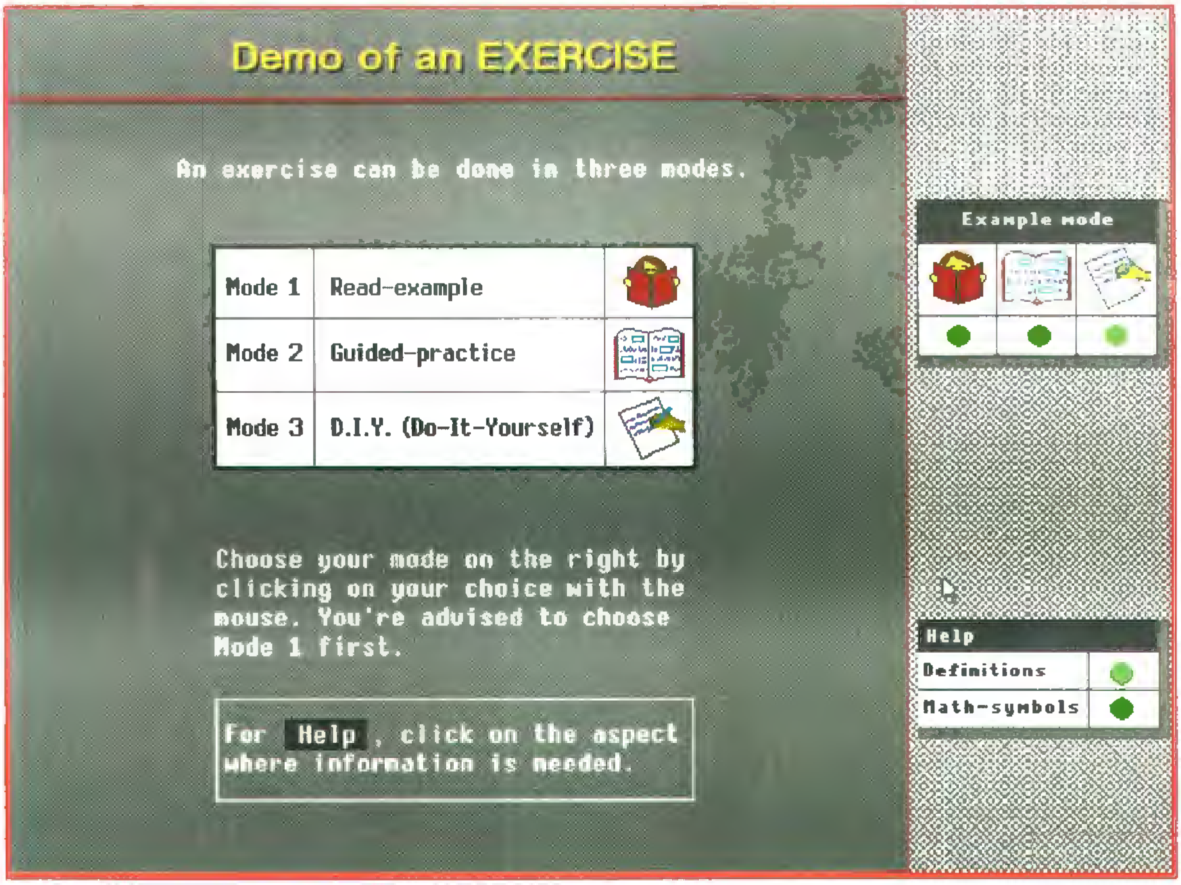
The discussion in section 8.6.3 of the characteristics of a good user interface described the current tendency away from low-density screen displays, and towards increased resolution screens with well-structured information. FRAMES screens are in line with this trend and, as recommended, the screen is partitioned to minimize distraction and focus attention on the task in hand. The structure is as congruent as possible to the underlying structure of the problem to be solved. Control is to the right, reference material appears to the left on request, and the format of the proof structure is as in a written proof. As far as space permits, appropriate alignment is used. These points are evident from the proofs shown in Screen Displays.

The user can easily *change* his selection of a component, since selection is not binding until he clicks on <GO>. The only selections activated immediately, without being on hold for <GO>, are the definition "blackboards". As the user clicks on a relation (any one of P, Q, S, T, V, W or TR), its definition appears instantly at top left. This enables him to browse and inspect prior to selecting a component for his next instructional transaction.

If he wishes to *cancel* a transaction during use, he can do so by clicking on <Menu> and making another selection. To *quit* the FRAMES session altogether, he must click on <Exit>.

The system is highly responsive as described in point 6 of section 9.3.5. Feedback is provided for each intermediate stage of a proof, correcting errors before the learner proceeds to the next step.

To familiarize the user with the environment, a *demo* in each of the three modes is available. Screen Displays 9.18 and 9.19 show how the user accesses and uses a *demo.*



Demo of an EXERCISE

An exercise can be done in three modes.

Mode 1	Read-example
Mode 2	Guided-practice
Mode 3	D.I.Y. (Do-It-Yourself)

Choose your mode on the right by clicking on your choice with the mouse. You're advised to choose Mode 1 first.

For Help, click on the aspect where information is needed.

Screen Display 9.18

The graphic icons used as a metaphor to represent the three instructional strategies, or modes, appear on the main menu (Screen Display 9.9) and in the mini-menu on other screens.

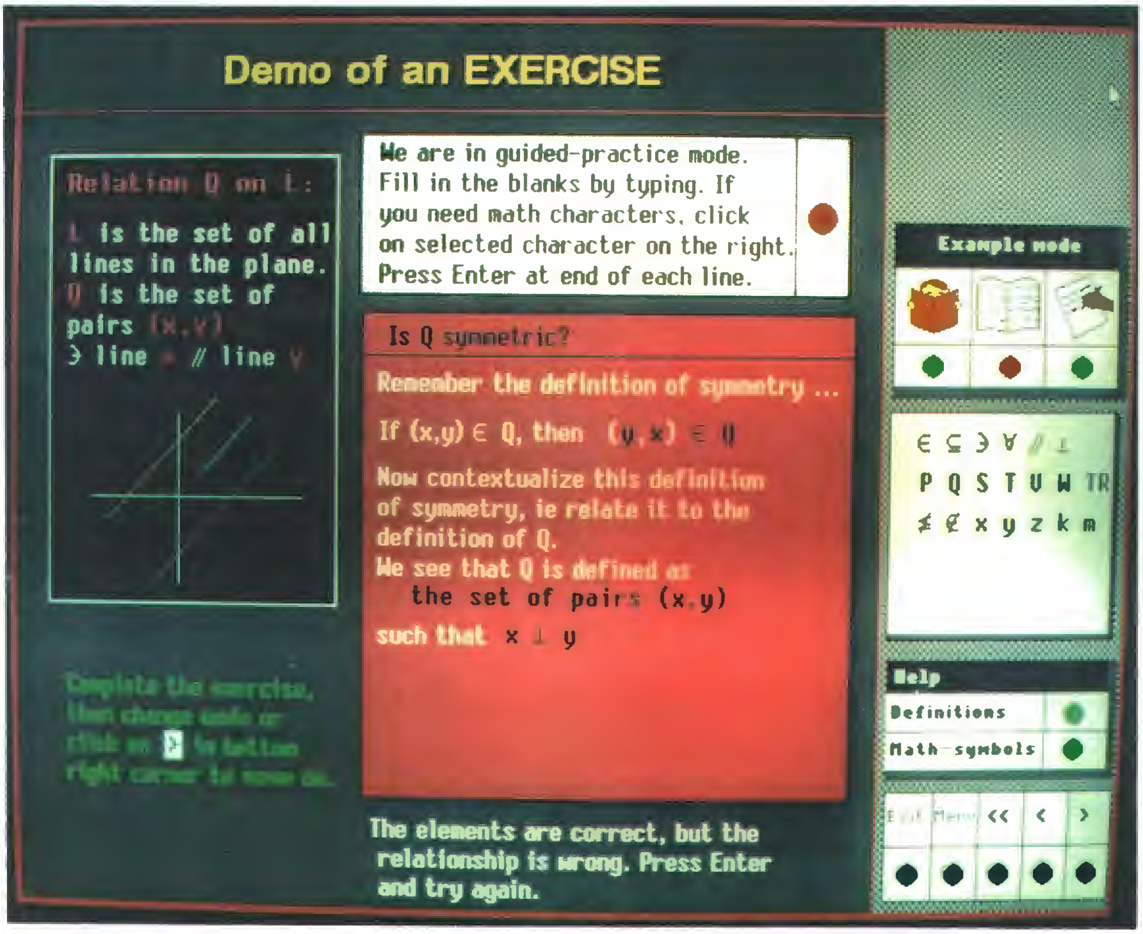
Screen Display 9.14 shows the screen entitled "What YOU can DO in FRAMES", which offers access to the *learn-by-doing demos*. On calling the demos, but prior to actual use, the user is presented with the screen above, which introduces the three modes:

Mode 1 - read-only, is symbolized by an open book

Mode 2 - guided practice, is represented by a printed page with blank spaces

Mode 3 - D.I.Y. (do-it-yourself) is indicated by a pen-to-paper graphic.

Selection of a mode from the right-hand control area produces a worked-proof for Mode 1, and interactive proofs for modes 2 and 3. Screen Display 9.19 shows the Mode 2 version.



Demo of an EXERCISE

Relation Q on L:

L is the set of all lines in the plane.
Q is the set of pairs (x,y) ∋ line x // line y

We are in guided-practice mode. Fill in the blanks by typing. If you need math characters, click on selected character on the right. Press Enter at end of each line.

Is Q symmetric?

Remember the definition of symmetry ...

If (x,y) ∈ Q, then (y,x) ∈ Q

Now contextualize this definition of symmetry, ie relate it to the definition of Q.
We see that Q is defined as
the set of pairs (x,y)
such that x ⊥ y

Complete the exercise, then change mode or click on > in bottom right corner to move on.

The elements are correct, but the relationship is wrong. Press Enter and try again.

Screen Display 9.19

The Mode 2 *demo* offers a limited symbol pad, but full *Help* facilities.

While doing a *demo*, the user is compelled to complete the proof; clicks on <Menu> or <Exit> are not accepted. In real usage of the FRAMES system, however, he may quit during an instructional transaction by going to the menu or exiting from the practice environment.

Meaningful, specific feedback is provided as shown above, where the user clicked on the symbol for *perpendicular* instead of *parallel*.

9.4.8 INSTRUCTIONAL EVENTS WITHIN FRAMES

Gagné's nine events of instruction, intended to provide external conditions of learning, are introduced in section 4.2.3 and applied to CAI in section 6.3. Several are incorporated in FRAMES. Examples are:

3. *Stimulating recall of prior learning*:
   Definitions are accessible via <Help>; context setting is provided in Mode 2.

4. *Presenting stimuli with distinctive features*:
   Proof structures and cuing in Modes 2 and 3 provide an infrastructure for problem-solving.

5. *Guiding learning*:
   Worked examples in Mode 1 demonstrate exactly how to test relations for particular properties.

6. *Eliciting performance:*
   A prime aim of the environment is active participation.

7. *Providing feedback*:
   As stated, remedial feedback is provided in Mode 2 and more limited feedback is available in Mode 3.

9. *Enhancing retention and transfer*:
   Variety of examples exposes the learner to similar analyses in different contexts, as he tests the same property in different relations.

These strategies should help the student to think principially and to contextualize initially abstract concepts.

9.5 FURTHER CHARACTERISTICS OF THE FRAMES PROTOTYPE

Many features of FRAMES and its behaviour have been mentioned in this chapter. This section complements the previous ones by mentioning aspects not discussed elsewhere.

9.5.1 PROBLEMS AND IDIOSYNCRASIES OF FRAMES

- Some unpredictable behaviour by <Help> when it is called or dismissed at unanticipated times.
- In student-response to first question in a *kind of relation* attribute, the theoretically correct response, "reflexive on Z", can't fit in, so "reflexive" is acceptable.
- A double <Enter> or a double mouse-click is necessary to continue with an exercise after use of Help.
- An apparent weakness that is actually a deliberate strength: When the learner moves on to a different relation, the new definition appears in the blackboard position, but the exercise/s from the previous relation remain on-screen in the central exercise region. This is done so that the learner may, if he wishes, compare the same exercise, proof or disproof from two different relations.
- Space was sometimes a problem and various subtle omissions were made to fit in essential material. As an example, in one case, the period, ".", was omitted after the "P" in "R.T.P.", in order to fit in the words "that if" to emphasize it was, in that case, a goal rather than a consequence.
- On rare occasions an unpredictable "execution error" occurs and terminates the run.
- Spelling tolerance can cause its own problems. In the tests for "Kinds of relations", *reflexive* and *irreflexive* are considered synonymous by the spelling-tolerator, and both are accepted as a valid condition for an equivalence relation, whereas only *reflexive* is correct. The solution was to exclude spelling tolerance, and to build in a set of acceptable mis-spellings.
- Mode 3 presents the learner with continuous lines for his response, so as not to implicitly prompt. On occasions two short lines are separated by an adverb or a conjunction. If a user erroneously mouse-clicks in a large element, e.g. " (x,y)", so that it exceeds the response area and overwrites a hard-coded word, then <Backspace> will not clear it and the line will be judged "Incorrect".

9.5.2 THE PROTOTYPE AND MODIFICATION OF REQUIREMENTS

As discussed in section 3.4.2, Lantz describes how initially fuzzy areas can evolve into refined definitions and precise designs by using a prototype. The construction and evaluation of FRAMES has demonstrated the truth of this statement. The prototype has been tested, but not exhaustively. The aim was to demonstrate the feasibility of presenting a variety of activities in a practice environment and not, at this stage, to create a robust product for release among the target population. Errors and inadequacies are present, which would have to be corrected before any release as a production model.

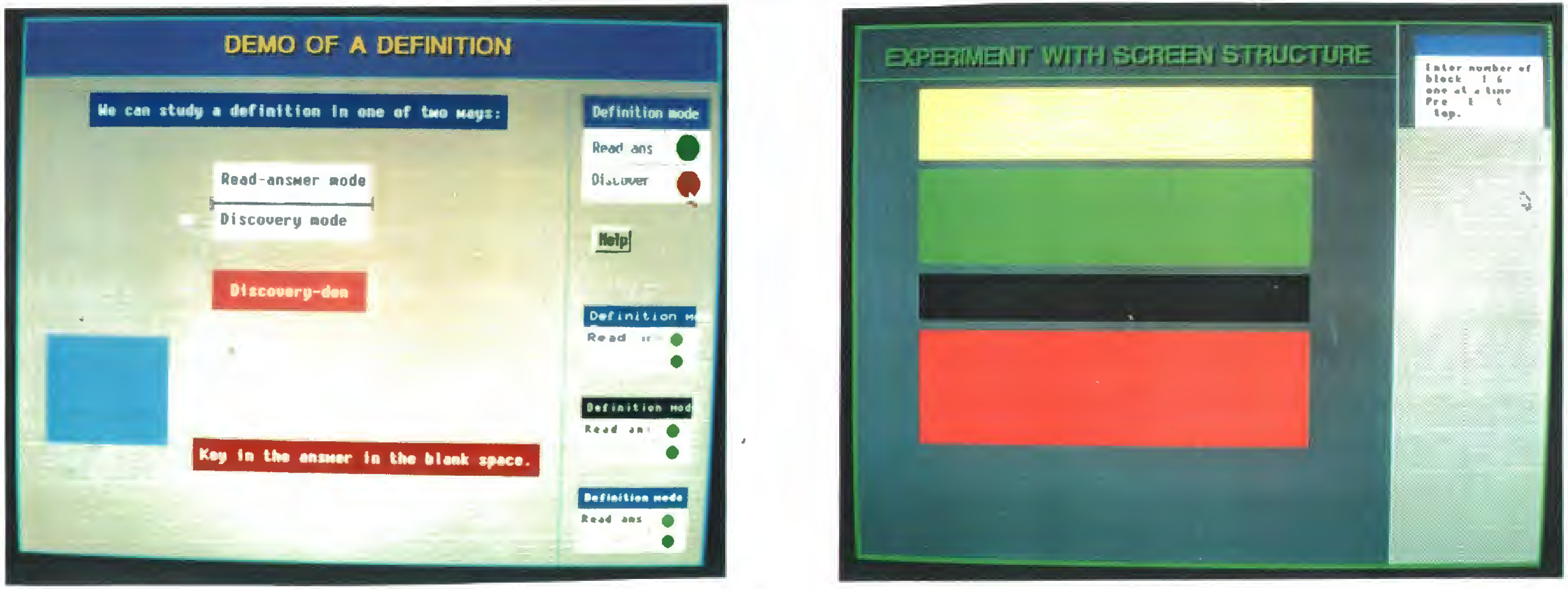
Initial feasibility testing, shown in Screen Displays 9.20 and 9.21, was done on a pre-prototype *mock-up*:

- to implement the step-scrolling mechanism, and
- to perfect mouse click access and importation of mathematical operations and symbols.

A prime purpose of a prototype is the modification of imprecise or unrealistic requirements. On actually running courseware, problems show up that were not evident on a paper-based design. Some examples from FRAMES were:

- The relations were originally called P, R, S, T, V, W and TR. The name R was also used for the *general case*, appearing in definitions of properties and Mode 2 tests. This caused confusion and ambiguities, so the name of the second relation was changed to Q.
- The definition of anti-symmetry can be phrased in two ways. Different learners feel intuitively more comfortable with different definitions, so the designer, in a magnanimous spirit of individualization, opted to incorporate both in FRAMES and to present all antisymmetry exercises in two alternative ways. On viewing the prototype this appears cumbersome and confusing. In retrospect it was somewhat of an over-kill. It would have been better had the designer *cum* lecturer *cum* subject-matter expert selected a single approach and stuck to it.
- Obtrusive interactions occurred between <Help> and exercise components when <Help> was called at unanticipated times.
- There are inadequacies in the feedback to Mode 3 examples. Feedback was partially redesigned after tests with the prototype, but anticipation of all possible responses is a huge task, beyond the scope of this study.

Evaluation of the prototype has been system-oriented, towards usability and general instructional factors. It has not been focused on the strictest purity of the mathematical aspects. Still required are assessment of the material by a subject-matter expert to ensure utmost mathematical precision in all details, and learner-evaluation to identify ambiguities and possible weaknesses in control and content.



We can study a definition in one of two ways:

Key in the answer in the blank space.

Enter number of block 1 6 one at a time Pre t t top.

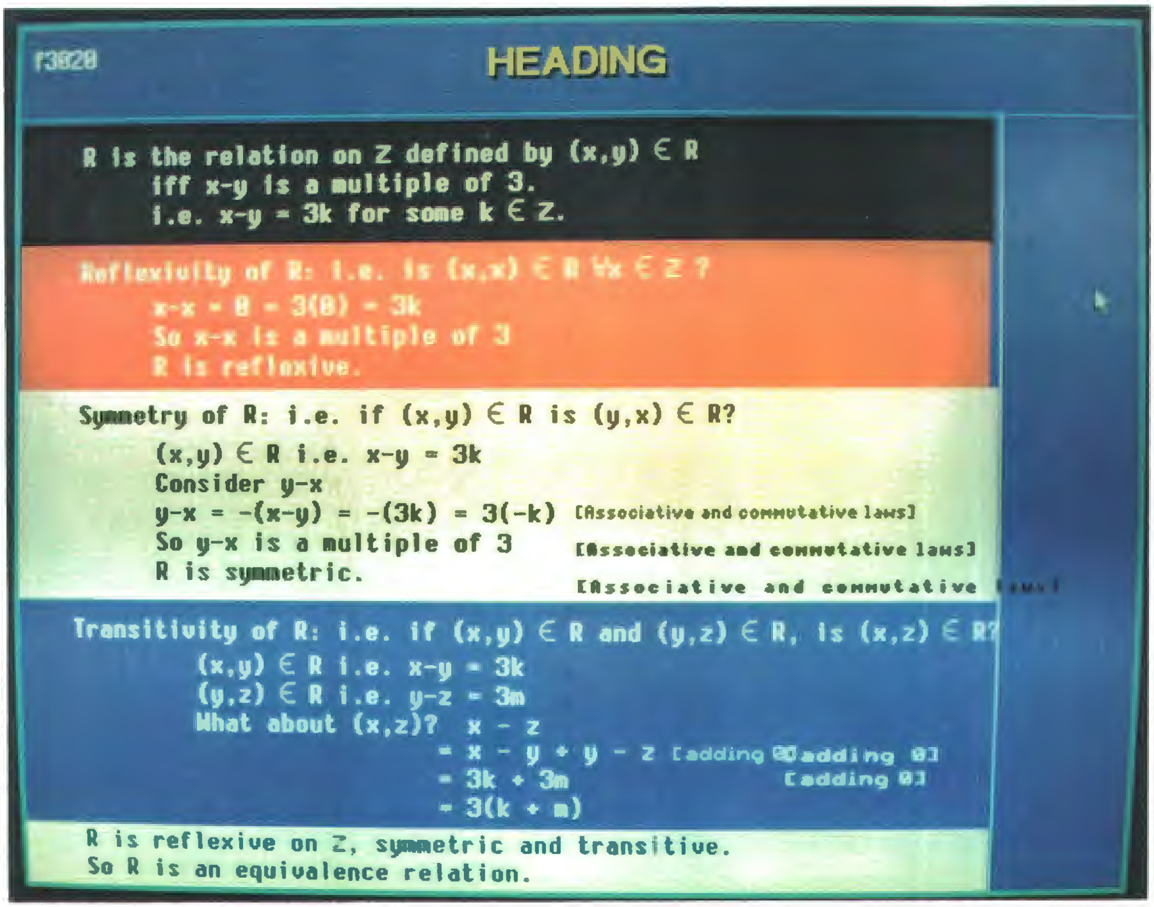
Screen Display 9.20

The left-hand screen display shows initial feasibility testing of user-control by mouse-clicking. The mouse-arrow remains in place where the "user" selected the second mode, *discovery*.

The screen was also used to experiment with fonts and colours. FRAMES was dependent on a satisfactory small font for its very existence. The version at bottom right is the style adopted for the FRAMES control area.

These issues, as well as the importation of mathematical characters from the symbol pad, had to be resolved before even the most rudimentary development could take place.

The right-hand display depicts the screen structure experiment which was used to perfect the step-scroll mechanism.



```
f3020                       HEADING

R is the relation on Z defined by (x,y) ∈ R
     iff x-y is a multiple of 3.
     i.e. x-y = 3k for some k ∈ Z.

Reflexivity of R: i.e. is (x,x) ∈ R ∀x ∈ Z ?
     x-x = 0 = 3(0) = 3k
     So x-x is a multiple of 3
     R is reflexive.

Symmetry of R: i.e. if (x,y) ∈ R is (y,x) ∈ R?
     (x,y) ∈ R i.e. x-y = 3k
     Consider y-x
     y-x = -(x-y) = -(3k) = 3(-k) [Associative and commutative laws]
     So y-x is a multiple of 3    [Associative and commutative laws]
     R is symmetric.
                                  [Associative and commutative laws]

Transitivity of R: i.e. if (x,y) ∈ R and (y,z) ∈ R, is (x,z) ∈ R?
        (x,y) ∈ R i.e. x-y = 3k
        (y,z) ∈ R i.e. y-z = 3m
        What about (x,z)?  x - z
                         = x - y + y - z [adding 0[adding 0]
                         = 3k + 3m               [adding 0]
                         = 3(k + m)

  R is reflexive on Z, symmetric and transitive.
  So R is an equivalence relation.
```

Screen Display 9.21

This initial presentation of an equivalence relation analysis differs greatly from the ultimate FRAMES screen design. The viewer perceives a *whole*, rather than *components*, and the concept of left-hand and right-hand functional regions had not begun to evolve. Once again, various fonts and styles were tested.

The similarity to a flag is not an accident. The original intention was that a completed proof would appear in such a form, but in line with the considered policy of "no gimmicks, bells and whistles" the researcher threw this one out!

9.5.3 SUMMARY OF PROTOTYPE CONSTRUCTION

CYCLIC ITERATION OF THE FOLLOWING STEPS:

Study of requirements

Design of:

- instructional strategies,
- control logic,
- screen objects and design.

Prototype Implementation of above using:

- Tencore 5.0,
- graphics software.

Hands-on testing and evaluation of:

- control system,
- instructional strategies and activities,
- user interface and other usability factors.

Revision entailing:

- refinement of requirements,
- modification and expansion of design,
- associated changes to program code.

9.6 CONCLUSION

This chapter was a phase by phase account of the courseware engineering life-cycle model of FRAMES, i.e. a description of the development process comprising requirements analysis, design and prototype construction. There were two fundamental goals in the research and development process. The first was the implementation of an object-oriented design and the second was development by a prototyping life-cycle model.

With reference to the first factor, the component-based approach lends itself to an object-oriented implementation. The prototype FRAMES system is perceived as being object-oriented, yet it was built using the non-object-oriented authoring language, TenCORE 5.0. Although this situation was successfully achieved, it was somewhat "forced". The ideal case would be definition of the components and instructional transactions as object classes with set standards. Instead, each instance of a class is separately coded. For example, on changing colour standards in a certain type of exercise, the change has to be made for each of the fourteen instances in the prototype.

A further frustrating process occurred in positioning exercise components on the screen. Their step-scroll mechanism is implemented by separately re-positioning each line of text, instead of treating the entire block as an object and supplying a single set of co-ordinates. This problem occurs as a result of the non-windowing authoring environment.

A strong features of the OOP, the class feature, is not fully utilized. FRAMES bears no resemblance to, for example, the archetypal bank account example used to demonstrate the utility of a multiclass- and inheritance-based structure. The "kinds of relations" are, however, subclasses or specializations of relations, and *inherit* existing code for the required tests.

A further characteristic of the object-oriented paradigm is software re-use, and this has been achieved. The program is coded in a structured, modular fashion, and modules are continually re-used. A simple example is re-use of the fill-in-the-blank property definitions used in Mode 2, which are coded once for each property and called by every relation. Similarly, as mentioned in section 9.5.1, the tests for *kinds* use existing modules for all their constituent components except the brief introductory and concluding sections. Content-free re-use of the software structure is a long term goal, discussed further in Chapter Ten.

The second engineering goal, namely to develop instructional software by prototyping, was successful. Due to factors such as the new ground broken by FRAMES, the full, yet compact, and highly interactive screens, the roles of both keyboard and mouse in user-control and user-response, and the three different modes for doing exercises, the prototyping life-cycle proved to be of far more value than a development methodology with a detailed paper-based specification and design. Evaluation and empirical use of the prototype resulted in many changes to the visual, functional and instructional aspects. TenCORE proved itself an excellent authoring system for rapid prototyping.

In the conclusion of Chapter Five reference was made to

- teaching as the deconstruction of knowledge
- learning as the construction of knowledge.

The component-based approach permitted **decomposition of the topic of relations** into its constituent parts, with relation both to content and to the type of performance expected. The prototype is a *working model* of the envisaged FRAMES practice environment, and an extended system will be released as a *production model* for use by the target population. The nature of the envisaged conversion is addressed in Chapter Ten. It is hoped that the software will facilitate analysis and synthesis of the topic by the learner, and result in **construction of his own experiential knowledge**.

CHAPTER TEN

CONCLUSION

This MSc half-dissertation was a multi-disciplinary study, which aimed to integrate a software engineering approach with instructional factors in the decision-making, analysis, design and development processes of instructional software. Software engineering models, tools and representations were used in the *process* of software construction. With reference to the fundamental characteristics of the software *product*, several disciplines and factors, from both instructional and computing perspectives were considered, and the most appropriate approach/es selected. Software engineering, instructional design and instructional theory fulfilled their stated roles (Figure 1.1) as pillars of courseware engineering.

The object-oriented design paradigm and a prototyping life-cycle model were found to be most suitable for development of computer-aided instruction. The conceptual study was illustrated by prototype development of a component-based multi-activity practice environment that offers the learner perusal or practice according to his preferred learning style or need.

In conclusion, two questions can be posed:

> "What has been achieved?" and "Where does it lead?".

10.1 WHAT HAS BEEN ACHIEVED?

The research makes three main contributions:

1. **Application of SE to CAI - the process**

   The study integrated software engineering principles with instructional factors in the development *process* of instructional software, using a prototyping life-cycle model. Each of the multi-disciplinary factors considered, as well as conventional instructional design procedures, were related to the appropriate phase/s of the courseware life-cycle. The use of software engineering practices and the application of object-oriented design in ISD are in line with the "Megatrends of Instructional Computing" identified by McLean [McLean 1989].

The prototyping life-cycle demonstrated its worth in the context of instructional software and is recommended for CAI development.

2. Application of Merrill's CDT to CAI - the product

The primary goal of the research, point 1 above, was successfully attained. However the researcher feels equally enthusiastic about the component-based, user-controlled instructional software *product* developed to accompany the theoretical study. The resulting FRAMES prototype is a breakthrough in the type of software developed in South Africa, an *androgogic activity box* of varied instructional transactions driven by user-initiative, another of McLean's envisaged "megatrends". The application of Component Display Theory and the consequent deconstruction of material into components is an excellent starting point for any instructional development. The components can be categorized according to type of content and the strategies used for presentation or elicitation of performance from the student.

3. Development of a prototype practice environment for COS101-S at Unisa

The situation selected for development of prototype software to illustrate the concepts and principles encountered in the study is a practice environment in the topic of relations in Theoretical Computer Science. An extensive literature survey was undertaken of each factor impacting on instructional software and, in each case, an appropriate approach was selected for the identified need in COS101-S. Table 10.1, a sequel to Table 1.1, shows the factors and approach/es.

It is hoped that the system will fulfil its promise and meet the identified need. Depending on the authoring system used, the decision must be made whether to:

- extend the existing FRAMES into the final version, thus using it as an *evolutionary* prototype, or
- build a new system, using an object-oriented development environment, in which case the initial FRAMES would play the role of *throwaway* prototype.

The final production system would include a complete set exercises for all seven relations, more extensive feedback for exercises in Mode 3, and correction of errors and inadequacies. Its development process should also incorporate pilot testing by end-users.

Table 10.1 Factors Investigated and Approaches Selected for FRAMES

DISCIPLINE/FACTOR	APPROACH USED IN FRAMES
Theories and models of thinking, learning and instruction	Cognitive science, implemented with a constructivist approach; aspects of behaviourism.

Instructional design	*Practice environment* with a component-based design;
	highly interactive; strong on user-control and individualization;
	presenting different activities in a variety of modes, offering lesser or greater support, providing perusal and practice in subskills and composite skills.

Context, subject-matter and target population	Distance education of tertiary-level Computer Science students on *relations*, a topic from discrete mathematics.
Software engineering methodologies, models and tools	Object-oriented paradigm; prototyping life-cycle model; use of software engineering tools and representations; development of prototype in TenCORE 5.0.
The object-oriented design paradigm	Object-oriented design; components viewed as objects, non-prespecified screen composition.
Artificial intelligence	Not an ICAI or ITS; an intelligent support environment with limited knowledge.
Control, hypertext, usability and user interface factors	User-control, high navigability; object-based control structure; usability features and user interface resembling windowing environment.

10.2 WHERE DOES IT LEAD?

The project opens further research opportunities:

1. **Investigation into use of the software structure as a shell**

   Using Merrill's assumption [Merrill 1990b; Merrill 1991] that **instructional strategy is independent of the knowledge to be taught**, it is hoped that the object-based design and control structure developed for FRAMES can be used as a *generic, content-independent shell* to present tuition, exercises, and graphic representations for varying subjects and content, and in different modes or instructional strategies. Such use, both of the content structure and the control structure, would capitalize on the modularity and re-use aspects of the object-oriented design, but could be implemented more efficiently in an object-oriented development environment.

2. **Research into alternative programming techniques**

   The concluding phrase of the previous paragraph points to the need for investigation into alternative authoring environments, optimal for the creation of dynamic instructional systems with window-based screen layouts, such as FRAMES.

APPENDIX A

1. HARDWARE REQUIREMENTS

Computer with a mouse and a colour monitor, DOS operating system, at least a VGA graphics card, hard disk with at least 1 Mb available, 640K memory and a high-density 5.25" or 3.5" diskette drive. FRAMES looks particularly good on a 17" Pentium screen.

2. INSTALLATION

The CAI lesson FRAMES and a TenCORE driver are available on a high density diskette.

1. Create a hard disk subdirectory. Key in:
   md frames

2. Change to that directory:
   cd frames

3. Insert the diskette into the A drive (or B):

4. Copy the contents of the diskette into the subdirectory.
   copy a:*.* (or **copy b:*.***)

5. Start the session by keying in:
   frames

3. USING THE PRACTICE ENVIRONMENT

A recommended walkthrough of FRAMES follows. It suggests a path through the prototype which exposes the user to representative functionality of the environment. Since the entire walkthrough may take up to two hours, you may prefer doing it in stages.

3.1 The FRAMES demo

Work through the introductory screens as far as the screen, "**A VIEW of what YOU can DO with FRAMES**", shown in Screen Display A.1. At this screen , click on ">" to continue with introductory screens.

At the demo screen "**What YOU can DO with FRAMES**", click on any of the **R A M** options to read their elaborations. Then click on the **demo** request for a learn-by-doing demonstration.

From the screen "**Demo of an exercise**", please access all three modes in order, and read all the information on the screen. Note the instructional strategies of the three modes:

Mode 1: Read-only

Mode 2: Guided practice (filling in blanks)

Mode 3: D.I.Y. (Do-it-yourself)

In the **Mode 2** demo, click on **Help\Definitions** and elaborate **Symmetry**, so that you can answer the first question! Remember the double click or double <Enter> after using **Help**. In completing the proof, import some of the necessary characters by mouse-clicks on the symbol pad. Feel free to use **Help\Math** as well.

3.2 The real thing - Mode 1 (Read-Only)

When finished with demos, go to **Menu**. Use the main **Menu** for your first selection. (After this a mini-menu appears in right-hand control area. Continue by using the mini-menu, except when the screen must be cleared, in which case, return to main-menu.)

Compose each *frame* by clicking on **R A M control**, followed each time by **<GO>** to activate the selection.

A suggested list follows:

Relation	Attribute	**Mode**	**Comments**
P	Graphic aid	None	No mode applicable
P	Eg	None	No mode applicable. You don't need to re-click on a non-change aspect, such as **P**.
P	Prop\Ref	1	
P	Prop\As1	1	
P	Prop\Tra	1	Use **Help** if required.
Q	Graph	None	Note that *definition blackboard* changes.
Q	Prop\Tra	1	Same *property*, different *relations*
Browse through the relation definitions, clicking on **P**, **Q**, **S**, **T**, **V**, **W**, **TR** and note the instant appearance of each "blackboard". This is the only function activated immediately, without a **<GO>** click; it permits inspection of the relations prior to deciding on component selection.			

3.3 Mode 2 - Guided Practice

Return to main **Menu** to clear screen. Select:

Relation	**Attribute**	**Mode**	**Comments**
Q	Graphic aid	None	No mode applicable
Q	Prop\Irr	None	Follow the resulting prompt and choose any mode. Note the counter-example proof style for a property test that is negative.
Q	Prop\Tri	1	
Q	Prop\Sym	1	
Q	Prop\Sym	2	Same *property*, different *mode*. Use Mode 1 above as a guide. Remember to enter your *goal* in the **R.T.P.** area. Use **Help\Def** to access the definition of symmetry, remembering that the "Def of symmetry" required on the right is the general case for **R** on **X**. See Screen Display A.2 for comments on the definition blocks.
Q	Prop\Tri	2	Hit any **key/s** then **Enter Enter** to move fast. If you decide not to complete a transaction, click on menu, then re-select.
Q	Eg	None	Try some more *Example synthesis* exercises, and view more *graphic aid* components.
S	Graph	None	
S	Eg	None	
T	Graph	None	
T	Eg	None	
V	Graph	None	See Screen Display A.3
V	Eg	None	
W	Graph	None	
W	Eg	None	

3.3 Mode 3 - Do-it-yourself

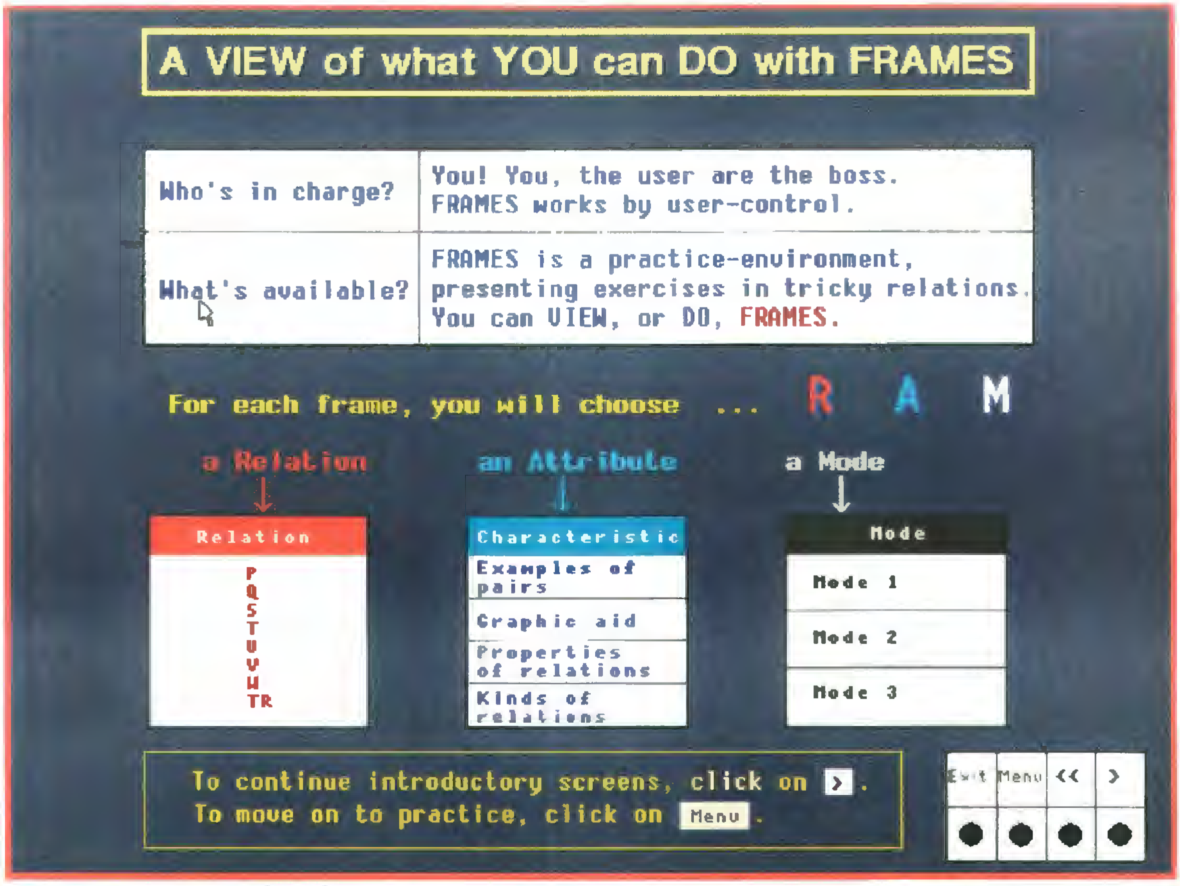
Now for some Mode 3 instructional transactions:

Relation	Attribute	Mode	**Comments**
P	Graph	None	
P	Prop\Sym	3	Hit any **key/s + Enter Enter** to move fast.
P	Prop\Ref	3	
P	Prop\Tra	3	Note pleasing appearance of a full screen, and refer to Screen Display A.4.
P	Prop\Tri	3	

Finally, some **Kind** attributes:

Relation	Attribute	Mode	**Comments**
Q	Kind\Eq	1	See the **Prop** tests presented automatically.
Q	Kind\Eq	3	Same *kind*, different *mode*
Q	Kind\WTO	3	After **reflexivity** test, change to mode 2, to use different modes within the same analysis. The antisymmetry test was negative, so the proof ends.

If ready to quit, click on **Exit**.



A VIEW of what YOU can DO with FRAMES

Who's in charge?	You! You, the user are the boss. FRAMES works by user-control.
What's available?	FRAMES is a practice-environment, presenting exercises in tricky relations. You can VIEW, or DO, FRAMES.

For each frame, you will choose ... R A M

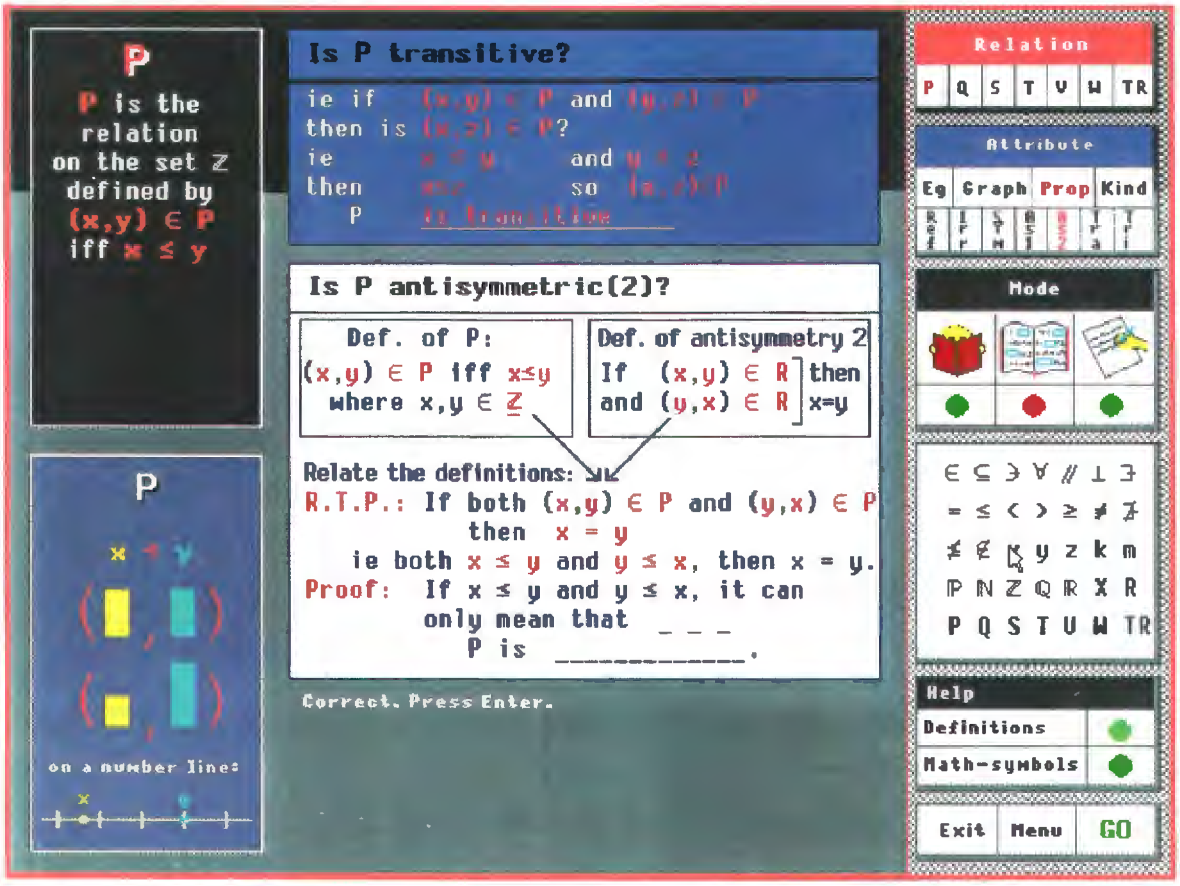
a Relation an Attribute a Mode

To continue introductory screens, click on >.
To move on to practice, click on Menu.

Screen Display A.1

The "VIEW of what YOU can DO with FRAMES" screen briefly introduces:

- ♦ FRAMES' purpose
- ♦ RAM control.



P

P is the relation on the set ℤ defined by (x,y) ∈ P iff x ≤ y

Is P transitive?

```
ie if     (x,y) ∈ P and (y,z) ∈ P
then is (x,z) ∈ P?
ie        x ≤ y      and y ≤ z
then      x≤z        so  (x,z)∈P
   P      is transitive
```

Is P antisymmetric(2)?

Def. of P:
(x,y) ∈ P iff x≤y
where x,y ∈ ℤ

Def. of antisymmetry 2
If (x,y) ∈ R and (y,x) ∈ R then x=y

Relate the definitions:

```
R.T.P.: If both (x,y) ∈ P and (y,x) ∈ P
          then   x = y
     ie both x ≤ y and y ≤ x, then x = y.
Proof:    If x ≤ y and y ≤ x, it can
          only mean that  _ _ _
             P is  _____________.
```

Correct. Press Enter.

Screen Display A.2

These *frames* illustrate problem-solving with the graphic aid on view.

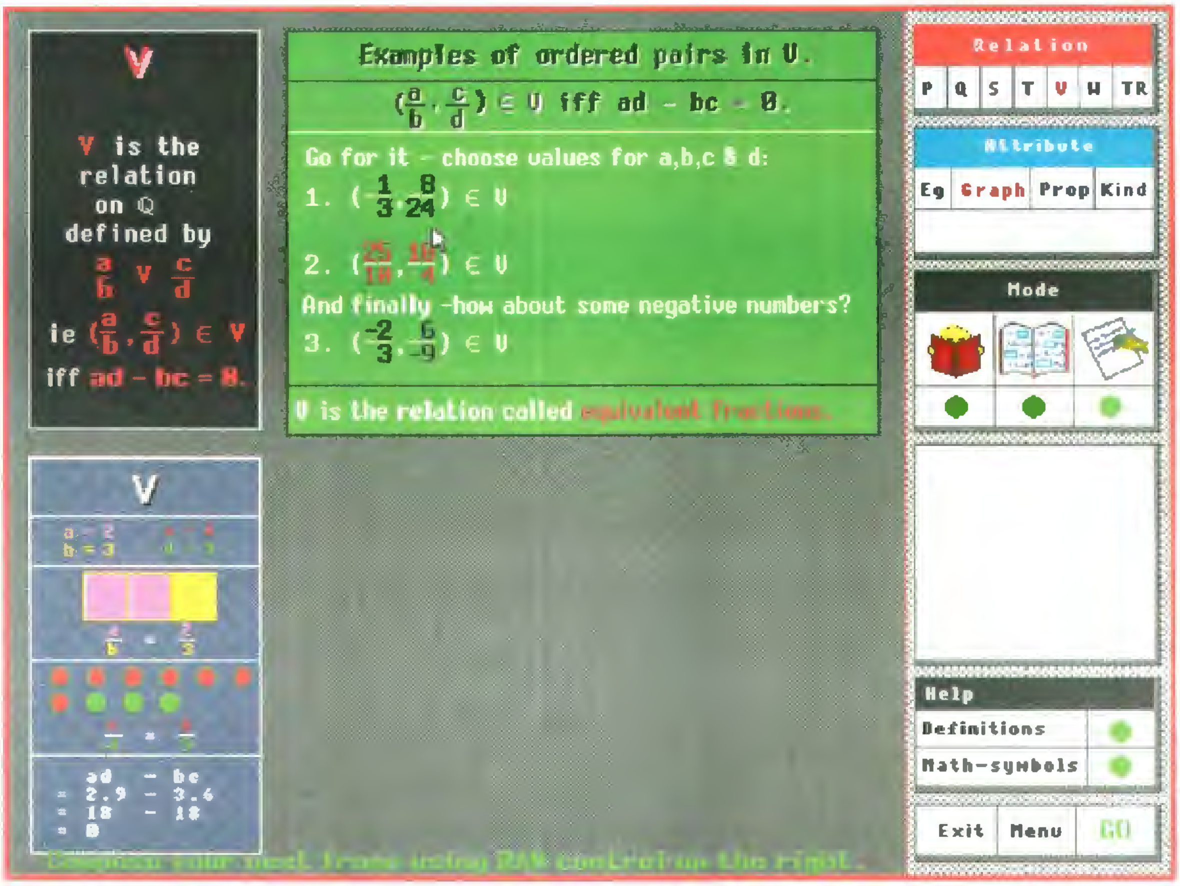
P Tra Mode 3 Note that the user's close spacing in line 4 is as acceptable to FRAMES as the more spread-out entries above.

P As2 Mode 2 This shows use of As2, the alternative definition of antisymmetry (which may be preferred by some users), instead of As1 as in Screen Displays 9.8 and 9.15.

It can be seen how the two **Definition** blocks, the relation on the left and the property on the right, are related to contextualize the general definition for a particular relation.
The user can follow the "blackboard" at top left to fill in the **P** definition, and **Help** can be used to fill in the definition of antisymmetry 2 (see Screen Display 9.17). The student then deduces the content of the **R.T.P.**, i.e. the subgoal, by integrating the two definitions.

The *guided practice* of Mode 2 is *faded* in Mode 3.

On a correct response, FRAMES requests the user to "Press Enter" to continue.



V

V is the relation on $\mathbb{Q}$ defined by $\frac{a}{b}$ V $\frac{c}{d}$ ie $(\frac{a}{b},\frac{c}{d}) \in$ V iff ad − bc = 0.

Examples of ordered pairs in V.

$(\frac{a}{b},\frac{c}{d}) \in V$ iff ad − bc = 0.

Go for it − choose values for a,b,c & d:

1. $(\frac{1}{3},\frac{8}{24}) \in V$

2. $(\frac{25}{10},\frac{10}{4}) \in V$

And finally −how about some negative numbers?

3. $(\frac{-2}{3},\frac{6}{-9}) \in V$

V is the relation called equivalent fractions.

Compose your next frame using RAM control on the right.

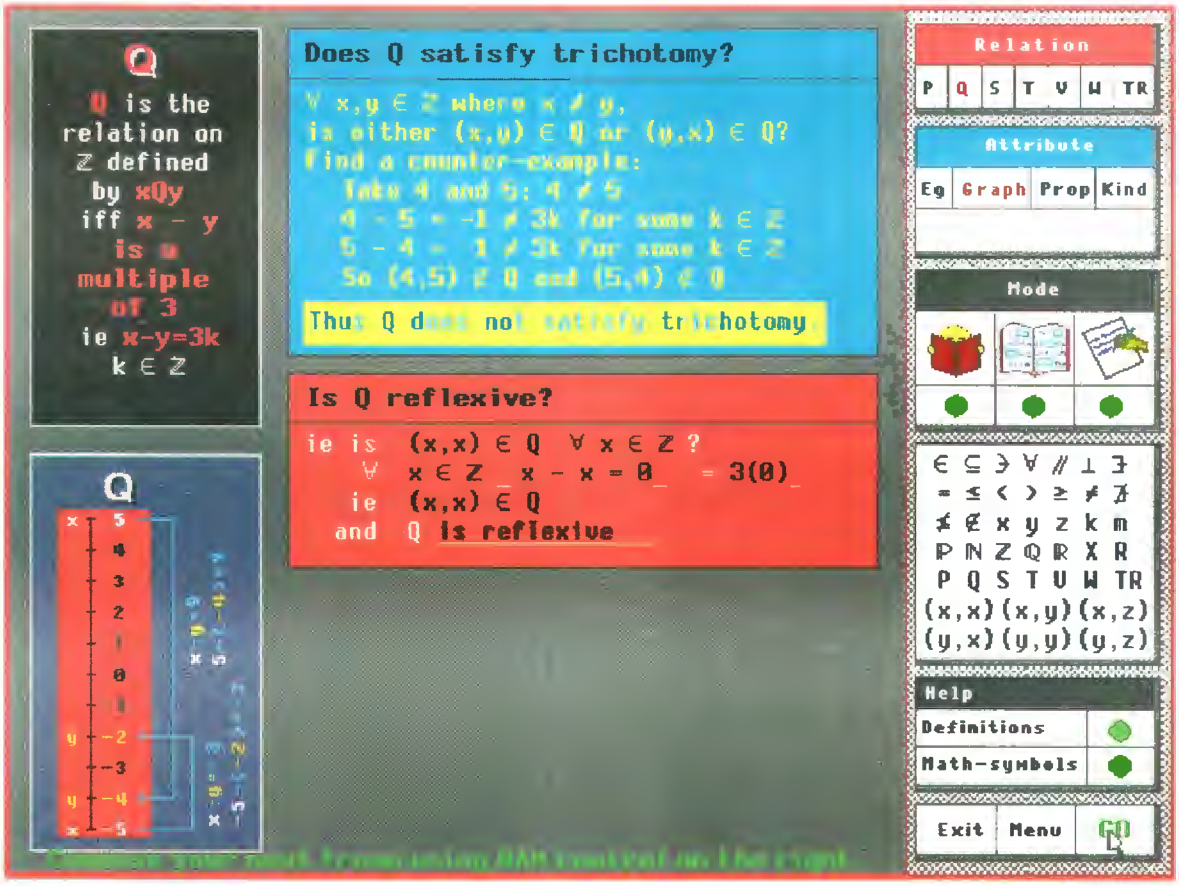
Screen Display A.3

This screen shows the selection **V** **Eg** at a completed stage.

The user got the second example wrong on both attempts and was presented with feedback in the form of a calculation (not shown) to explain why his response was unacceptable, followed by a correct version in red.

At the end of an example frame, a concluding sentence provides additional information about the relation to *anchor* it in its *real-world* meaning.

(The colour of feedback has subsequently been changed to white).



Q

Q is the relation on $\mathbb{Z}$ defined by xQy iff $x - y$ is a multiple of 3 ie $x-y=3k$ $k \in \mathbb{Z}$

Does Q satisfy trichotomy?

$\forall\ x,y \in \mathbb{Z}$ where $x \neq y$,
is either $(x,y) \in Q$ or $(y,x) \in Q$?
Find a counter-example:
Take 4 and 5: $4 \neq 5$
$4 - 5 = -1 \neq 3k$ for some $k \in \mathbb{Z}$
$5 - 4 = 1 \neq 3k$ for some $k \in \mathbb{Z}$
So $(4,5) \notin Q$ and $(5,4) \notin Q$

Thus Q does not satisfy trichotomy

Is Q reflexive?

ie is $(x,x) \in Q\ \ \forall\ x \in \mathbb{Z}$?
$\forall\ x \in \mathbb{Z}\ \ x - x = 0 = 3(0)$
ie $(x,x) \in Q$
and Q is reflexive

Compose your next frame using RAM control on the right

Screen Display A.4

The user selected:

Q Tri Mode 1 and **Q Ref Mode 3.**

Note the:

- aesthetic balance of a full screen layout
- symbol pad (*soft keyboard*) of Mode 3; compare this extended symbol pad with that of Mode 2 shown in Screen Display A.2.

Two different proof structures, a counter example and a general proof, are shown. They are appropriate for tests which turn out negative and positive respectively.

On conclusion, the user is prompted to select his next frame.

BIBLIOGRAPHY

[Alessi 1991] Alessi, S.M. & Trollip, S.R. (1991). *Computer-Based Instruction: Methods and Development.* Englewood Cliffs, N.J.: Prentice Hall.

[Anderson 1983] Anderson, J.R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.

[Anderson 1987] Anderson, J.R., Boyle, C.F., Farrell, R. & Reiser, B.J. (1987). Cognitive Principles in the Design of Computer Tutors. In: Morris, P. (Ed.) *Modelling Cognition.* Chichester: John Wiley & Sons.

[Anderson 1990] Anderson, J.R., Boyle, C.F., Corbett, A.T. & Lewis, M.W. (1990). Cognitive Modelling and Intelligent Tutoring. *Artificial Intelligence 42*, 7-49

[Anderson 1992] Anderson, J.R., Corbett, A.T., Fincham, J.M., Hoffman, D. & Pelletier, R. (1992). General Principles for an Intelligent Tutoring Architecture. In: Regian, J.W. & Shute, V.J. (Eds), *Cognitive Approaches to Automated Instruction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Appel 1987] Appel, M. & Appel, S. (1987). A critique of CAI: the case of SERGO. *South African Journal of Education 7* (4), 278-282.

[Aronson 1983] Aronson, D.T. & Briggs, L.J. (1983). Contribution of Gagné and Briggs to a Prescriptive Model of Instruction. In: Reigeluth, C.M. (Ed.), *Instructional-Design Theories and Models: An Overview of their Current Status.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Atkins 1991] Atkins, M.C. & Brown, A.W. (1991). Principles of Object-Oriented Systems. In: McDermid, J.A. (Ed.), *Software Engineer's Reference Book.* Oxford: Butterworth-Heineman.

[Barrow 1989a] Barrow, J. (1989). Hypertext as an Educational Medium. *Proceedings of the First Southern African Conference on Educational Technology.* Human Sciences Research Council, Pretoria.

[Barrow 1989b] Barrow, J. (1989). Structuring Hypertext. *Proceedings of the Fifth South African Computer Symposium.* Johannesburg.

[Bell 1992] Bell, D., Morrey, I., & Pugh, J. (1992). *Software Engineering: A Programming Approach* (2nd ed.). Hemel Hempstead: Prentice Hall International (UK) Ltd.

[Bevan 1991] Bevan, N., Kirakovski, J. & Maissel, J. (1991). What is Usability? In: Bullinger, H.J. (Ed), *Human Aspects in Computing: Design and Use of Interactive Systems and Work with Terminals.* Amsterdam: Elsevier Science Publishers B.V..

[Black 1987] Black, T. R. (1987). CAL Delivery Selection Criteria and Authoring Systems. *Journal of Computer-Assisted Learning 3*, 204-213.

[Black 1988] Black, T.R. (1988). Prototyping CAL Courseware: A Role for Computer-Shy Subject Experts. In: Mathias, H., Rushby, N. & Budgett, R. (Eds), *Aspects of Educational Technology, Vol XXI, Designing New Systems and Technologies for Learning.* London: Kogan Page.

[Black 1989] Black, T.R. & Hinton, T. (1989). Courseware Design Methodology: the Message from Software Engineering. In: Bell, C., Davies, J. & Winders, R. (Eds), *Aspects of Educational and Training Technology, Vol XXII, Promoting Learning.* London: Kogan Page.

[Boder 1990] Boder, A. & Cavallo, D. (1990). An Epistemological Approach to Intelligent Tutoring Systems. *Intelligent Tutoring Media 1* (1), 23-29.

[Bodker 1991] Bodker, S. (1991). *Through the Interface: A Human Activity Approach to User Interface Design.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Booch 1991] Booch, G. (1991). *Object-Oriented Design: with Applications.* Redwood City, CA: Benjamin/Cummings Publishing Company, Inc.

[Bowers 1989] Bowers, D. (1989). The Software Design Document: More than a User's Manual. *Educational Technology 29* (12), 15-18.

[Bransford 1990] Bransford, J.D., Sherwood, R.D., Hasselbring, T. S., Kinzer, C.K. & Williams, S.M. (1990). Anchored Instruction: Why We Need It and How Technology Can Help. In: Nix, D. & Spiro, R. (Eds), *Cognition, Education, Multimedia: Exploring Ideas on High Technology.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Briggs 1981] Briggs, L.J. & Wager, W.W. (1981). *Handbook of Procedures for the Design of Instruction.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Briggs 1991] Briggs, L.J., Gustafson, K.L. & Tillman, M.H. (Eds) (1991). *Instructional Design Principles and Applications.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Brown 1989] Brown, J.R. & Cunningham, S. (1989). *Programming the User Interface: Principles and Examples.* New York: John Wiley and Sons, Inc.

[Budgen 1994] Budgen, D. (1994). *Software Design.* Wokingham: Addison-Wesley.

[Carbonell 1970] Carbonell, J.R. (1970). AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction. *IEEE Transactions on Man-Machine Systems 11* (4), 190-202.

[Carrier 1988] Carrier, C.A. & Jonassen, D.H. (1988). Adapting Courseware to Accommodate Individual Differences. In: Jonassen, D.H. (Ed.), *Instructional Designs for Microcomputer Courseware.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Chapanis 1991] Chapanis, A. (1991). Evaluating Usability. In: Shackel, B. & Richardson, S.J. (Eds), *Human Factors for Informatics Usability.* Cambridge: Cambridge University Press.

[Chen 1976] Chen, P.P. (1976). The Entity-Relationship Model: Towards a Unified View of Data. *ACM Trans. Database Systems 1* (1), 9-36.

[Chen 1989] Chen, J.W. & Shen, C. (1989). Software Engineering: A New Component for Instructional Software Development. *Educational Technology 29* (9), 9-15.

[Chen 1990] Chen, J.W. & Chen, M. (1990). Towards the Design of an Intelligent Courseware Production System using Software Engineering and Instructional Design Principles. *Journal of Educational Technology Systems 19* (1), 41-52.

[Christensen 1990] Christensen, L.C. & Bodey, M.R. (1990). A Structure for Creating Quality Courseware. *Collegiate Microcomputer 8* (3), 201-209.

[Clancey 1981] Clancey, W.J. & Letsinger, R. (1981). Neomycin: Reconfiguring a Rule-Based Expert System for Application to Teaching. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Volume II.* Los Altos, CA: William Kaufman Inc.

[Clancey 1982] Clancey, W.J. (1982). Tutoring Rules for Guiding a Case Method Dialogue. In: Sleeman, D. & Brown, J.S. (Eds), *Intelligent Tutoring Systems.* London: Academic Press.

[Clancey 1987] Clancey, W.J. (1987). Methodology for Building an Intelligent Tutoring System. In: Kearsley, G. (Ed.), *Artificial Intelligence and Instruction: Applications and Methods.* Reading, MA: Addison-Wesley.

[Clancey 1990] Clancey, W.J. & Soloway, E. (Eds) (1990). *Artificial Intelligence and Learning Environments.* Cambridge, MA: MIT Press.

[Coad 1990] Coad, P. & Yourdon, E. (1990). *Object-Oriented Analysis.* Englewood Cliffs, N.J.: Prentice Hall.

[Collins 1983] Collins, A. & Stevens, A.L. (1983). A Cognitive Theory of Inquiry Teaching. In: Reigeluth, C.M. (Ed.), *Instructional Design Theories and Models: An Overview of their Current Status.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Collis 1987] Collis, B. & Gore, M. (1987). Combining Software Engineering and Instructional Design in a New Type of Course for Educators. *Journal of Research on Computing in Education 20* (2), 104-116.

[Conger 1994] Conger, S.A. (1994). *The New Software Engineering.* Belmont, CA: Wadsworth Publishing Company.

[Corbett 1991] Corbett, A.T. & Anderson, J.R. (1991). LISP Intelligent Tutoring System: Research in Skill Acquisition. In: Larkin, J.H. & Chabay R.W. (Eds), *Computer Assisted Instruction and Intelligent Tutoring Systems.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Corbett 1993] Corbett, A.T., Anderson, J.R. & O'Brien, A.T. (1993). The Predictive Validity of Student Modelling in the ACT Programming Tutor. *Proceedings of AI-ED 93, World Conference on Artificial Intelligence in Education.* Edinburgh: Association for the Advancement of Computing in Education.

[Cumming 1990] Cumming, G. (1990). Artificial Intelligence and Images of Natural Learning. In: McDougall, A. & Dowling, C. (Eds), *Computers in Education.* Amsterdam: Elsevier Science Publishers B.V. (North Holland).

[Cumming 1991a] Cumming, G. (1991). Using Artificial Intelligence to Achieve Natural Learning. In: Lewis, R. & Otsuki, S. (Eds), *Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education.* Amsterdam: North-Holland.

[Cumming 1991b] Cumming, G. & Self, J. (1991). Learner Modelling in Collaborative Intelligent Educational Systems. In: Goodyear, P. (Ed.), *Teaching Knowledge and Intelligent Tutoring.* Norwood, N.J.: Ablex.

[De Villiers 1989a] de Villiers, C. (1989). *'n Rekenaargebaseerde onderrigstrategie vir Rekenaarwetenskap met besondere verwysing na afstandonderrig.* Unpublished MEd dissertation, University of South Africa, Pretoria.

[De Villiers 1989b] de Villiers, M.R. (1989). *Structured Knowledge Representation with Particular Reference to Frames.* Unpublished Special Topic Report, Department of Computer Science, University of South Africa, Pretoria.

[De Villiers 1992] de Villiers, C., Pistorius, M.C., Alexander, P.M. & du Plooy, N.F. (1992). Constraints on Computer-Assisted Instruction in a Distance Education Environment. *Computing Control Engineering Journal 3* (1), 11-13.

[De Villiers 1993] de Villiers, M.R. (1993). *Relations: A CAI Tutorial in Theoretical Computer Science.* Unpublished MEd mini-dissertation, University of Pretoria, Pretoria.

[De Wet 1994] De Wet, L. (1994). *A Comparison of the Usability Properties of Character-based and Graphical-based User Interfaces.* Unpublished MSc thesis, University of South Africa, Pretoria.

[Dick 1991] Dick, W. (1991). An Instructional Designer's View of Construction. *Educational Technology 31* (5), 41-44.

[Dijkstra 1990] Dijkstra, S., van Hout Wolters, B.H.A.M. & van der Sijde, P.C. (Eds) (1990). *Research on Instruction: Design and Effects.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Dix 1993]	Dix, A., Finlay, J., Abowd, G. & Beale, R. (1993). *Human-Computer Interaction.* Hemel Hempstead: London: Prentice Hall International (UK).
[Fischetti 1990]	Fischetti, E. & Gisolfi, A. (1990). From Computer-Aided Instruction to Intelligent Tutoring Systems. *Educational Technology 30* (8), 7-17.
[Fleming 1978]	Fleming, M.L. & Levie, W.H. (1978). *Instructional Message Design: Principles from the Behavioural Sciences.* Englewood Cliffs, N.J.: Educational Technology Publications.
[Ford 1988]	Ford, L. (1988). The Appraisal of an ICAI System. In: Self, J.A. (Ed), *Artificial Intelligence and Human Learning.* London: Chapman and Hall.
[Forman 1988]	Forman, G. & Pufal, P.B. (Eds) (1988). *Constructivism in the Computer Age.* Hillsdale, N.J.: Lawrence Erlbaum Associates.
[Gagné 1985]	Gagné, R.M. (1985). *The Conditions of Learning.* New York: Holt, Rinehart and Winston.
[Gagné 1987]	Gagné, R.M. & Glaser, R. (1987). Foundations in Learning Research. In: Gagné, R.M. (Ed.), *Instructional Technology: Foundations.* Hillsdale, N.J.: Lawrence Erlbaum Associates.
[Gagné 1991]	Gagné, R.M., Wager, W. & Rojas, A. (1991). Planning and Authoring Computer-Assisted Instruction Lessons. In: Briggs, L.J., Gustafson, K.L. & Tillman, M.H. (Eds), *Instructional Design Principles and Applications.* Englewood Cliffs, N.J.: Educational Technology Publications.
[Gery 1987]	Gery, G. (1987). *Making CBT Happen.* Boston: Weingarten.
[Goodyear 1991]	Goodyear, P. (Ed.) (1991). *Teaching Knowledge and Intelligent Tutoring.* Norwood, N.J.: Ablex.
[Görner 1992]	Görner, C., Vossen, P. & Ziegler, J. (1992). Direct Manipulation User Interface. In: Galer, M., Harker, S. & Ziegler, J. (Eds), *Methods and Tools in User-Centred Design for Information Technology.* Amsterdam: Elsevier Science Publishers B.V.
[Gray 1994]	Gray, D.E. & Black, T.R. (1994). Prototyping of Computer-Based Training Materials. *Computers in Education 22* (3), 251-256.
[Gropper 1983]	Gropper, G.L. (1983). A Behavioral Approach to Instructional Prescription. In: Reigeluth, C.M. (Ed.) *Instructional Design Theories and Models: An Overview of their Current Status.* Hillsdale, N.J.: Lawrence Erlbaum Associates.
[Gruber 1977]	Gruber, H.E. & Voneche, J.J. (1977). *The Essential Piaget.* New York: Basic Books, Inc. Publishers.
[Hammond 1989]	Hammond, N.V. & Allinson, L.J. (1989). Extending Hypertext for Learning: An Investigation of Access and Guidance Tools. In: Sutciffe, A. & Macauly, L. (Eds), *People and Computers V.* Cambridge: Cambridge University Press.
[Hammond 1992a]	Hammond, N.V. (1992). Learning with Hypertext: Problems, Principles and Prospects. In: McKnight, C., Dillon, A. & Richardson, J. (Eds), *Hypertext: A Psychological Perspective.* Chichester: Ellis Horwood.
[Hammond 1992b]	Hammond, N.V. (1992). Tailoring Hypertext for the Learner. In: Kommers, P.A.M., Jonassen, D.H. & Mayes, J.T. (Eds), *Cognitive Tools for Learning.* Berlin: Springer-Verlag.
[Hannafin 1988]	Hannafin, M.J. & Peck, K.L. (1988). *The Design, Development, and Evaluation of Instructional Software.* New York: MacMillan Publishing Company.

[Hardman 1988] Hardman, L. (1988). Hypertext Tips: Experiences in Developing a Hypertext Tutorial. In: Jones, D.M. & Winder, R. (Eds), *People and Computers IV.* Cambridge University Press.

[Hasselerharm 1990] Hasselerharm, E. & Leemkuil, H. (1990). The Relation between Instructional Control Strategies and Performance and Attitudes in Computer-Based Instruction. In: Pieters, J.M., Simons, P.R.J. & de Leeuw, L. (Eds), *Research on Computer-Based Instruction.* Amsterdam: Swets and Zeitlinger B.V.

[Henderson-Sellers 1990] Henderson-Sellers, B. & Edwards, J.M. (1990). The Object-Oriented Systems Life Cycle. *Communications of the ACM 33* (9), 142-159.

[Ibrahim 1989] Ibrahim, B. (1989). Software Engineering Techniques for Computer-Aided Learning. *Education and Computing 5* (4), 215-222.

[Ince 1991] Ince, D. (1991). Prototyping. In: McDermid, J.A. (Ed.), *Software Engineer's Reference Book.* Oxford: Butterworth-Heineman.

[Inhelder 1958] Inhelder, B. & Piaget, J. (1958). *The Growth of Logical Thinking from Childhood to Adolescence.* New York: Basic Books Inc. Publishers.

[Jonassen 1988] Jonassen, D.H. (1988). Integrating Learning Strategies into Courseware to Facilitate Deeper Processing. In: Jonassen, D.H. (Ed.), *Instructional Designs for Microcomputer Courseware.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Jonassen 1989] Jonassen, D.H. (1989). *Hypertext / Hypermedia.* Englewood Cliffs, N.J.: Educational Technology Publications.

[Jonassen 1992] Jonassen, D.H. (1992). Effects of Semantically Structured Hypertext Knowledge Bases on Users' Knowledge Structures. In: McKnight, C., Dillon, A. & Richardson, J. (Eds), *Hypertext: a Psychological Perspective.* Chichester: Ellis Horwood.

[Kearsley 1987] Kearsley, G. (Ed.) (1987). *Artificial Intelligence and Instruction: Applications and Methods.* Reading, MA: Addison-Wesley.

[Keller 1987] Keller, A. (1987). *When Machines Teach: Designing Computer Courseware.* New York: Harper and Row.

[Keller 1988] Keller, J.M. & Suzuki, K. (1988). Use of the ARCS Motivational model in Courseware Design. In: Jonassen, D.H. (Ed.), *Instructional Designs for Microcomputer Courseware.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Kok 1990] Kok, W. & Poorthuis, G. (1990). The Effects of Different Teaching Strategies in Three CAI Programs within the Same Content Area. In: Pieters, J.M., Simons, P.R.J. & de Leeuw, L. (Eds), *Research on Computer-Based Instruction.* Amsterdam: Swets and Leitzinger B.V.

[Kontos 1985] Kontos, G. (1985). Instructional Computing: In Search of Better Methods for the Production of CAI Lessons. *Computer Education 49,* 16-19.

[Korson 1990] Korson, T. & Mc Gregor, J.D. (1990). Understanding Object-Oriented: A Unifying Paradigm. *Communications of the ACM 33 (9),* 40-60.

[Kotzé 1995] Kotzé, P. (1995). *An Option Space for the Authoring of Interactive Tutoring Systems.* Unpublished DPhil thesis proposal, Department of Computer Science, University of York, United Kingdom.

[Labuschagne 1993] Labuschagne, W. (1993). *A User-friendly Introduction to Discrete Mathematics for Computer Science.* University of South Africa, Pretoria.

[Lantz ??] Lantz, K.E. (no publication date - possibly 1985). *The Prototyping Methodology.* Englewood Cliffs, N.J.: Prentice-Hall.

[Lanzing 1994] Lanzing, J.W.A. & Stanchev, I. (1994). Visual aspects of Courseware Engineering. *Journal of Computer Assisted Learning 10*, 69-80.

[Laridon 1989] Laridon, P.E. (1989). Design Features in Interactive Video Mathematics Lessons. *Proceedings of the First Southern African Conference on Educational Technology*, Human Sciences Research Council, Pretoria.

[Larkin 1991] Larkin, J.H. & Chabay, R.W. (Eds) (1991). *Computer Assisted Instruction and Intelligent Tutoring Systems*. Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Lavoie 1991] Lavoie, M., Gagné, M. & Jacques, A. (1991). Specifications of a Software System which assists in the Design and Construction of Knowledge-Based Instructional Software. In: Lewis, R. & Otsuki, S. (Eds), *Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education*. Amsterdam: North-Holland.

[Lesgold 1992] Lesgold, A., Eggan, G., Katz, S., & Rao, G. (1992). Possibilities for Assessment using Computer-Based Apprenticeship Environments. In: Regian, J.W. & Shute, V.J. (Eds). *Cognitive Approaches to Automated Instruction*. Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Lewis 1991] Lewis, R. & Otsuki, S. (Eds) (1991). *Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education*. Amsterdam: North-Holland.

[Lippert 1989] Lippert, R.C. (1989). Expert Systems: Tutors, Tools, and Tutees. *Journal of Computer-Based Instruction 16* (1), 11-19.

[Lippert 1990] Lippert, R.C. (1990). *Wat maak RGO Intelligent?* Paper at the INSTRUCTA 90 Seminar, Rand Afrikaans University, Johannesburg, South Africa.

[Lippert 1993] Lippert, R.C. (Ed.) (1993). *Computer-Based Education and Training in South Africa*. Pretoria: J.L. van Schaik Publishers.

[McKnight 1992] McKnight, C., Dillon, A. & Richardson, J. (Eds) (1992). *Hypertext: a Psychological Perspective*. Chichester: Ellis Horwood.

[McLean 1989] McLean, R.S. (1989). Megatrends in Computing and Educational Software Development. *Education and Computing 5*, 55-60.

[Mehl 1993] Mehl, M.C. & Sinclair, A.J.L. (1993). Defining a Context for CAI: In Quest of Educational Reality. In: Lippert, R.C. (Ed.), *Computer-Based Education and Training in South Africa*. Pretoria: J.L. van Schaik Publishers.

[Merrill 1983] Merrill, M.D. (1983). Component Display Theory. In: Reigeluth, C.M. (Ed.), *Instructional Design Theories and Models: An Overview of their Current Status*. Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Merrill 1987] Merrill, M.D. (1987). The New Component Design Theory: Instructional Design for Courseware Authoring. *Instructional Science 16*, 19-34.

[Merrill 1988] Merrill, M.D. (1988). Applying Component Display Theory to the Design of Courseware. In: Jonassen, D.H. (Ed.), *Instructional Designs for Microcomputer Courseware*. Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Merrill 1990a] Merrill, M.D. & Li, Z. (1990). An Instructional Design Expert System. In: Dijkstra, S., van Hout Wolters, B.H.A.M. & van der Sijde, P.C. (Eds), *Research on Instruction: Design and Effects*. Englewood Cliffs, N.J.: Educational Technology Publications.

[Merrill 1990b] Merrill, M.D., Li, Z. & Jones, M.K. (1990). Limitations of First Generation Instructional Design. *Educational Technology 30* (1), 7-11.

[Merrill 1991] Merrill, M.D. (1991). Constructivism and Instructional Design. *Educational Technology 31*(5), 45-52.

[Midoro 1991] Midoro, V., Olimpo, G., Persico, D. & Sarti, L. (1991). Multimedia Navigable Systems and Artificial Intelligence. In: Lewis, R. & Otsuki, S. (Eds), *Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education.* Amsterdam: North-Holland.

[Minsky 1975] Minsky, M.L. (1975). A Framework for Representing Knowledge. In: Winston, P.H. (Ed.), *The Psychology of Computer Vision.* New York: McGraw-Hill.

[Minsky 1985] Minsky, M.L. (1985). *The Society of Mind.* New York: Simon and Schuster.

[Morris 1987] Morris, P. (Ed) (1987). *Modelling Cognition.* Chichester: John Wiley & Sons.

[Morrison 1988] Morrison, G.R. & Ross, S.M. (1988). A Four-Stage Model for Planning Computer-Based Instruction. *Journal of Instructional Development 11* (1) 6-14.

[Naisbitt 1982] Naisbitt, J. (1982). *Megatrends.* New York: Warner Books.

[Newell 1972] Newell, A. & Simon, H.A. (1972). *Human Problem Solving.* Englewod Cliffs, N.J.: Prentice-Hall Inc.

[Nicholson 1988} Nicholson, R. (1988). SCALD - Towards an Intelligent Authoring System. In: Self, J.A. (Ed.), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* London: Chapman & Hall.

[Nielsen 1990] Nielsen, J. (1990). *Hypertext and Hypermedia.* Boston: Academic Press.

[Nix 1990] Nix, D. (1990). Should Computers Know what you can do with Them? In: Nix, D. & Spiro, R. (Eds), *Cognition, Education, Multimedia: Exploring Ideas in High Technology.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[O'Shea 1983] O'Shea, T. & Self, J. (1983). *Learning and Teaching with Computers.* Brighton: The Harvester Press.

[Papert 1980] Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas.* New York: Basic Books.

[Papert 1988] Papert, S. (1988). The Conservation of Piaget: The Computer as Grist to the Constructivist Mill. In: Forman, G. & Pufal, P.B. (Eds), *Constructivism in the Computer Age.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Pieters 1990] Pieters, J.M., Simons, P.R.J. & de Leeuw, L. (Eds) (1990). *Research on Computer-Based Instruction.* Amsterdam: Swets and Zeitlinger B.V.

[Pistorius 1992] Pistorius, M.C., de Villiers, C. & Alexander, P.M. (1992). CAI - Alive and Well at Unisa. *CBE in Tertiary Education, Proceedings of the Third CBE/CBT Conference.* University of South Africa, Pretoria.

[Poppen 1988] Poppen, L. & Poppen, R. (1988). The Use of Behavioral Principles in Educational Software. *Educational Technology 28* (2), 37-41.

[Powers 1990] Powers, M.J., Cheney, P.H. & Crow, G.B. (1990). *Structured Systems Development.* Boston, MA: Boyd and Fraser Publishing Co.

[Preece 1993] Preece, J. (Ed.) (1993). *A Guide to Usability: Human Factors in Computing.* Wokingham: Addison Wesley.

[Price 1991] Price, R.V. (1991). *Computer-Aided Instruction: A Guide for Authors.* Pacific Grove, CA: Brooks/Cole.

[Pufall 1988] Pufall, P.B. (1988). Function in Piaget's System: Some Notes for Constructors of Microworlds. In: Forman, G. & Pufal, P.B. (Eds), *Constructivism in the Computer Age.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Quillian 1968] Quillian, M.R. (1968). Semantic Memory. In: Minsky, M. (Ed), *Semantic Information Processing.* Cambridge, MA: MIT Press.

[Ravden 1989] Ravden, S.J. & Johnson, G.I. (1989). *Evaluating Usability of Human-computer Interfaces: A Practical Method.* Chichester: Ellis Horwood.

[Regian 1992] Regian, J.W. & Shute V.J. (1992). Automated Instruction as an Approach to Individualization. In: Regian, J.W. & Shute V.J. (Eds), *Cognitive Approaches to Automated Instruction.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Reigeluth 1983] Reigeluth, C.M. (Ed.) (1983). *Instructional-Design Theories and Models: An Overview of their Current Status.* Hillsdale N.J.: Lawrence Erlbaum Associates.

[Ridgeway 1988] Ridgeway, J. (1988). Of course ICAI is Impossible... Worse though, it might be Seditious. In: Self, J.A. (Ed), *Artificial Intelligence and Human Learning.* London: Chapman and Hall.

[Roblyer 1988] Roblyer, M.D., Castine, W.H. & King, F.J. (1988). *Assessing the Impact of Computer-Based Instruction: A Review of Recent Research.* New York: Haworth Press.

[Sakasai 1990] Sakasai, Y. & Watanabe, T. (1990). A CAI System for Software Engineers: SOLAS. *Nippon Telegraph and Telephone Review 2* (2), 81-88.

[Schach 1990] Schach, S.R. (1990). *Software Engineering.* Boston, MA: Aksen Associates Inc. Publishers.

[Schiever 1991] Schiever, S.W. (1991). *A Comprehensive Approach to Teaching Thinking.* Boston: Allyn and Bacon.

[Self 1979] Self, J.A. (1979). Student Models and Artificial Intelligence. *Computers and Education 3,* 309-312.

[Self 1985] Self, J. (1985). *Microcomputers in Education: A Critical Evaluation of Educational Software.* Brighton: The Harvester Press.

[Self 1988] Self, J.A. (Ed.) (1988). *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* London: Chapman and Hall.

[Simon 1981] Simon, H.A. (1981). *The Sciences of the Artificial* (2nd ed.). Cambridge, MA: MIT Press.

[Siviter 1992] Siviter, D. & Brown, K. (1992). Hypercourseware. In: Kibby, M.R. & Hartley, J.R. (Eds). *Computer Assisted Learning: Selected Contributions from the CAL91 Symposium.* Oxford: Pergamon Press.

[Skinner 1938] Skinner, B.F. (1938). *The Behaviour of Organisms: An Experimental Analysis.* New York: Longman.

[Sleeman 1982] Sleeman, D. & Brown, J.S. (Eds) (1982). *Intelligent Tutoring Systems.* London: Academic Press.

[Smith 1984] Smith, P.L. & Boyce, B.A. (1984). Instructional Design Considerations in the Development of Computer-Assisted Instruction. *Educational Technology 24* (7), 5-11.

[Sommerville 1992] Sommerville, I. (1992). *Software Engineering* (4th ed). Wokingham: Addison-Wesley Publishing Company.

[Soulier 1988] Soulier, J.S. (1988). *The Design and Development of Computer Based Instruction.* Boston: Allyn and Bacon, Inc.

[Spiro 1990] Spiro, R.J. & Jehng, J. (1990). Cognitive Flexibility and Hypertext: Theory and Technology for the Nonlinear and Multidimensional Traversal of Complex Subject Matter. In: Nix, D. & Spiro, R.J. (Eds), *Cognition, Education and Multimedia: Exploring Ideas in High Technology.* Hillsdale, N.J.: Lawrence Erlbaum Associates.

[Tang 1991] Tang, H., Barden, R. & Clifton, C. (1991). A New Learning Environment Based on Hypertext and its Techniques. In: Lewis, R. & Otsuki, S. (Eds), *Advanced Research on Computers in Education, Proceedings of the IFIP TC3 International Conference on Advanced Research on Computers in Education.* Amsterdam: North-Holland.

[Thimbleby 1990] Thimbleby, H. (1990). *User Interface Design.* Wokingham: Addison Wesley.

[Thomas 1989] Thomas, T.A. (1989). Intelligent Tutoring Systems. *Proceedings of the First Southern African Conference on Educational Technology.* Human Sciences Research Council, Pretoria.

[Tripp 1990] Tripp, S.D. & Bichelmeyer, B. (1990). Rapid Prototyping: An Alternative Instructional Design Strategy. *Educational Technology, Research and Development 38* (1), 31-44.

[Venezky 1991] Venezky, R. & Osin, L. (1991). *The Intelligent Design of Computer-Assisted Instruction.* New York: Longman.

[Verhagen 1988] Verhagen, P.W. & Plomp, T. (1988). Educational Technology: A Dutch Contribution to the Debate. In: Mathias, H., Rushby, N. & Budgett, R. (Eds), *Aspects of Educational Technology, Vol XXI, Designing New Systems and Technologies for Learning.* London: Kogan Page.

[Visser 1995} Visser, M.S.P. (1995). *Rekenaarbenutting in die opleiding van inligtingsvaardige gebruikers aan 'n akademiese inligtingsdiens.* Uncompleted DPhil thesis, Department of Information Science, University of Pretoria, Pretoria.

[Vockell 1989] Vockell, E. & van Deusen, R.M. (1989). *The Computer and Higher-Order Thinking Skills.* Watsonville, CA: Mitchell Publishing, Inc.

[Webb 1989] Webb, G.I. (1989). Courseware Abstraction: Reducing Development Costs while Producing Qualitative Improvements in CAL. *Journal of Computer Assisted Learning 5* (2), 103-113.

[West 1991] West, C.K., Farmer, J.A. & Wolff, P.M. (1991). *Instructional Design: Implications from Cognitive Science.* Englewood Cliffs, N.J.: Prentice Hall.

[Wilson 1992] Wilson, B. & Cole, P. (1992). A Review of Cognitive Teaching Models. *Educational Technology Research and Development 39* (4), 47-64.

[Winograd 1987) Winograd, T. & Flores, F. (1987). *Understanding Computers and Cognition: A New Foundation for Design.* Reading, MA: Addison-Wesley.

[Winston 1975] Winston, P.H. (Ed.) (1975). *The Psychology of Computer Vision.* New York: McGraw-Hill.

[Wong 1993] Wong, S. C. (1993). Quick Prototyping of Educational Software: An Object-Oriented Approach. *Journal of Educational Technology Systems 22* (2), 155-172.

[Woodhead 1991] Woodhead, N. (1991). *Hypertext and Hypermedia: Theory and Applications.* Wokingham: Addison-Wesley.

[Woodroffe 1988] Woodroffe, M.R. (1988). Plan Recognition and Intelligent Tutoring Systems. In: Self, J.A. (Ed.), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* London: Chapman and Hall.

[Woolf 1988] Woolf, B.P. (1988). Representing Complex Knowledge in an Intelligent Machine Tutor. In: Self, J.A. (Ed), *Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction.* London: Chapman and Hall.