THE UNIVERSITY OF ASTON IN BIRMINGHAM

KNOWLEDGE-BASED SYSTEMS AND SOFTWARE ENGINEERING

JONATHAN LESLIE BADER

Submitted for the Degree
of
Doctor of Philosophy

MAY 1988

THE UNIVERSITY OF ASTON IN BIRMINGHAM

Knowledge-Based Systems and Software Engineering

Jonathan Leslie Bader

Submitted for the Degree of Doctor of Philosophy

1988

THESIS SUMMARY

The work described was carried out as part of a collaborative Alvey software engineering project (project number SE057). The project collaborators were the Inter-Disciplinary Higher Degrees Scheme of the University of Aston in Birmingham, BIS Applied Systems Ltd. (BIS) and the British Steel Corporation. The aim of the project was to investigate the potential application of knowledge-based systems (KBSs) to the design of commercial data processing (DP) systems.

The work was primarily concerned with BIS's Structured Systems Design (SSD) methodology for DP systems development and how users of this methodology could be supported using KBS tools. The problems encountered by users of SSD are discussed and potential forms of computer-based support for inexpert designers are identified. The architecture for a support environment for SSD is proposed based on the integration of KBS and non-KBS tools for individual design tasks within SSD - the *Intellipse* system. The *Intellipse* system has two modes of operation - Advisor and Designer. The design, implementation and user-evaluation of Advisor are discussed. The results of a Designer feasibility study, the aim of which was to analyse major design tasks in SSD to assess their suitability for KBS support, are reported. The potential role of KBS tools in the domain of database design is discussed.

The project involved extensive knowledge engineering sessions with expert DP systems designers. Some practical lessons in relation to KBS development are derived from this experience. The nature of the expertise possessed by expert designers is discussed.

The need for operational KBSs to be built to the same standards as other commercial and industrial software is identified. A comparison between current KBS and conventional DP systems development is made. On the basis of this analysis, a structured development method for KBSs is proposed - the POLITE model. Some initial results of applying this method to KBS development are discussed. Several areas for further research and development are identified.

Keywords: Knowledge-Based Systems, Expert Systems, Software Engineering, Data Processing, Structured Systems Design.

For my Father

Acknowledgements

Thanks are due first to my colleagues at BIS Applied Systems Limited with whom I have worked closely for three years: Raj Arya, Janet Clifford, David Hannaford and Chris Harris Jones. A special thank you to David Hannaford, the *Intellipse* project manager, with whom I have had many illuminating discussions. I am particularly grateful to David for finding the time to read through this thesis, and for suggesting several helpful amendments. Thanks are also due to the BIS consultants who acted as domain experts, and without whom much of the project work would have been impossible: Philip Alderman, Mike Capewell, Charles Haddon, Keith Hindle and Ann Inman. I should also like to thank Russel Thomas and Kevin Wilson of BIS Banking Systems Limited; Bernard Buckroyd of the Corporate Management Services Division of ICI plc.; and Tony Dignan of the Alvey Directorate. A special vote of thanks must go to my colleagues at the British Steel Corporation who "hung in there", even when it must have appeared to them that things were getting a little remote from their needs: Alan Griffiths, John Hornsby, Mike Humphreys, John Poston and Tony Spriggs.

I should like to acknowledge the fellowship of my contemporary IHD students: Clive Bright, Andrew Carruthers, David Davies, Stephen Haake, James Pang, Al Rodger and Hardial Sagoo. Crises of confidence are safer in the company of other PhD students. Thanks are also due to Jenny Owens (Departmental Secretary) and Carol Seale (formerly of the IHD Office) for their many valuable administrative services. I am grateful for Carol's continued support, even after she had moved to another department. I should also like to thank Dr David van Rest, former Director of the IHD Scheme, who offered me many insights into the process of PhD research. A very sincere thank you to Dr Alastair Cochran, acting Director of the IHD Scheme, who appeared never to tire of answering an endless stream of questions about how a good PhD thesis should be constructed. I have learnt a great deal from Alastair while at Aston, and without his support when I arrived at the University, I would never have started this project.

I owe a great deal to my personal supervisor, Dr John Edwards. His keen interest in the project, and his careful and perceptive analysis of my work, have ensured that I followed a consistent path. His timely expressions of confidence, at the inevitable low points during my project, were especially invaluable. I look forward to continuing my collaboration with John on other projects which we have identified.

Finally, I would like to acknowledge the contribution of my wife, Janice. As always, she has provided me with the encouragement and support, without which I would never have been able to complete this thesis.

LIST OF CONTENTS

LIST OF FIGURES

CHAPTER ONE

CHAPTER TWO

CHAPTER THREE

CHAPTER FIVE

CHAPTER SIX

CHAPTER SEVEN

CHAPTER EIGHT

CHAPTER NINE

CHAPTER TEN

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AIT	Advanced Information Technology
APM	Activity Program Module
ASCII	American Standard Code for Information Interchange
BIS	BIS Applied Systems Ltd
BISBK	BIS Banking Systems Ltd.
BSC	British Steel Corporation
CASE	Computer Assisted Software Engineering
CCTA	Central Computer and Telecommunications Agency
CTA	Cognitive Task Analysis
DB	Database
DBA	Database Administrator
DBMS	Database Management System
DP	Data Processing
DPEA	Data Processing Estimator Association
DPSD	Data Processing System Design
ES	Expert System
4GL	Fourth Generation Language
FOT	Futures and Options Trading
ICI	Imperial Chemical Industries plc.
IHD	Inter-disciplinary Higher Degrees
IKBS	Intelligent Knowledge-Based System
IPSE	Integrated Project Support Environment
ISF	Information System Factory
IT	Information Technology
ITAM	Intelligent Total Advisory Module
KAM	Knowledge Acquisition Module
KB	Knowledge Base
KBS	Knowledge-Based System
KBSA	Knowledge-Based Software Assistant
KE	Knowledge Engineering
LD	Logical Design
LDM	Logical Data Model
LSC	Logical System Chart

MIS	Management Information System
MMI	Man-Machine Interface
NCC	National Computing Centre
NLFE	Natural Language Front End
PC	Personal Computer
PD	Physical Design
POLITE	Produce Objectives Logical/physical design
	Implement Test Edit
R & D	Research and Development
RDA	Relational Data Analysis
RUDE	Run Understand Debug Edit
SD	Structured Design
SDA	Structuring Data
SDLC	Software Development Life-Cycle
SE	Software Engineering
SP	Structuring Processes
SSA	Structured Systems Analysis
SSD	Structured Systems Design
STDC	Software Tools Demonstration Centre
VLSI	Very Large Scale Integration
WIMPS	Windows, Icons, Mouse, Pop-up/Pull-down menus

Chapter One

An Introduction

> Although owing to the envy inherent in man's nature it has always been no less dangerous to discover new ways and methods than to set off in search of new seas and unknown lands because most men are much more ready to belittle than to praise another's actions, none the less, impelled by the natural desire I have always had to labour, regardless of anything, on that which I believe to be for the common benefit of all, I have decided to enter upon a new way, as yet untrodden by anyone else. And, even if it entails a tiresome and difficult task, it may yet reward me in that there are those who will look kindly on the purpose of these my labours. And if my poor ability, my limited experience of current affairs, my feeble knowledge of antiquity, should render my efforts imperfect and of little worth, they may none the less point the way for another of greater ability, capacity for analysis, and judgement, who will achieve my ambition; which, if it does not earn me praise, should not earn me reproaches.
>
> Niccolo Machiavelli 1469-1527

Preamble

This thesis is concerned with work carried out by the author as part of a collaborative Alvey project funded under the Alvey software engineering programme (project number SE057). The project collaborators were the Inter-disciplinary Higher Degrees (IHD) Scheme at Aston University, BIS Applied Systems Ltd (BIS) and the British Steel Corporation (BSC). The aim of the project was to investigate the potential application of intelligent knowledge-based systems (IKBS) to the design of commercial data processing (DP) systems. The thesis also discusses the potential use of conventional software engineering techniques for the design and implementation of knowledge-based systems.

1.1 The Alvey Programme

The Japanese government announced in 1981 their "Fifth Generation", long-term research programme in Information Technology (IT). The Japanese announcement led to the creation of many committees and working parties in the UK, including the Alvey Committee, which were set up to investigate and report on the long-term needs of Britain's IT industry. The reports which emerged all included grave statements linking the future prosperity of the country with the development of its IT industry.

The National Economic Development Office, for example, published a report in November 1982 entitled, *Policy for the UK Information Technology Industry*[NEDO82]. The report contained a wide-ranging list of recommendations for government and industry, covering many aspects of IT development in the UK. It stated that:

> "...the UK needs to have a strong IT industry if its standard of living is not to be further eroded." (Page 31.)

The National Computing Centre's report, *Information Technology Strategy*[NCC82], published in 1983, began by quoting the Prime Minister:

> "I believe that the future prosperity of Britain depends on our being bang up to date in the latest technology, and preferably one step ahead of our rivals..." (Page 4.)

The report went on:

> "The most critical challenge facing Britain today is for the UK industry as a whole to become more competitive in the markets of the world...One industrial sector which is key to the national economy is the information technology sector, since modern business is dependent to an unprecedented degree on the timely availability of accurate information as a prime resource. Thus the capability of the information technology industry to meet the information needs of today's business is central to the successful development of industrial strength in any modern society." (Page 5.)

The Alvey Committee was set up by the government in 1982 under the Chairmanship of John Alvey, then Senior Director of Technology at British Telecom. The Committee submitted their report to the government in August 1982[ALVEY82]. Like the other organisations mentioned above, the Alvey Committee also warned that the future well-being of the nation depended on its ability to compete on equal terms in the international market-place with the IT industries of the USA, Japan and Western Europe. The report acknowledged its debt to the Japanese on its first page:

> "The catalyst to the formation of the Committee was the unveiling last October of Japan's Fifth Generation Computer Programme." (Page 5.)

After outlining an *Advanced Information Technology (AIT) Programme,* the report declared that:

> "Unless there is action to implement the AIT programme...the prospects of the UK competing successfully in the world IT market will be sharply reduced. The spread of advanced IT applications in the UK will also be sharply constrained.

Both of these would be extremely damaging to employment prospects, to our efficiency as a nation, and to our general economic position." (Page 12.)

The Alvey Committee, in accordance with the brief given to them by the government, put forward a comprehensive set of recommendations on how Britain should develop the necessary basis for an internationally competitive IT industry. The central theme of their recommendations was a detailed research and development programme in IT which they believed would have to be followed, if the desired industrial goals were to be realised.

The government announced in May 1983[OAK85] that they were accepting the main recommendations of the Alvey Committee for:

"i a programme of co-operative pre-competitive research in the enabling or underlying technologies of Information Technology;

ii the directed programme to embrace all sectors of the community, government, industrial research laboratories, academic research workers and public or government laboratories;

iii funding, to be made by DTI, MoD, SERC and industry, of £350 million over five years, including £200 million from government;

iv a programme of research much as outlined by the committee." (Page 7.)

The Alvey Committee identified four main enabling technologies as essential to the progress of IT in Britain. These were: software engineering (SE); the man-machine interface (MMI); intelligent knowledge-based systems (IKBS); and very large scale integration (VLSI) of circuits on silicon. Figure I.i shows the scope of the Alvey programme.

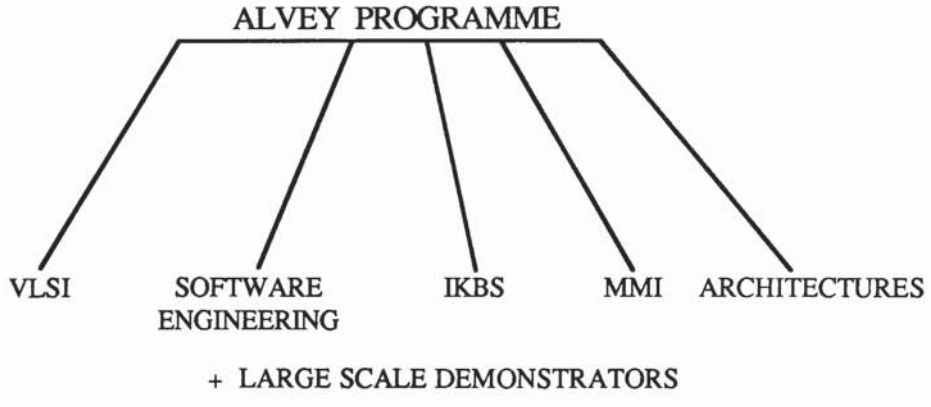

Figure I.i - The Alvey Programme

1.2 The Alvey Software Engineering Strategy

Of the four enabling technologies identified by the Alvey Committee, software engineering was singled out in the technical section of the Committee's report[ALVEY82] as of prime importance. The report stated:

> "A strong domestic capability in the technology of building IT systems is the key to a leading position in the world market. Software is a fundamental component of IT systems and accounts for an increasing proportion of their cost. Correspondingly, the capability to design and build software in the most reliable and cost-effective way is a crucial element in establishing an important position in the world IT systems market." (Page 22.)

An overview of the software engineering programme is shown in Figure I.ii. Although the Alvey Committee[ALVEY82] said that the "...technical leadership...(of the UK)...software industry is well known", their report scorned ad hoc approaches to software development and proposed that:

> "Efficient production, an engineered approach to reliability, conformity with requirements, and economical development and operation must become the UK norm. The strategic objective proposed in this area is that the UK should become a world leader in this Software Engineering technology by the end of the 80s." (Page 22.)

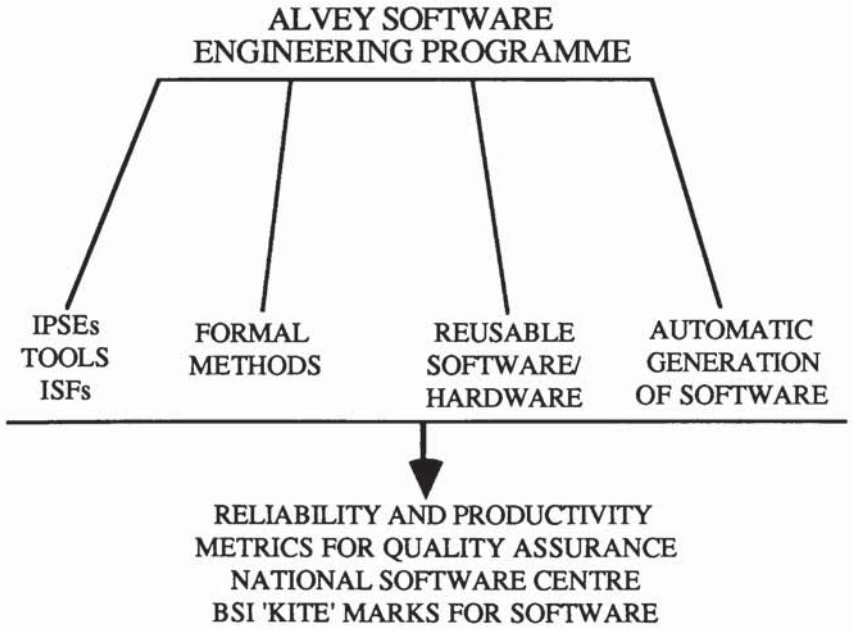

Figure I.ii - The Alvey Software Engineering Programme

The report continued:

> "The programme to meet this objective is focussed on developing Information System Factories (ISFs). The aim is to establish by the end of the decade a lead in the UK's ability to provide ISFs...An ISF will be a computer system, both hardware and software, which provides an integrated set of tools for producing IT systems using software engineering techniques. It will be developed from successive generations of Integrated Programming Support Environments (IPSEs), starting with a consolidated set of the tools available today. IT systems produced through the ISFs will meet the parallel requirements of efficient production and operation, and of improved reliability and performance, which the competitive market of the 1990s will impose." (Page 22.)

Other long-term objectives set for the UK software industry were a *National Quality Control Centre* for software reliability and the inauguration of a *Software Products Brokerage Scheme.*

The phrase *Integrated Programming Support Environment* was soon changed to *Integrated* ***Project*** *Support Environment* within the Alvey community. In fact, the *Software Engineering Strategy* (ALVEY83), published by the Alvey Directorate as early as November 1983, was already employing the 'new' terminology. This change was a recognition by the Alvey Directorate that software tools were required to support the whole of the software development life-cycle, and not just the programming phase.

The Alvey Committee envisaged three generations of IPSEs, with the Third Generation IPSE containing knowledge-based tools. The Alvey software engineering strategy(ALVEY83) said that:

> "The 3rd Generation IPSE (or ISF), containing knowledge bases and 'intelligent' tools, requires significant research which must begin now if the 1989 target date for the Information System Factory is to be met." (Page I.17.)

By *intelligent* the Committee meant that the ISF would:

> "...provide 'automatic' assisted system development from user requirements expressed in high-level terms appropriate to the application rather than the implementation." (Page I.5.)

In August 1984, the Alvey Software Engineering Directorate published a document which gave a more detailed description of what was expected from a 3rd Generation, knowledge-based IPSE containing intelligent tools(DIG84).

It stated that:

> "...intelligent tools may be loosely defined as those which utilise techniques which are normally classified under the general heading of IKBS. Such tools may replace conventional tools (e.g. a rule based syntax checker), enhance conventional tools (e.g. intelligent front end to a complex test generator) or provide assistance for tasks which have little, if any, conventional tool support (e.g. expert system advisor for requirements analysis)." (Page 4.)

The lack of clarity concerning the concept of intelligent tools within the software engineering community, and the need for further research into the general area of IKBS tools and software engineering, was also emphasised in the document:

> "IKBS offers a somewhat different paradigm for system development...However, the IKBS paradigm is immature and unproven. It may not deliver as much as it promises when exposed to the commercial and technical constraints of the market place. Indeed it has yet to address, or even recognise, some of the legitimate concerns of the software engineer." (Page 5.)

(The author of the above report, Tony Dignan, was appointed as the Alvey monitoring officer to the Aston-BIS-BSC project described in this thesis.)

1.3 The Aston-BIS-BSC (*Intellipse*) Project

The *Intellipse* project is concerned with the development of IKBS software tools - tools which could, in principle, be incorporated into a 3rd Generation IPSE. The project proposal submitted to the Alvey Directorate in November 1984[BIS84] defined the objectives of the project as these. To:

> "- demonstrate the potential for the use of IKBSs in systems development,
>
> - provide a basis for tools for systems design,
>
> - promote industry and academic awareness of the capabilities of IKBS in systems development,
>
> - produce a framework of techniques and tools which could be incorporated in the development of an IPSE." (Page 3.)

The proposal related these objectives to the Alvey strategy in this way:

> "This project will contribute to the Alvey objectives in the areas of IKBS and software engineering by:
>
> - improving the efficiency of design of IT systems,

- demonstrating the viability of IKBS in systems development,
- promoting collaboration between academic and industrial partners and,
- dissemination of experience with IKBSs." (Page 3.)

The first Alvey 'Poster' for the project, published in 1985[ALVEY85] after the project had been approved by the Alvey Directorate, contained a further summary of the project's aims:

> "This project aims to produce a set of IKBS tools which can be used in the system design stage of a commercial data processing development. It is proposed that the BIS Structured Design Methodology is used as a basis for this work. The IKBS based tools will have two modes of operation - a designer mode and an adviser mode. It is proposed that, to enable the adviser mode of operation to be effective, a front end query evaluation component should be developed which can be used with each expert system. The expert systems produced could become part of a wider framework of development support tools by subsequent insertion into an IPSE." (Page 129.)

1.4 The IHD Scheme

The Alvey Programme was seen from the start as a collaborative venture between industry and academia. Section 2.3 of the Alvey Committee's report[ALVEY82] said that:

> "The programme should be a collaborative effort between industry, the academic sector and other research organisations, in order to improve the harnessing of our technical strengths to industrial objectives; to get the best value from Government support; and to allow the widest possible involvement and exploitation." (Page 9.)

One of the original aims of the IHD Scheme at Aston University, when it was launched in 1968[COCH81], was to:

> "...encourage industry/university collaboration." (Page 5.)

The IHD Scheme was, therefore, a natural home for an Alvey project. The IHD Scheme has extensive experience of industry/university collaborative projects which combine innovative research with 'real-world' problem solving. The scheme also enables participants in collaborative projects to register for the higher degrees of M.Phil or PhD within the university.

An additional factor favouring the involvement of the IHD Scheme in an Alvey project was that, during the period when the Alvey project was being discussed at Aston, other

organisations connected with the Scheme, were also suggesting collaborative IHD projects in IKBS applications for industrial problems.

Another aim of the IHD Scheme[COCH81] is to:

> "encourage internal University broadening by bringing together disciplines which did not normally collaborate." (Page 5.)

This aspect of the IHD Scheme was felt by the Alvey Directorate to be particularly appropriate since subjects likely to be relevant to the project included: computing; linguistics (for natural language); and psychology (for MMI issues) - these subjects, of course, traditionally residing in separate University departments.

1.5 BIS Applied Systems

BIS Applied Systems Ltd., founded in 1965, provide a broad range of services for organisations which apply and use information technology. One of the company's main interests is in information systems development and they offer a comprehensive list of services to support project management and systems development.

Major investment by the company during the early 1980s, with government backing through the *National Software Products Scheme*, resulted in the development of the *BIS/IPSE* - a 1st Generation IPSE judged according to the Alvey Committee's definition. In the words of one of the company's brochures[BIS86]:

> "Designed by system developers, BIS/IPSE brings a total systems approach offering improvements in quality control and productivity to all aspects of system development. And, unlike many other development aids, BIS/IPSE brings particular benefit to the on-going tasks of system maintenance and enhancement. In particular, BIS/IPSE offers management of, and access to, all project documentation; support tools for formal methods and structured techniques; project control facilities; document completeness and consistency checks, together with design verification, through its integrated data dictionary software..."

The collection of structured design and management techniques are marketed by BIS under the generic term, *MODUS*. *MODUS* services are available to BIS's clients through various forms of consultancy, including training courses and the provision of *Standards* manuals. The company believe that a knowledge-based or expert system-type software development tool, based upon the *MODUS* techniques, is in line with the Alvey software engineering strategy, as well as having important commercial potential.

An unstated, but also an important, aim of the company was to enhance their reputation and credibility in the field of expert systems through their association with an Alvey project.

BIS were responsible for formulating the original project proposal to the Alvey Directorate, and were the prime contractor in the Aston-BIS-BSC project.

1.6 The British Steel Corporation

The third collaborator was the British Steel Corporation's Group Computer Systems Development Team (Strip Products Group), based at Port Talbot in Wales. They are responsible for developing and maintaining data processing systems for the Group's UK-wide operations. These systems are typically to support order processing, inventory control and other similar functions. The Strip Products Group of the BSC represent approximately 40% of the total BSC operation.

BSC's role in the project, as an industrial end-user, was defined in the original project proposal[BIS84] in these terms:

> "The end user partner will be an existing user of a structured systems development methodology and will be able to compare the effectiveness of the proposed tools with those which he currently has, as well as assessing the practicability of the tools in a live development environment by their use on a pilot project." (Page 19.)

The benefits to the end-user of involvement with the project were expressed as:

> "...early access to tools which will enhance his systems development productivity, and the knowledge that such tools will be applicable to his development environment." (Page 19.)

1.7 Project Funding and Staffing

The funding allocated to the project by the Alvey directorate amounted to a total of £800k. Alvey industrial collaborators are funded up to a maximum of 50% of their costs, and academic partners receive 100% funding. The funding was for the equivalent of two full-time staff at BIS for three years and two full-time research officers at Aston for three years. BSC received a much smaller proportion of the total funding than did the two main collaborators, as their role as end-user entailed significantly less effort.

1.8 Thesis Structure

This thesis consists of ten chapters including the introduction. An outline of each of the chapters is given below.

Chapter Two: *The Intellipse Project 1985-88.* An historical overview of the project. A comparison is made of the actual role played by the collaborators with the roles envisaged in the original project specification. The latter discussion is related in particular to the author's work.

Chapter Three: *Knowledge-Based Systems and DP Systems Design.* An introduction to knowledge-based systems technology and the key literature in the field, and a review of the literature on knowledge-based systems and their application in the area of DP systems design (DPSD).

Chapter Four: *The Initial Feasibility Study.* A description of BIS's structured systems design (SSD) methodology, and a report on the findings of the author's survey and interviews with various industrial and commercial software developers, concerning their perceived problems with current development tools and BIS's structured design method.

A development of the research themes pursued during the *Intellipse* project.

Chapter Five: *The Concept and Design of Intellipse.* A discussion of the concept of an *active* or knowledge-based support tool. An initial appraisal of the DPSD domain and its suitability for a KBS approach.

The concept and design of *Intellipse* - an Intelligent IPSE. Explanation of the Advisor/Designer concept and the knowledge-bases required by the *Intellipse* architecture.

Chapter Six: *Advisor.* An account of the design and construction of Advisor. An initial view is given of a structured approach to knowledge engineering, and the lessons learnt from building Advisor are discussed.

Chapter Seven: *The Designer Feasibility Study.* Why a major feasibility study was necessary and how this was undertaken. The results of the feasibility study, the aim of which was to analyse major design tasks in SSD to assess their suitability for KBS

support, are reported.

A discussion of the nature of the expertise possessed by experts in DPSD.

Chapter Eight: *Practical Engineering of KBSs*. The differences between KB and DP systems development are identified. The need for a structured development methodology for KBSs is discussed. The POLITE life-cycle model is introduced.

Chapter Nine: *The ITAM Investigation*. A discussion of the initial steps taken to validate the POLITE model in the context of the ITAM investigation. A review of the early results of using the POLITE model.

Chapter Ten: *Discussion and Conclusions*. A discussion of the main conclusions derived from the author's work.

1.9 Aims of the Thesis

The aims of the thesis will be to show that:

i. CASE tools for DP system design will need to offer more than mechanical support to be effective. CASE tools augmented with *knowhow* about DP design can be built to provide more active support to inexperienced designers;

ii. KBS techniques can be applied successfully in the domain of DP systems design to support the activity of expert and non-expert practitioners, but there are key areas within DP systems design which are not amenable to KBS approaches, based on the use of if-then production rules;

iii. KBSs and conventional software should be fully integrated in future computer-based systems designed to solve problems in highly complex domains;

iv. Conventional software engineering techniques should be used to enrich KBS technology and help the latter become an established technique within the DP community;

v. KBSs can and should be built using an amended conventional SDLC and by adopting an appropriate structured development methodology. KBSs built in this way are much

more likely to have the robustness and reliability demanded in an industrial or commercial environment.

Chapter Two

The *Intellipse* Project 1985-88

All's Well That Ends Well.

William Shakespeare 1564 - 1616

Preamble

The *Intellipse* project has been a close collaboration between Aston and BIS, with BSC involved as and when progress demanded. The collaboration was greatly facilitated by the geographical proximity of the university to BIS's office (about 15 minutes walking distance). The project proposal to the Alvey directorate[BIS84] set out a clear demarcation of responsibility between each of the collaborators, which will be summarised below. However, Aston and BIS have, in practice, worked together on all aspects of the project.

Monthly project meetings have taken place throughout the project, involving representatives of the three collaborators as well as the Alvey Directorate's monitoring officer. In addition, weekly technical meetings have taken place, involving just Aston and BIS personnel.

The author played a substantial role in determining the strategic direction of the project, as well as participating fully in the day to day decision making process. This role was primarily exercised through written proposals submitted to the monthly project meetings. The practical work flowing from the decisions taken by the project team was shared between Aston and BIS. Where BIS or BSC have carried out the major part of any exercise, this is made clear in the thesis. If no indication is given, then the author was responsible for the work described.

2.1 The Original Project Specification

The project specification[BIS84] submitted and approved by the Alvey directorate stated that:

> "This project aims to produce a set of IKBS based tools which can be used in the system design stage of a commercial data processing development." (Page 4.)

The tools were to be based on BIS's paper-based Structured Systems Design (SSD) methodology mentioned in chapter one.

It went on to describe the proposed mode of operation of the IKBS tools.

> "...the IKBS based tools will have two styles of operation, a designer mode and an adviser mode.
>
> When operating in designer mode, the expert system will provide the lead, by working through the procedural rules of the sub-process..[within SSD].., and extracting from the user, the information necessary to carry out the required tasks.
>
> When operating in adviser mode, it is expected that a designer will approach the IKBS with a problem associated with applying either the procedural techniques or documentation rules and will wish to receive advice, and perhaps be 'walked through' a specific part of the the sub-process against his problem.
>
> It is proposed that, to enable the adviser mode of operation to be effective, a front end query evaluation component should be developed which can be used with each expert system." (Page 9.)

Chapter five will describe the architecture of the *Intellipse* tools eventually devised and it will be seen that the *Intellipse* concept differs significantly from the original proposals above.

2.2 The Roles Played by the Three Collaborators

The work to be contributed by each of the collaborators was set out in the project specification as follows:

> "BIS will provide the main input on system design methodology and the building of the designer role IKBS tools.
>
> The academic partner will provide the main input on creating the English based query evaluator, which will provide the adviser interface.

> The user will review the specifications, evaluate the products and make suggestions for modification, as the project develops, to ensure that the tools produced will be of real benefit to subsequent industrial users." (Page 18.)

It became clear from the start that changes would be needed in the designated responsibilities listed above, in order to meet the actual demands of the project objectives, and to utilise effectively the expertise available within each of the collaborating organisations. The changes made, and the reasons for them, are dealt with in the next three sub-sections.

2.2.1 Aston and the Natural Language Component

A brief study of existing commercial and research-based natural language front end (NLFE) systems indicated that four key problems needed to be solved before any work on a NLFE to the proposed IKBS tools could be started:

i. The detailed reasons for including a natural language interface had to be identified, and the consequent functional specification for the interface defined.
ii. The breadth of semantic information to be contained within the IKBS knowledge bases (KBs) had to be specified.
iii. An analysis of the prospective users of the IKBS tools was necessary, to help identify the functional requirements for the NLFE.
iv. It was necessary to know the precise data structures to be employed in the KBs before a NLFE design could be attempted.

It was concluded that the design and functional specification of the IKBS tools was too incomplete, and the extent to which natural language capability was required too vague, to enable any serious work on a NLFE to begin at Aston at the start of the project. It was further decided that the second academic researcher, when appointed, should consider the requirement for natural language during the initial period of his/her appointment. It was expected that, by the time of this appointment, some of the outstanding issues mentioned above would be clarified.

Chapter six will indicate that the natural language capability required by *Intellipse* turned out to be of a very limited nature. The second academic researcher at Aston was mainly

concerned with the more general aspects of MMI design for IKBSs.

The author's attention was concentrated from a very early stage on the feasibility, design and implementation of the core IKBS components. This remained the case throughout the project.

2.2.2 BIS and the IKBS Components

For a number of reasons, it was found necessary that BIS and Aston collaborate more closely on particular areas of the project than was envisaged in the original project proposal.

Firstly, the complexity of systems design suggested that an extensive review of the project specification was necessary to determine the overall feasibility. The proposal submitted to the Alvey directorate had been written mainly at BIS. It was important therefore that Aston, an outside agency, as well as BIS should conduct this review. It was also found desirable for substantial amounts of the knowledge engineering for *Intellipse* to be done by people who were not experts in SSD. An expert practitioner in SSD takes many of its rules and procedures for granted and it is difficult for the expert, alone, to make explicit the principles and techniques which are important for the inexperienced SSD user. The use of knowledge engineers, naive in SSD, was more likely to lead to the successful elicitation of the required knowledge.

Secondly, the paper-based SSD methodology was subject to various internal reviews within BIS around the time the project started. Also, there appeared to be some inconsistency of view between different BIS consultants on particular aspects of the SSD methodology. Thus it was necessary to identify a version of SSD which could serve as a basis for the IKBS tools.

Thirdly, BIS are not primarily a research and development (R & D) company. Their strengths lie in the application and management of established DP techniques to administrative and financial systems within commerce and industry. The lack of experience in R & D was compounded by the particular difficulties inherent in IKBS development, which are absent in conventional DP systems development projects.

At the start of the project, the potential use of BIS's *MODUS* techniques for the

mangement of IKBS development was recognised as an important area for investigation. In fact, it had already been agreed by the project team that, if possible, the applicability of conventional DP development techniques to IKBS design and development should be addressed at some point during the *Intellipse* project. This issue will be discussed fully in chapter eight.

Fourthly, the BIS personnel attached to the project, although possessing wide DP experience, did not have as much experience in IKBSs. No members of the project team were widely experienced in IKBSs, reflecting the fact that in 1985 this was a very new area of technology. Aston, however, did have some practical experience of building a KBS[EDW88] which BIS did not possess. One of the general aims of the Alvey programme is to widen the base of expertise within the UK in this area of technology.

There was initially a little uncertainty on the part of some BIS project team members concerning the applicability of IKBSs to SSD. Inevitably, for a company whose main source of revenue was from the sale of consultancy services, the consistency of involvement of some of the BIS personnel was a problem. During the first year of the project, for example, consultants attached to the project were often called away at short notice to work on other, more urgent client-projects. This caused particular difficulties, bearing in mind the central role which BIS were meant to play in the project.

The combination of these factors indictated that Aston had to take a more active role in the technical management of the project than was originally envisaged. BIS and Aston also had to work more closely on most aspects of the project, instead of each partner having discrete areas of responsiblity. The proximity of BIS and Aston facilitated this closer co-operation. This situation proved successful and the project is reputedly one of the more successful examples of university/industrial collaboration within the Alvey programme[DIG87].

2.2.3 BSC and Structured Design Methods

It would have been desirable for the end-user collaborator to have been a user of BIS's SSD methodology. However, the original choice of end-user, who was an SSD user, had to withdraw at the last minute and BSC were brought in just before the start of the project. BSC were not an SSD user and, in fact, had no "official" structured design method in place. BSC did have their own informal development method. A lengthy

period of induction was necessary for BSC in BIS's structured techniques, although this did not necessitate the adoption of SSD by the company. The lengthy induction period hampered BSC's early involvement in the project, but did not have any adverse effects in the long run.

2.3 The Main Project Phases

This section gives an historical outline of the work involved in the *Intellipse* project. This historical picture is necessary here since, although the thesis chapters are arranged roughly in chronological sequence, their organisation is principally designed to reflect the logical relationship between the various aspects of the *Intellipse* work. Figure II.i illustrates the main phases of the project. The theme of *knowledge-based systems applied to software engineering* (*KBS for SE*) was present throughout the project; the complementary *SE for KBS* theme only manifested itself strongly during the latter phases of the work.

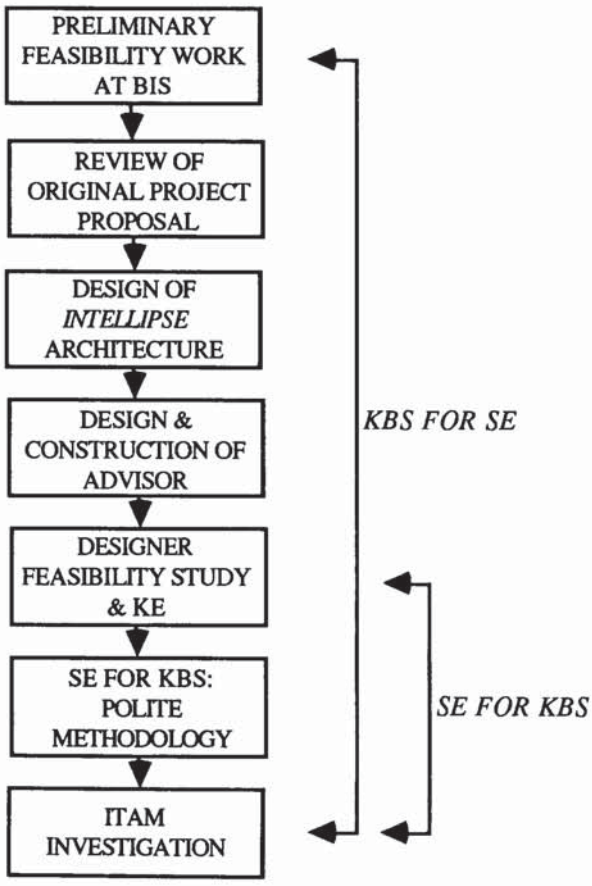

Figure II.i - Main Phases of the *Intellipse* Project

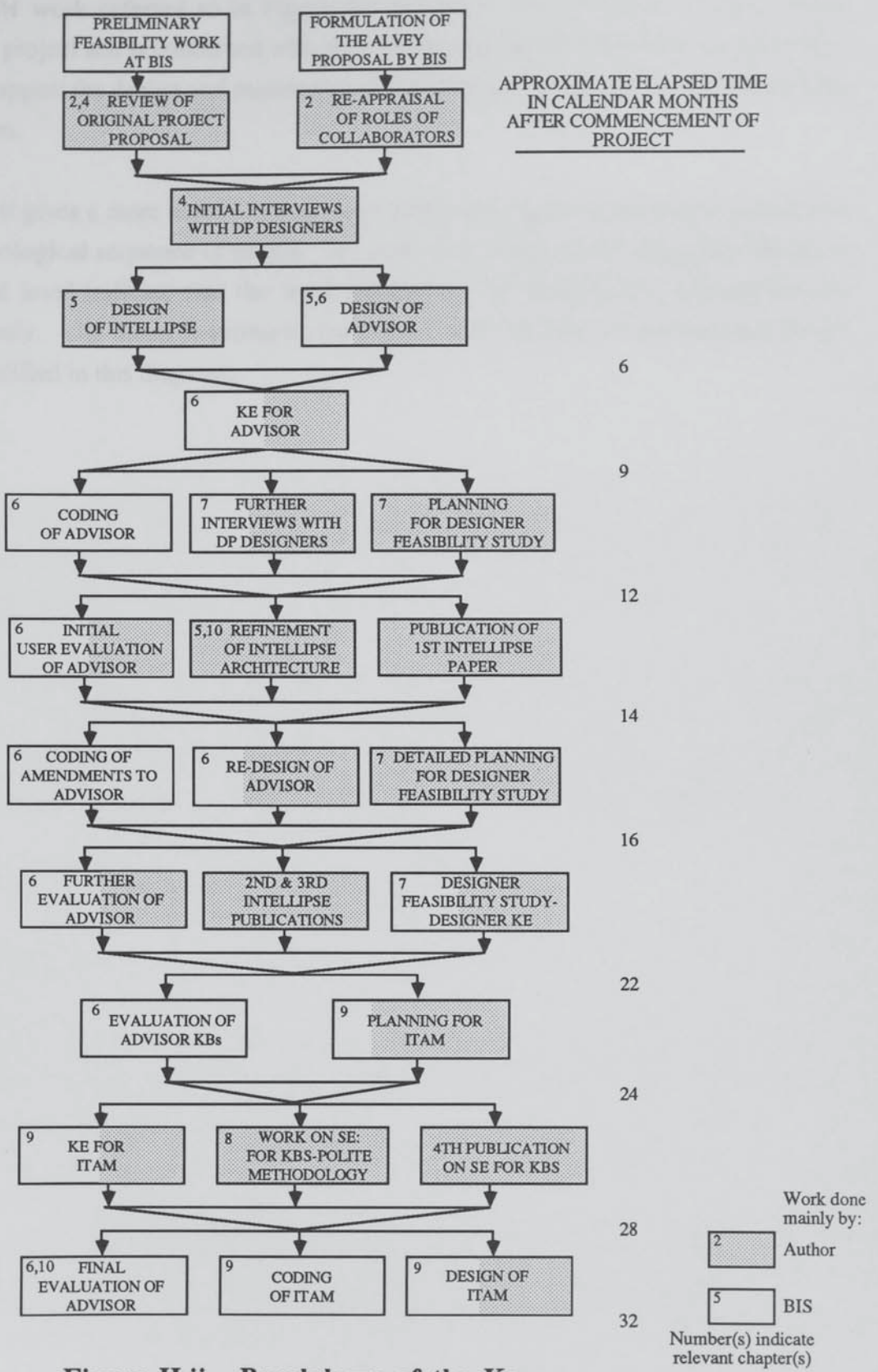

Figure II.ii - Breakdown of the Key Stages in the *Intellipse* Project

The *ITAM* work referred to in Figure II.i relates to the last phase of work of the *Intellipse* project and is concerned with an investigation into the feasibility of using KBS tools to support the design and maintenance of a database management system (DBMS) application.

Figure II.ii gives a more detailed breakdown of the key stages in the project and shows the chronological sequence of events. In Figure II.ii, boxes which appear on the same horizontal level indicate that the work represented by those boxes was carried out concurrently. The thesis is primarily concerned with the author's contribution to the work identified in this diagram.

Chapter Three

Knowledge-Based Systems and DP Systems Design

> ...Rule X
> To gain sagacity, our mind must be trained on the very problems that other men have already solved, and it must methodically examine even the most trivial of human devices, but especially those which manifest or imply an orderly arrangement.
>
> Descartes 1596-1650

Preamble

This chapter discusses the nature of KBSs and the type of problems they have been used to solve. It describes the key features of KBS design and construction, and reviews current research into the use of KBSs to support the software life-cycle.

The chapter does not analyse in detail the differences between building conventional computer systems and KBSs. This topic is more specifically related to the application of traditional software engineering techniques to the design of KBSs, and is discussed in chapter eight, where a structured development methodology for KBSs is proposed.

3.1 What Are Knowledge-Based Systems?

3.1.1 A Definition

The field of expert systems (ESs) emerged from research into artificial intelligence (AI) in the 1960s and early 1970s - mostly in American universities. It is the branch of AI which has received most publicity and has had significant commercial and industrial success. In the *Handbook of Artificial Intelligence*[BARR82] ESs were defined as:

> "...computer systems that can help solve complex, real-world problems in specific scientific, engineering, and medical specialities. These systems are most strongly characterised by their use of large bodies of *domain knowledge* - facts and procedures, gleaned from human experts, that have proved useful for solving typical problems in their domain." (Page 79.)

The *Handbook* also gave a succinct description of an human expert:

> "Specialists are distinguished from laymen and general practitioners in a technical domain by their vast task-specific knowledge, acquired from their training, their subsequent readings, and especially their experience of many hundreds of cases in the course of their practice." (Page 80.)

Hayes-Roth et al in their book, *Building Expert Systems*[HAYES83] said that:

> "The area of expert systems..[within AI]..investigates methods and techniques for constructing man-machine systems with specialized problem-solving expertise." (Page 3.)

Waterman[WAT86] said that ESs are:

> "...special-purpose computer programs, systems that..[are]..expert in some narrow problem area." (Page 4.)

Alex D'Agapayeff, who specialises in the application of ESs in commerce and industry, gave this definition of ESs in a report to the Alvey Directorate which received wide circulation in the UK[DAGA85].

> "An Expert System is a program, or a component of a program, the functioning of which is significantly determined by both the knowledge and the reasoning derived from one or more persons skilled in the domain of the application and from other sources (e.g. books)." (Page 89.)

Arising from the many attempts which have been made to define ESs, a large number of confusing terms are employed to describe this class of computer systems. The most commonly used terms are *expert systems*, *intelligent knowledge-based systems* or *knowledge-based systems;* each term has a different meaning, depending on who is using it. The term used throughout the thesis, when referring to the *Intellipse* tools, is *knowledge-based systems* defined as follows:

> *KBSs are computer-based systems which support, or perform automatically, cognitive tasks in a narrow problem domain which are usually only carried out by human experts. The human expert performs these tasks by employing his/her skill, expertise and judgement acquired and learnt over a period of time. By "cognitive tasks" are meant tasks whose successful completion requires heuristic knowledge and expertise which is not accessible in any other*

organised external form, although the operational KBS can be regarded as a semi-formal description of the human expert's modus operandi.

This definition encompasses the three essential characteristics of a KBS summarised in Figure III.i. The definition has the merit of excluding any reference to the tools or methods used to build a KBS, and is expressed solely in terms of the external, functional characteristics of the system, as these would be observed by an end-user.

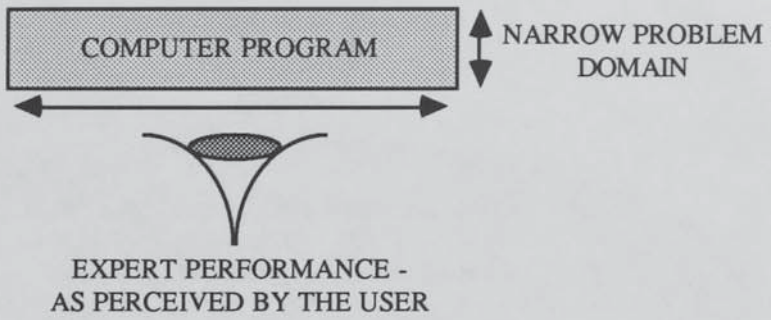

Figure III.i - The Three Essential Characteristics of a KBS

Figure III.ii depicts the author's view of the nature of, and relationship between, work in AI, IKBSs, KBSs and ESs. This relationship can be summarised as follows:

AI Research - research whose objective is the development of new techniques for knowledge representation and heuristic search that could be used in the computer-based modelling of human performance in specific domains;

IKBS R & D - development work designed to turn newly developed experimental techniques into operational technology;

KBS/ES Development - the use of established technology to produce operational systems in specific application areas.

The avoidance of the term *intelligence* when describing operational KBSs in commerce or industry is important. Current operational KBSs do not replicate, except in the simplest fashion, three key attributes of human intelligence namely: abstraction, deduction and learning.

As D'Agapayeff[DAGA85] said in 1985:

> "It is necessary to correct the widespread impression that Expert Systems are inherently complex, risky and demanding. This impression deflects management from competitive developments in both products and applications." (Page 90.)

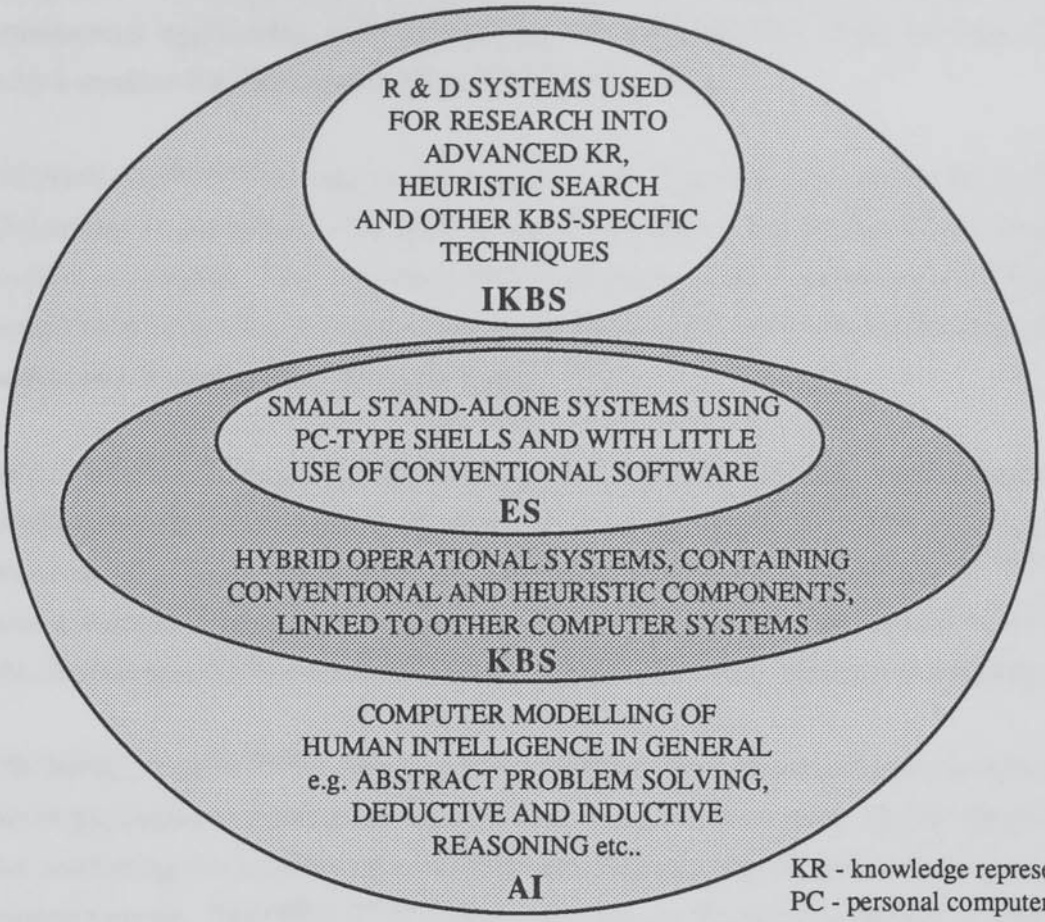

Figure III.ii - Relationship Between Work in AI, IKBSs, KBSs and ESs

In his second survey of expert systems in UK business in 1987 for the Alvey Directorate[DAGA87], D'Agapayeff went on to say that:

> "Phrases like 'artificial intelligence', 'machine learning' and 'superior reasoning systems', relative to a human context, have no meaning whatsoever in current applications. There is nothing remotely akin to natural intelligence or to human reasoning that is being, or could be, programmed now in business." (Page 5.)

The erroneous impression of expert systems in commerce and industry has arisen partly because of the loose employment of the term *intelligence* by many writers, to describe the capabilities of ESs.

3.1.2 Foundations of Expert System Technology

This section describes the first important ESs. These systems, developed in the 1960s and 1970s in US universities, were large systems, implemented in LISP on mainframe computers. Although they were cumbersome, lacking in performance and had limited commercial application, they established the fundamentals of ES technology, upon which modern ES shells and KBS applications are based.

DENDRAL[BUCH69] - one of the first ESs developed in the late 1960s at Stanford University in the USA. The system was designed for the spectroscopic analysis of molecular samples. The main objective of the project was to demonstrate that heuristic search techniques could achieve similar results to exhaustive, conventional algorithmic search in a fraction of the computer-time[BARR82].

MYCIN[SHORT76] - probably the best known of the early US ESs. MYCIN uses if-then production rules and gives consultative advice on the diagnosis and therapy of infectious diseases. Its development led to the first important ES shell, EMYCIN[VANM81] which was a domain-independent version of MYCIN. Empty MYCIN was MYCIN without the domain specific knowledge, but retaining the rule-based inference mechanism.

PROSPECTOR[DUDA79] - this sytem was developed to give expert advice which could assist geologists in finding ore deposits based on geological data. The system is famous for predicting the location of a molybdenum deposit which had not been predicted by human experts. PROSPECTOR's prediction was confirmed by subsequent drilling.

PUFF[KUNZ78] - a rule-based ES for the interpretation of pulmonary function test results. According to Hayes-Roth et al[HAYES83], PUFF was in routine use in the laboratory and 95% of its reports were accepted without modification.

Many of the early systems shared the architecture shown in Figure III.iii. The separation of an ES into a knowledge base and an inference engine remains an important concept in ES development. It will be useful later to contrast the similarities and differences of the *Intellipse* tools and these first ESs, and to consider whether the early definitions of what constitutes an ES need to be modified.

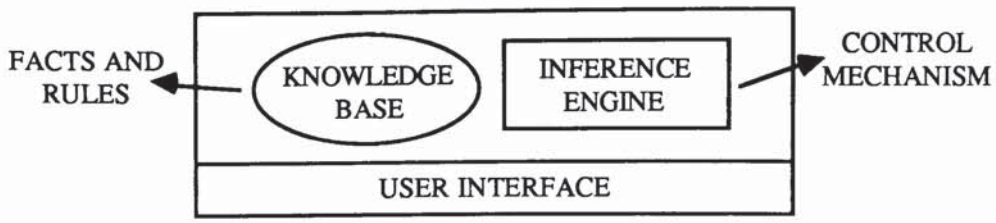

Figure III.iii - Architecture of Early Expert Systems

3.1.3 Styles of KBS

The early expert systems described above supported predominantly diagnostic tasks. However, as Hayes-Roth[(HAYES83)] et al and Waterman[(WAT86)] pointed out, a variety of tasks can be supported by a KBS. Hayes-Roth et al list the following types of task:

"Interpretation	Inferring situation descriptions from sensor data
Prediction	Inferring likely consequences of given situations
Diagnosis	Inferring system malfunctions from observables
Design	Configuring objects under constraints
Planning	Designing actions
Monitoring	Comparing observations to expected outcomes
Debugging	Prescribing remedies for malfunctions
Repair	Executing plans to administer prescribed remedies
Instruction	Diagnosing, debugging, and repairing student behaviour
Control	Governing overall system behaviour." (page 33.)

The type of task performed is directly linked to the role the KBS is playing in relation to its user. The list of KBS roles below is adapted from Worden[(WORD87)].

Assistant - invoked by the user to perform a specific task as part of a wider exercise;
Critic - reviews work already completed by the user and comments on its accuracy, consistency and completeness;
Second opinion - executes a task and compares its results with those of the user;

Expert consultant - offers advice or an opinion given certain information;
Tutor - trains the user to expertly perform a specific task;
Automaton - completes an expert task automatically and independently of the user;

The roles toward the top of this list are those in which the user's expertise is greater than that of the system. As we move towards the end of the list it is the KBS which operates as if it has greater expertise than the user.

The particular style of operation of a KBS will depend on whether the system is used by an expert or a non-expert, and the degree to which the KBS has automated the task being supported. As Waterman[WAT86] said, a KBS can be used to distribute or archive expertise within an organisation. It can improve the efficiency and consistency of an expert's performance, as well as relieving him of the burdens of training and the solution of routine problems; the expert can then concentrate on more difficult problems. All these factors can influence the style of a KBS.

3.1.4 Interaction of KBSs with Existing Systems

Commercial DP installations tend to accumulate significant amounts of computer software over a period of time. Newly developed software must usually be integrated with these old systems. The interaction of KBS tools with existing computer systems, has therefore been an important issue during the *Intellipse* project and is discussed later in the thesis. The author's interpretation of the way in which a KBS can interact with other computer systems is given below and is illustrated in Figure III.iv.

Stand-alone - interaction is with the user only: these systems can be characterised as small systems implemented using PC-based ES shells - see Figure III.ii;

Integrated - interaction is with company databases, management information systems or PC-based tools, such as spreadsheets and database packages. The operation of the KBS involves the exchange of data between the KBS and other systems, as well as direct interaction with the user;

Embedded - complete absorption of a KBS within an information system. The user does not interact directly with the KBS but through the user interface of the host system. The latter handles interaction with the KBS components.

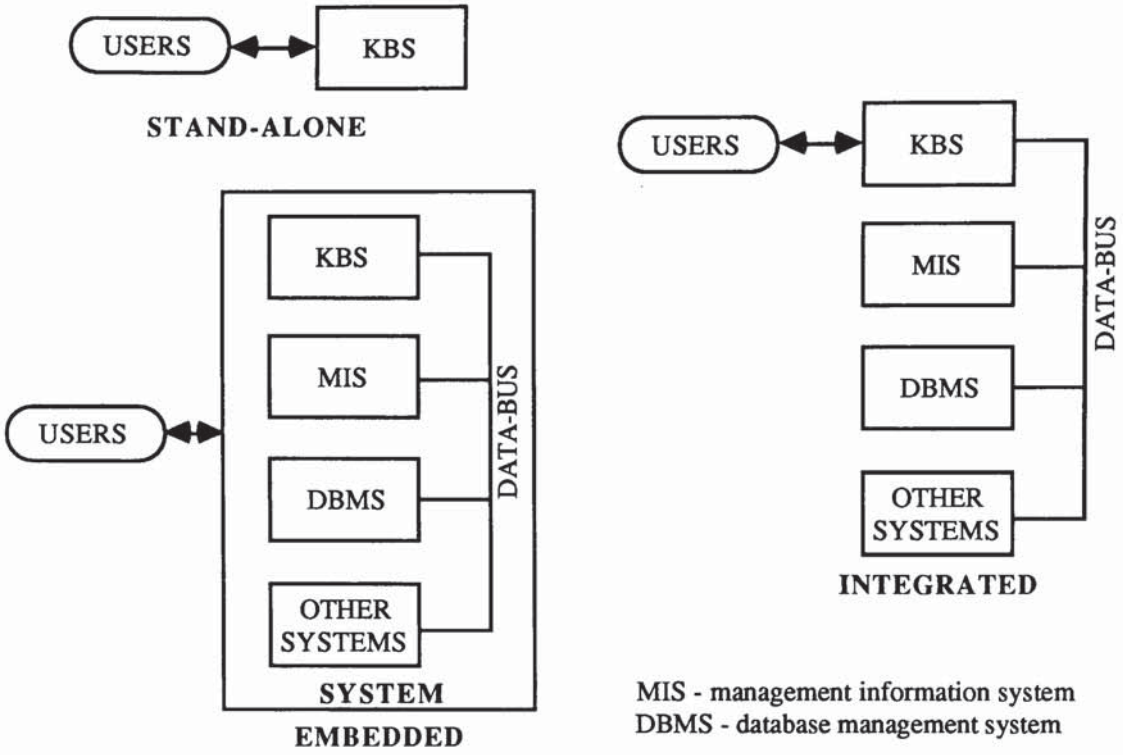

Figure III.iv - Types of KBS Interaction with Other Systems

The majority of reported KBSs are characterised by the first mode. However, there is an increasing awareness that, in order to maximise the effectiveness of KBSs, the second and third modes must be used more often[(SMI84), (DAGA87), (WORD87), (JONES87a), (NOM87)].

3.2 Applications of KBSs

3.2.1 Problems Addressable Using KBSs

Most writers on operational KBSs stress the importance of selecting an appropriate problem before attempting a KBS approach, but few offer a precise set of criteria for measuring this suitability factor[(BAS84), (BOB86), (HAYES83), (JONES86a), (PRER85), (WAT86), (ZACK87)]. Of the literature cited above, Prerau's paper[(PRER85)] gives the most useful guide to assessing the suitability of a problem for a KBS approach. The difficulty of establishing measurable criteria has meant that much of the understanding of what makes

a problem amenable to a KBS approach has been obtained empirically. The factors summarised below are an interpretation of the criteria judged to be important by the authors cited above. The importance of factor ix below has been particularly borne out by the experience of the early part of the *Intellipse* project.

i. *Narrowness of the domain*: the problem should be well-bounded, and the boundaries, within which data relevant to the solution can be found, well-defined. Problems which rely significantly on general knowledge about the domain are very unsuitable for a KBS approach.

ii. *Complexity of the problem*: the problem should not involve a great deal of common sense reasoning. This point is related to that of general knowledge. It should also be possible to structure or factor the problem solution to some extent, and to partially represent the solution, for example, in the form of a hierarchical structure diagram.

iii. *Nature of the problem*: the problem should be primarily cognitive in nature. Problems requiring visual or other sensory skills are not appropriate.

iv. *Nature of the experts*: except in a very small number of cases, at least one expert in the domain is necessary. The expert needs to be able to articulate his knowledge and be available and willing to co-operate in the project.

v. *Training*: established domains, like medicine, in which expert performance is attained after long periods of systematic formal training, are more likely to be suitable for a KBS. The training techniques will reflect explicitly the problem-solving strategies built up over many years, and knowledge acquisition may therefore be easier.

vi. *Speed of solution*: does the problem take seconds, minutes, hours or days to solve? Problems which must be solved in real-time, involving large quantities of data, may involve insurmountable performance difficulties for a KBS approach, at present.

vii. *Sensitivity of the problem*: domains such as medicine, the law or

personal finance can involve sensitive ethical and legal issues. The use of a KBS in these domains, therefore, may not be appropriate, even though it may be possible to construct them.

viii. *Conventional solutions*: it is unwise to attempt to build a KBS until conventional, algorithmic solutions have been considered. The complexities of KBS design and construction are such that conventional solutions, if available, are likely to be less expensive to develop, and easier to maintain.

ix. *Written material*: the existence of manuals, procedures, case studies and other documentation in the domain will greatly assist the development of a KBS - especially in the early stages of knowledge acquisition.

In chapter five a detailed comparison is made between each of these factors and the DPSD domain in general. In chapter seven, where the *Designer-mode* feasibility study is described, a comparison is made between specific tasks within SSD and these factors.

3.2.2 Operational KBSs

Several hundred systems bearing the names IKBS, KBS or ES are reported in the literature. Very few of these can be classed as operational systems. Waterman[WAT86] identified 180 systems in his *catalog of expert systems*. Only eight of these were classified as "commercial systems" and only two were said to be in everyday use.

Obtaining precise information on operational KBSs is difficult. This is partly because of commercial secrecy, partly because some systems may not be classed as KBSs by their owners or developers, and lastly because there is no accepted definition of the term *operational*. In an article in *Computing* in 1987[DURH87], Brian Johnstone, the manager of Intelligent Systems at *Istel*, was quoted as saying that:

> "The users of expert systems form a very elite club with extremely limited membership! Most expert systems activity is in the academic or prototyping environment. The number of expert systems in regular everyday use is still probably less than 100 worldwide." (Page 22.)

Buchanan[BUCH86] listed over sixty "working" ESs covering twelve application domains.

The *CRI Directory of Expert Systems*[CRI86] lists six hundred ESs. The Ovum Report on *Commercial Expert Systems in Europe*[OVUM86] identifies fifty examples of "operational" expert systems and two hundred "commercial" ESs.

Judging by two conferences[IES87], [AVIG87] in London and Avignon which were oriented towards operational ESs, and the D'Agapayeff survey mentioned earlier[DAGA87], the use and development of KBSs in commerce and industry is increasing. KBS technology has some way to go before it can be said to be in widespread use, but the indications are that its importance is being increasingly recognised in commerce and industry.

The best publicised operational KBS is Digital Equipment Corporation's *R1*, now called *XCON*. In 1980 McDermott[MCD80] described R1 as follows:

> "R1's domain of expertise is configuring Digital Equipment Corporation's VAX-11/780 systems. Its input is a customer's order and its output is a set of diagrams displaying the spatial relationships among the components on the order; these diagrams are used by the technician who physically assembles the system." (Page 269.)

In 1980, according to McDermott, R1 had a database of 420 components and a knowledge-base of 772 rules. In 1986[VAN86], van de Brug et al reported that R1 had grown to a database of 10,000 components and a knowledge base of 4,000 rules. Dennis O'Connor, the DEC executive responsible for the original R1 project, was quoted in 1987[DURH87] as claiming that R1 had saved his company $40 million in that year.

3.3 The Key Features of KBS Construction

3.3.1 Knowledge Engineering

One of the main problems in constructing a KBS is the elicitation and representation of the expertise of a human expert, and the translation of this knowledge into a machine-executable form. This process is usually referred to in the literature as *knowledge engineering* and much has been written on the subject, particularly in relation to knowledge acquisition[HART86], [HAYES83], [OLS87], [READ87], [SCHW87], [WAT86].

Hayes-Roth et al[HAYES83] and Waterman[WAT86] identified five stages in the evolution of an expert system:

> "Identification — Determining problem characteristics
> Conceptulization — Finding concepts to represent the knowledge
> Formalization — Designing structures to organize knowledge
> Implementation — Formulating rules that embody knowledge
> Testing — Validating the rules that embody knowledge." (Page 24.)

Waterman[WAT86] described knowledge engineering:

> "The process of building an expert system is often called *knowledge engineering*. It typically involves a special form of interaction between the expert-system builder, called the *knowledge engineer*, and one or more human experts in some problem area. The knowledge engineer 'extracts' from the human experts their procedures, strategies, and rules of thumb for problem solving, and builds this knowledge into the expert system..." (page 5.)

The author prefers the term, *cognitive task analysis* (CTA), to describe the process of knowledge acquisition during KBS construction. This term is used to signify that CTA is a phase of development of KBSs which can be regarded as complementary to the stages of conventional analysis, design and implementation in commercial DPSD. This idea will be discussed again in chapter eight where the differentiation between KBS development and DPSD is made more explicit. The description of knowledge engineering given is the author's summary of the process as it has been described by the authors cited above. This description will be compared later in the thesis with the approach taken during the *Intellipse* project.

3.3.2 Cognitive Task Analysis

A simple view of an expert's knowledge or expertise is that it consists of facts, rules, heuristics and an inference strategy. The following simplified example concerning the operation of a circuit consisting of a battery, bulb and connecting wire, explains each of these categories.

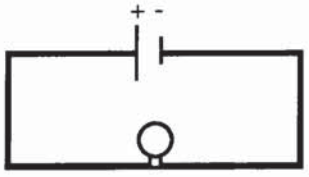

Figure III.v - Simple Circuit Diagram

Example of some FACTS: metals conduct electricity;
copper is a metal.

Example of a RULE: if the connector is copper,
and the battery is OK,
then the bulb will light.

Example of a HEURISTIC: if the bulb does not light,
then the battery is probably flat (Prob. 0.7),
or the bulb might be faulty (Prob 0.2), or the
connector may not be a conductor (Prob. 0.1)
CHECK THE BATTERY FIRST!

INFERENCE STRATEGY: a set of heuristics used by an expert to navigate efficiently from a problem description to an acceptable problem solution, without necessarily having to identify and test all candidate solutions.

Sommerville[SOMM83] made clear the distinction between a *rule* and a *heuristic* in the context of KBSs.

> "The distinction between a 'rule' and a 'heuristic' is that if a rule is applied to a set of outcomes, those which do not match the rule cannot be solutions. If a heuristic is applied, those which do not match the heuristic could be solutions but are not likely to be so." (Page 149.)

The objective of cognitive task analysis is to identify and codify the facts, rules, heuristics and inference strategy which the expert employs to solve problems in the application domain. The exercise has three basic components:

Acquisition - acquiring the basic knowledge from the expert.
Representation - organising and structuring the knowledge.
Execution - codifying the knowledge into a machine-executable format.

Figure III.vi summarises the process of cognitive task analysis. CTA can be very complex, due to the intrinsic difficulty of extracting and representing the knowledge of an expert and the acquisition stage, in particular, is often a bottleneck in KBS development.

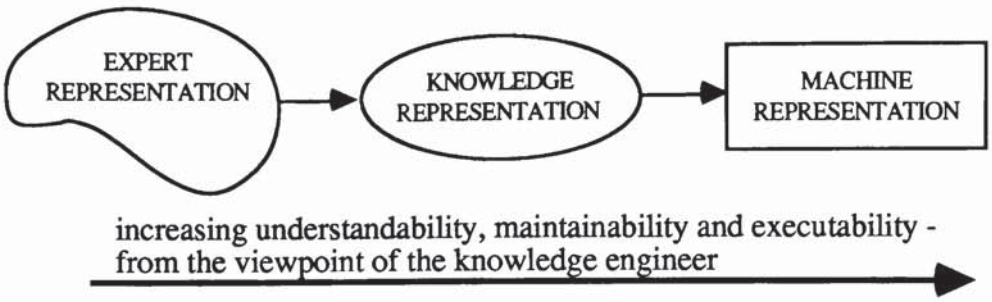

Figure III.vi - Cognitive Task Analysis

Hayes-Roth et al[HAYES83] summarised this problem:

> "Several major difficulties are involved in acquiring knowledge for an expert system. One of the most troublesome is representation mismatch, the difference between the way a human expert normally states knowledge and the way it must be represented in the program. Others include the human's inability to express knowledge possessed, limits on expert system technology, and the complexity of testing and refining the expert system." (page 153.)

The next three sections give an overview of the key issues involved in the acquisition, representation and execution stages respectively. Where the research described in the thesis refers to specific knowledge engineering techniques, these will be discussed more fully at the appropriate point in the text.

Acquisition

Many techniques have been devised for knowledge acquisition. Some have been borrowed from conventional systems analysis and subsequently modified to suit the needs of KBS analysis. Hart[HART86] identified the most commonly used knowledge acquisition techniques as *structured interviews, protocol analysis, repertory grid analysis, rule induction* and *case study observation.* These are different techniques by which the knowledge engineer can make explicit the inferencing process and data which the expert uses to solve a problem. The choice of technique can be influenced by many factors, some of which are listed below. This summary is based on Hart[HART86] and Hayes-Roth[HAYES83]. The observations and conclusions, made on the basis of the knowledge engineering done during the *Intellipse* project, are discussed in more detail later in the thesis.

- the availability and motivation of the expert;

- the ability of the expert to articulate verbally, or otherwise, his/her expertise;
- the degree of complexity of the problem domain;
- the nature of the expertise involved;
- the overall scale of the project;
- the availability of manuals and other written material;
- the experience of the knowledge engineers;
- the availability of tools or computer-based environments for supporting acquisition;
- the amount of development resources available.

The material which first emerges from the acquisition process may be in the form of transcripts of taped interviews, video recordings, hand-written notes, rough diagrams and so on. The next step requires the organisation and represention of this raw knowledge in a particular representation schema.

Representation

While the choice of acquisition technique is largely dictated by factors which leave little room for manoeuvre by the knowledge engineers, the choice of representation schema is more open. Five basic data structures are generally used for knowledge representation: *production rules*, *frames*, *semantic networks*, *objects* and *declarative logic*[WAT86]. Each of these mechanisms is very versatile and the ultimate choice will depend on the type of procedural control required and the degree of familiarity of the knowledge engineers with the different techniques. The representation schema can also be influenced by the features offered by the implementation tools being considered since many of the KBS toolkits are designed to support one particular form of knowledge representation schema, or inference mechanism.

Execution

Once the knowledge representation schema has been chosen, the raw cognitive model must be systematically translated into the new format. This can be a paper-based exercise although in many cases, especially when prototyping, the raw knowledge is transferred directly into the shell or toolkit being used. Where paper-based translation is used, the coding into machine form usually takes place during the implementation phase.

3.3.3 Physical User Interface and Explanation Facilities

Very often the impetus for building a KBS is the desire to remove the organisational bottleneck which arises when only a limited number of experts are available to perform an activity lying on some critical path in an organisation. The intention may be to enable non-experts to perform tasks previously only carried out by the expert. In this case, it will be necessary to define precisely the competence of the expected non-expert user, so that an operational KBS is built which does not rely for its success on a level of expertise in the user which the latter does not possess. For example, any questions which the system poses to the user must not require judgement which the user is not able to exercise. This point was discussed by Edwards and Bader [EDW88].

There is very little literature about the techniques that can be used to ensure that an interface is devised, which will maximise the usability and effectiveness of an operational KBS. This aspect of KBS design is still the subject of basic research[BERR86a], [BERR86b]. The following is a summary of the factors which the author concludes may significantly influence the choice of interface:

- the degree of expertise of the prospective KBS user;
- the amount and nature of direct user interaction required;
- the amount of textual explanations required;
- the importance of static or animated graphics;
- the degree of menu-type interaction necessary;
- the amount of concurrent help facilities required during operation;
- the appropriateness of windows, icons and mouse-style operation (Wimps);
- the importance of making explicit the inference process and data being used.

The information necessary to evaluate these factors is related to the type and style of KBS, the nature of the expert task and the degree of competence of the prospective users.

It is often desirable for highly user-interactive KBSs, in domains such as medicine or finance, to make explicit to the user the reasoning behind any advice offered. For this type of KBS, elaborate explanation facilities may be built into the system. One of the main characteristics of early KBSs, as D'Agapayeff[DAGA85] pointed out, was their

ability to "..provide explanations of their reasoning on demand". The explanation facilities provided with most KBS tools simply reproduce, in machine format, the particular rules used in the last consultation. This type of facility is useful for the knowledge engineer when developing the system, but its relevance to the end-user of the KBS is limited. However, the ability of an expert system to explain its reasoning is still regarded by some as a key feature of the technology. Waterman[WAT86] emphasised this in his book on ESs referred to earlier:

> "Most current expert systems have what is called an *explanation facility*. This is knowledge for explaining how the system has arrived at its answers. Most of these explanations involve displaying the inference chains and explaining the rationale behind each rule used in the chain. The ability to examine their reasoning process and explain their operation is one of the most innovative and important qualities of expert systems." (page 28.)

The amount and type of explanation facilities in a KBS will ultimately depend on the type of application domain, the style of the KBS, and the nature of the intended user.

3.3.4 Tools Used to Build KBSs

The tools used to implement KBSs are normally from one of the following groups.

i. Expert system shells running on personal computers or mainframes.
ii. AI environments or toolkits running on high power personal workstations. The latter are characterised by their use of high resolution graphics and Wimp-intensive operation.
iii. AI programming languages like Lisp, Prolog and OPS5.
iv. Conventional languages such as C, Pascal, Fortran etc.

The languages identified in *iii* and *iv* can run in any of the three hardware environments mentioned.

Waterman[WAT86] listed six questions that should be considered when selecting a tool for ES development:

> "Does the tool provide the development team with the power and sophistication they need?
> Are the tool's support facilities adequate considering the time frame for development?

Is the tool reliable?
Is the tool maintained?
Does the tool have the features suggested by the needs of the problem?
Does the tool have the features suggested by the needs of the application?"
(Page 143.)

Expert system shells and AI toolkits are specifically designed to support the knowledge representation schemata and procedural control commonly used for KBSs. As a result they can be very fast and convenient to use, once their extensive range of facilities has been mastered. However, they do not offer the execution speed and flexibility for interface design which is possible using conventional programming languages like C and Fortran. It is also the case that building a KBS in an AI or conventional language can be very expensive in development time and it is unlikely that the resulting code could be utilised easily in another application domain.

Choosing the right tool to build a KBS is a complex task, requiring careful consideration of a large number of interdependent factors. Figure III.vii illustrates some characteristics of the different tools available and shows the two most frequently used tools.

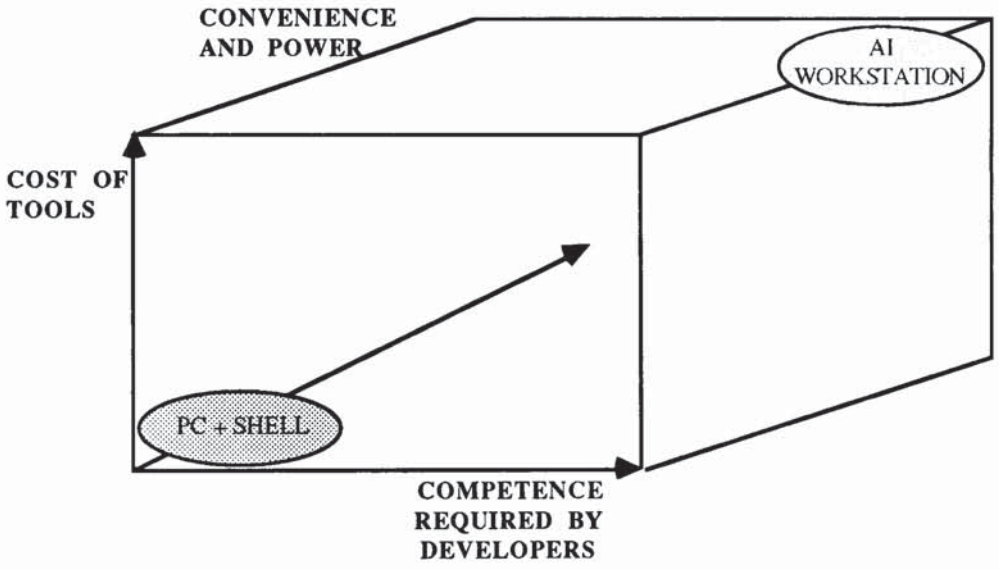

Figure III.vii - Comparison of KBS Tools

The following is a list of important factors which should be considered in relation to KBS tool selection. It illustrates the *logical* factors which can influence tool selection; pragmatic commercial considerations which may also be relevant are not discussed here.

SCOPE:	Prototype or operational system?
SIZE:	Small (< 100 rules), medium (< 500 rules), large? Volume of external data involved.
TYPE:	Stand-alone, integrated or embedded?
STYLE:	Assistant, critic...?
PROBLEM:	Classification, prediction...?
USER:	Expert, novice, public..?
USER INTERFACE:	Graphics, Wimps, animation...?
EXTERNAL INTERFACE:	Does the KBS need to link to a PC, MIS, DBMS, other..?
KR:	Frames, objects, rules..?
CONTROL:	Type of inferencing mechanism required?
ALGORITHMS:	How many conventional programs required?
DISTRIBUTION:	Single PC only; distributed PCs + floppy disks; LANs; mainframe..?
PERFORMANCE:	Batch, on-line, real-time..?
RESOURCES:	Development budget and time constraints?
DEVELOPERS:	Experience of knowledge engineers? Training required to use development tools?

3.4 Research into Automating the Software Life-Cycle

This section discusses the application of AI to the domain of software engineering. Ince et al[INCE86] summed up the relationship between the two disciplines:

> "One of the more exciting developments in software engineering over the past few years is the promise that techniques in artificial intelligence hold out for the software developer. That advances in areas such as: natural language processing, intelligent knowledge-bases systems and logic programming languages will enrich activities such as software design and requirements analysis." (Page 81.)

Although there is plenty of evidence of attempts to use AI techniques to support the software development process, much of the research must be regarded as having only long-term benefits for commercial DP in the UK. Most of this research is looking at ways to build support environments which can *automate* most of the life-cycle, through the generation of executable code by automatic transformation of high-level user-specifications.

There is little evidence of research into the application of KBSs to specific tasks within the current, conventional life-cycle. This is an important area since this type of approach could bring immediate benefits to the DP industry. Selecting particular tasks within the DP life-cycle, and building tools to support them, is likely to be easier and bring more immediate results than the AI research mentioned earlier.

A long-standing aim of AI research has been the automation of the complete software life-cycle. Frenkel[FREN85] summed up this objective in the following way:

> "At its most ambitious, automatic programming synthesis expresses the idea that automatic programming is the purest sort of automation. A team of programmers or users would focus their efforts on developing a complete specification of a user's requirement. Then an intelligent system would write the program in much the same way that today's compilers generate machine code from high-level languages." (Page 579.)

Frenkel's paper also described the general research area of applying AI techniques to software development:

> "Expert systems are now being considered by some as tools for improving today's software development techniques and programmer activity...However, they attempt to automate or facilitate only some phases of the software-development cycle, like debugging or maintenance. They are practical and short-term efforts improving on techniques within the current person-based software approach.

> Some researchers advocate a more ambitious approach. They see not only a shift from manual to automatic programming development, but they also focus on management aspects of large program-development projects. Those pursuing this course propose that stages of the software-development life-cycle, like requirements specification, implementation, documentation, and maintenance, can be automated if emphasis is placed not on the products of these processes but on the processes themselves. Because these processes are knowledge intensive, they could be managed with knowledge-based automated tools." (page 578-9.)

Research into 'intelligent' support for the software life-cycle can thus be grouped under two headings:

AI research: the long-term goal of an *environment* providing complete life-cycle support, with the ultimate aim of *automatic* generation of executable code from user-specified natural language specifications.

KBS research: the short-term goal of KBS tools to support *specific* tasks within the overall software cycle in the context of current, person-based, *conventional* development methods.

3.4.1 AI Research

The *Handbook of Artificial Intelligence*(BARR82) listed several "Automatic Programming Systems" (Volume II).

PSI(GREEN77) An objective of the PSI system, built at Stanford University, was that users should be able to specify programs interactively, using a loose high-level specification language. The application area addressed by PSI is symbolic programming, including information retrieval, simple sorting, and concept formation. The system was designed to allow the user to specify the desired program with a mixture of examples and mixed initiative, natural language dialogue. Two parts of the PSI system, LIBRA(KANT79) and PECOS(BARS79) were able to function as stand-alone systems.

PROGRAMMERS APPRENTICE(RICH78) This system was developed at the Massachusetts Institute of Technology in the late 1970s. The *Handbook* described the operation of the system as this:

> "The intent of the Apprentice is that the programmer will do the difficult parts of design and implementation, while the Apprentice will act as a junior partner and critic, keeping track of details and assisting the programmer in the documentation, verification, debugging, and modification of his (her) program." (Page 343.)

The PSI system depended for success on its ability to analyse natural language specifications. Generating executable code via English-like specifications requires sophisticated facilities for parsing and analysing natural language. This remains a major problem in the automatic synthesis of programs. Referring to this in relation to the PSI project, Frenkel[FREN85] said that:

> "..specification was extremely difficult because users usually did not know what they wanted; for those limited cases when they did know, languages, or formalisms, for expressing specifications were too complicated. There was little advantage in writing a specification that would be as long as the program itself, and in a language that was almost harder to write and more error prone than the program language." (Page 579.)

Work on the PSI system led to the design and implementation of the CHI system at the Kestrel Institute. The *Handbook* discussed the aims of the CHI project:

> "Automatic programming research in general, including the PSI project, has concentrated on methods for compiling programs expressed in a very high level language. The goal in CHI is to provide not only a knowledge-based synthesis system, but also a supportive, high-level programming environment that includes knowledge-based specification acquisition, consistency checking, debugging, editing, and maintenance...The CHI system uses a common knowledge base about the programming process to support all of these activities." (Page 334.)

This last quotation from the *Handbook* shows that AI research into automatic programming has evolved into a quest for knowledge-based environments which allow large parts of the development process, from high-level user-specification through to executable code generation, to be automated. Smith et al[SMI85], describing research at the Kestrel Institute in Palo Alto, USA, into knowledge-based software environments, confirmed this view:

> "The complexity of producing programs suggests that we begin to formalize and mechanize the programming process. In pursuit of these related goals of factorization and mechanization of the programming process, Kestrel Institute has as its primary focus research on knowledge-based software environments. That such systems are *knowledge-based* means that some of the diverse logical materials of the programming process are factored out and represented formally in a knowledge base. That they are *software environments* means that they are intended to provide automated support to the entire software lifecycle." (Page 1278.)

Other reported automatic programming projects in the literature are DEDALUS[MANN78], SAFE[BALZ78] and APE[BART81].

In 1983 the Kestrel Institute sponsored meetings to discuss a long-term research programme for applying AI to software development. The meetings resulted in the publication of a report setting five, ten and fifteen year research goals for a *Knowledge Based Software Assistant* (KBSA)[GREEN83]. Figure III.viii illustrates Balzer's[BALZ83] view of the KBSA. (Balzer was a participant in the meetings which led to the report.)

Figure III.viii - Generalized Knowledge-Based Software Assistant Structure
(From BALZ83 and GREEN83)

Balzer related the concept of the KBSA to an *automation-based software paradigm*. This paradigm has three basic elements[BALZ83]:

- formal specifications created and maintained by end-users;
- user-generated formal specifications which become prototypes of the desired system ensuring that the system will be responsive to user needs;
- implementation issues such as algorithm choice, control structure, representation selection, caching intermediate results, buffers etc. are excluded from the specification. These issues are decided either by computer or by a person and then carried out automatically.

Balzer went on to say that:

> "The automation-based software paradigm both facilitates and requires the existence of an assistant [..the KBSA..]. This is a consequence of having all of the development processes-requirements analysis, specification, implementation, and maintenance-machine mediated and supported. The development processes must be broken into individual activities so that the individual activities can be mediated and supported, and the decisions and rationales behind them recorded." (Page 43.)

The first five year phase of the KBSA is due for completion in 1988. The work in the first phase was split into five sub-projects[BENN88], each conducted by a separate contractor. These were: the *Project Management Assistant* (Kestrel Institute), the *Requirements Assistant* (Sander's Associates), the *Specification Assistant* (USC/Information Sciences Institute), the *Performance Assistant* (Kestrel Institute) and *Framework* (Honeywell Systems and Research Centre).

The Alvey *Information System Factory* (ISF) project has already been referred to in section 1.2. The ISF concept has similarities with the KBSA project in the USA. A project brochure[ALVEY87a] issued in 1987 by the four ISF collaborators, GEC Research, STC Ltd, and the Universities of Edinburgh and Lancaster, described their initial view of the ISF:

> "The broad specification of the ISF is that it should be a complete computer system, providing an integrated toolset for developing IT systems both hardware and software, based on sound engineering techniques. The ISF is to be developed from successive generations of IPSEs, an IPSE being a coherent set of tools covering the entire lifecycle of systems development, supporting both management and technical roles within a project." (Page 2.)

Although this passage does not state it explicitly, it is envisaged by the Alvey Directorate that the ISF will involve extensive use of knowledge-based tools. The Dignan report[DIG84] also referred to in 1.2 made this point clear. The ISF project is a long-term venture, although an interim report has been produced which is discussed in 10.3.

Alvey's idea of a software *factory* is not new. Bratman and Court[BRAT75] described an attempt to develop an integrated set of software development tools to support a disciplined and repeatable approach to software development. However, the environment they describe was very much oriented towards *program* design and implementation. The factory did not address phases in the life-cycle such as requirements analysis, project management and maintenance. In this respect a better

description of their concept would have been a *Programming Factory.*

Barbacci[BARB86] described work, at the Software Engineering Institute of Carnegie Mellon University in the USA, into the concept of a Software Factory. The Institute is attempting to lay the basis for the development of a Software Factory sometime in the future.

Eliot and Scacchi[ELI86] proposed a *Knowledge-Based System Factory* (KBSF):

> "The KBSF represents an innovative experiment in large-scale software development via concurrent application of advanced software tools, knowledge-based techniques, and strategies for organizing large development teams in the use of these tools and techniques...A research project, the KBSF explores knowledge-based factory concepts. Creating usable tools for actual industrial settings is one important outcome of our efforts." (Pages 55,56.)

According to Eliot et al the KBSF project in 1986, after five years work, had produced a number of usable tools representing 250,000 lines of code. The KBSF tools are also designed to support VLSI circuit design, thus reinforcing the idea of a *system* factory.

At the annual Alvey conference in 1987 a "technical cross-boundary session" was held to discuss *IKBS in Support of Software Engineering and Vice Versa.* A report[BADER87b] of the session summed up the the relationship between the two disciplines of SE and IKBS, within Alvey:

> "The UK was at the leading edge of IPSE technology. However, current generation IPSEs were complex and expensive tools and relied for their success on the expertise of the analyst or designer using them. If IPSEs were to get into widespread use the next generation would have to be designed so that the non-expert could utilise them relatively easily. This meant that IPSEs must play a more active role in system development by assisting the user at various points within development, and by automating more activities within the design process. The latter could be achieved by incorporating knowledge-based tools and AI techniques within an IPSE. The knowledge bases would have knowledge about design as well as information about project management, configuration control and other key aspects of the software life-cycle." (Page 12.)

Another Alvey project is concerned with the idea of a knowledge-based IPSE. The ISM project has similar objectives to those of the KBSA, ISF and *Intellipse* projects, and involves Software Sciences Ltd and the Universities of Lancaster and Keele. The ISM project Consortium described the aims of their work in 1987[ISM87]:

> "The ISM project is a research programme which is investigating the structure and

> applications of a knowledge-based IPSE. Its objective is to develop a prototype kernel for a knowledge-based IPSE along with demonstrator applications for the system...The ISM is a prototype for a knowledge-based IPSE. It is not a complete IPSE in itself as it does not include a complete range of tools. In essence, the project is concerned with investigating how knowledge based methods may influence the design and use of an IPSE." (Pages 1,2.)

Lubars and Harandi[LUB86] identified five areas where AI techniques could be used in the software life-cycle:

- schematic knowledge-based techniques to encode expert design knowledge in a form that can be accessed and incorporated into new software designs,
- rule-based refinement techniques to perform a stepwise refinement of specifications and high-level designs into detailed designs,
- constraint propagation techniques to propagate design decisions and constraining specifications from one part of a design to other affected parts of the design,
- planning techniques to solve design problems by automatically composing design components, and
- agenda-driven control strategies to help track goals, dependencies, and objectives to assist in managing design complexity for the user.

Lubars et al described an *Intelligent Design Aid System* (IDeA) implemented on a Unix-based workstation, which has been used to aid the design of small system examples. Their work developed ideas put forward in two of their earlier papers[HAR85a),(HAR85b] which considered the use of template-based specification and design. This concept envisages a library of stored design templates and a system for effective retrieval and manipulation of the templates. The latter represent abstract and generic solutions for different types of problem. The design templates, which can be in the form of fragments of data-flow diagrams, are the basis for a stepwise process involving the user interactively refining the design.

Blum and Sigillito[BLUM86] discussed the general theme of using expert systems for the design of information systems. They identified two types of knowledge appropriate to the information systems development domain:

- *application knowledge*, about the application environment and its needs, and

- *transformation knowledge*, about the software process itself.

The application knowledge is sub-divided into knowledge about a specific application, and generic knowledge about a whole class of applications. The transformation knowledge is divided into heuristic design knowledge, and algorithmically prescribed transformation knowledge.

Blum et al went on to describe an *Environment for System Building* (ESB) consisting of three modules:

- the *definition* module, for capturing the specification;
- the *transformation* module, an expert system to transform the specification into an executable specification;
- the *generation* module, a program generator to transform the executable specification into an operational program.

The ESB project began in 1984 and prototypes of some parts of the system have been implemented.

Persch[PER86] defined a model of a software environment based on an *expert system for transformations*. As with most transformation systems, the approach is based on the successive translation of formal specifications into an executable format. Persch's system is unusual in that it uses a transformation development environment constructed like an expert system. It consists of rules, an inference "machine" and the "explanator". The rules are implemented in Prolog. The system works by applying a given set of rules to map documents from one phase to another. The system has been used for some Ada-based applications.

Pidgeon and Freeman[PID85] proposed an idealised architecture for a *Design Quality Expert* (DQE) system. The DQE is expected to be an active design aid and to be able to:

- provide alternative suggestions for improvements to the design,
- explain or justify diagnoses and suggested improvements,
- be able to implement improvements selected by the designer,
- have the capacity to acquire new rules, and
- be able to instruct the neophyte.

The DQE architecture consists of a natural language/graphics front end, a knowledge-base and a blackboard. The knowledge base has facts, rules, an audit trail and performance statistics; the blackboard is for plans, agendas and solutions. It is also envisaged that the DQE will have a *learning* component for the acquisition of new rules, a *teaching* component for instructing, an *historian* for capturing an audit trail of changes to the design, and an *observer* to monitor not only the performance of the human designer, but also the performance of the rule base. The DQE is an idealised system and has not yet been implemented.

Grindley[GRIN86] considered the use of expert systems for the development of computer systems. He divided the problem domain into two basic areas; solution expertise needed, and problem definition facilities.

The solution expertise is made up of four parts:

- designing the program strategy,
- coding the programs,
- debugging the programs, and
- altering and adding to existing coded programs.

Grindley also separated the problem definition facilities into four areas of expertise:

- structuring the enquiry process,
- confirming the evidence (for each requirement),
- being friendly (asking questions of the user in layman's terms), and
- prompting the specifier (for other implied requirements).

A *Systematics Generator* is proposed which would contain *precoded solution templates*, *rules* for extracting the required parameters from the problem statement, a *parametizer* for applying the rules to the problem statement, and a *generator* which accepts the parameters and applies them to the relevant templates to automatically produce a solution-statement of coded and tested programs.

RUBRIC[VANA88] is an Esprit project looking at the applicability of a rule-based approach to information systems development. (Esprit is a European Community AIT programme with similar objectives to the UK Alvey programme.)

The basic RUBRIC paradigm is that:

> "...development of an information system should be viewed as the task of developing or augmenting the policy knowledge base of an organisation, which is used throughout the software development process, from requirements specification through to the run-time environment of application programs." (Page D3-7.)

Work on the definition of the concepts and development framework for RUBRIC is complete. Implementation of the system has begun on *Sun* workstations using Prolog.

3.4.2 KBS Research

Software Cost Estimation

Chapman and Seiler[CHAP86] described an expert system that can assist engineers and managers in estimating software development efforts and schedules. The system, COCOMOx is based on Boehm's[BOEH81] Cocomo model for software cost estimation. It was built using a proprietary expert system shell.

Cuelenaere et al[CUEL87] discussed the role of an expert system in supporting the calibration of a software cost estimation model. They developed a prototype expert system containing about 200 rules.

Edwards-Shea et al[EDWA87] described the development of the *BIS/Estimator*. This is an expert system for estimating development times for DP software systems. It is written in Prolog and is based on an empirical model of cost estimation. The model is calibrated using the expertise of BIS consultants built up over many years of actual development experience. *BIS/Estimator* is available as a commercial product from BIS Applied Systems Ltd. and runs in the IBM PC environment.

Specification

Kampen[KAMP85] described research at Boeing Computer Services (BCS) into the application of AI to software engineering. In particular, Kampen discussed a prototype system called ARGUS II used for supporting the development of formal specifications. The goal of the BCS research project is said to be "an expert system that elicits specifications directly from the customer or end-user of an application".

Programming

Lewis Johnson and Soloway[LEWI85] described PROUST, a knowledge-based system for analysing and understanding Pascal programs written by novice programmers. Knowledge about what implementation methods should be used in programming is codified in PROUST. In a preliminary test of the system, PROUST constructed complete analyses of 161 out of 206 sample programs. It is reported that, of the "problem" programs it identified, 95% did have faults.

Oddy[ODD86] reported on the Knowledge Based Programmer's Assistant (KBPA). The KBPA is an advanced interactive software development tool which uses techniques from AI to help programmers in the task of program composition. Like PROUST, the KBPA contains knowledge about programming plans and mechanisms. The KBPA is a prototype system implemented on a *Sun* workstation.

Project Management

Hurst et al[HUR85] report the work of an Esprit project which looked at the application of rule-based techniques for software project management and maintenance. Hurst et al described a system called the Software Production and Maintenance Management System (SPMMS). SPMMS uses rules to guide its internal operation. In addition, the end-user is able to enter rules which describe the supporting environment for the software project. Rules can also be used to control the products and processes relevant to the project. SPMMS is a pilot system and research into its applicability in a real environment is continuing.

Basili and Ramsey[BASI85] described ARROWSMITH-P a protoype expert system for software engineering management. They investigated the use of rule-based deduction and frame-based abduction for building expert systems in the management domain. Metrics were employed in the system such as, "software changes per line of source code", and "programmer hours per software change". The performance of the expert system was compared with what actually happened during the development of a project. Basili et al claimed that the system performed "moderately well".

Maintenance

Dyer[DYER84] considered the application of expert systems to the problem of planning for

software maintainability. Dyer concluded that:

> "The use of rule-based systems to evaluate some aspects of software design may not be practical at present because the complexity of the implementations would require a prohibitively large number of rules." (Page 299.)

Leung and Choo[LEUN85] discussed the potential of Prolog for efficiently managing the information concerning the different modules, and their inter-relationships, in a large software system. They described a possible architecture for a rule-based design and maintenance expert system.

Alperin and Kedzierski[ALPE87] described Problem Manager (PM) which is one of a number of "expert managers" forming part of a larger Knowledge-Based Software Development Environment used by the US company, Carnegie Group Inc. The environment is a framework for building integrated tools to support software managers, designers, programmers, and maintainers. PM deals with software problems, including bugs, changes, and enhancements to the system. A prototype system has been built using the AI environment, *Knowledge Craft*. Boeing Computer Services, mentioned earlier, are funding the work.

3.4.3 AI and the Rapid Prototyping of Conventional Software

Dunning[DUNN85], in a paper which proposed the use of expert systems for supporting the rapid prototyping of conventional software, said that:

> "In contrast to the sequential steps involved in the traditional software development life-cycle, rapid prototyping allows rapid iterations through design and implementation. A prototype is not a complete implementation of the system, but is a partial implementation representing the major required functions and the interfaces between functions. The prototype represents what the system is to do and how it will be accomplished." (Page 3.)

Dunning described the use of LISP-based tools to animate the specifications and designs of conventional systems, but points out that it is likely that the LISP-based prototype will need to be re-implemented in a conventional language before an operational system is obtained.

Ince[INCE88] suggested that the use of artificial intelligence languages could greatly facilitate the prototyping of conventional software. He proposed that a set of user requirements could be expressed as *rules*, and the rules coded into Prolog allowing the

user-specification to be demonstrated very quickly. Ince cited the development by the accountants, Arthur Anderson in Chicago, of a system for accounting for the movement of equipment from one oil lease to another. The requirements for the system were very complex and an expert system shell was used to incrementally build up a prototype specification of the system, which allowed the users to check that the system was behaving correctly in different situations. The ES model of the design was subsequently used to build and test a conventional implementation of the accounting system.

Many of the proposed "new" paradigms[AGRE86] for information system development are based on some form of rapid prototyping. However, so far, the proponents of the prototyping approach have not identified AI techniques as a key element in the new strategy. The use of prototyping to accurately identify user-requirements was summed up well by Carey and Mason[CAR83]:

> "Prototypes...attempt to present the user with a realistic view of the system as it will eventually appear. With prototypes a distinct attempt is made to produce a 'specification' which users can directly experience. Communication with users, particularly the non-specialist middle management user, is a major motivator behind the recent interest in prototypes." (Page 49.)

The use of prototyping in DPSD for the purpose described by Carey et al is closely analogous to techniques used in civil engineering. For example, when a new building is being designed, the developers often build scale models to help finalise the design, and satisfy interested parties that the proposed structure will blend in appropriately with its surroundings. The close relationship between traditional engineering design and DPSD suggests that the potential use, if any, of KBSs in the engineering disciplines should be investigated, to see if there are any lessons for the applicability of KBSs in the DPSD domain.

3.5 KBSs in Engineering Design

Mittal[MITT86] characterised the link between KBSs and design in the following way:

> "Designing an artifact is one of the most challenging problem-solving tasks performed by engineers. It is a task requiring both large amounts of domain-specific knowledge (that is, knowledge specific to the class of artifact being defined) as well as considerable problem-solving skill." (Page 102.)

Hayes-Roth et al[HAYES83] identified design as a generic application area for KBSs. They characterised design as: "configuring objects under constraints." (page 14).

There is evidence that AI in general, and KBSs in particular, are being used extensively in engineering design. Duffy[DUFF87] published a bibliography covering AI in engineering design which cites over 300 papers. An annual conference on *Applications of Artificial Intelligence in Engineering Problems*[SRIR86] was held for the first time in the UK in 1986 and has become an annual event. Rychener[RYCH85] discussed the application of ESs for engineering design. His analysis was related to chemical and civil engineering and he concluded:

> "Expert systems are being applied to a wide variety of engineering design domains where expertise is known to exist and to be in demand. They are effective in providing a starting basis for ongoing projects that can make significant impacts in industrial and commercial settings." (Page 40.)

Few authors make explicit the link between traditional engineering design and DPSD, yet the emergence of the term *software engineering* in the 1960s, and structured development methods in the 1970s, was an explicit attempt to put the construction of computer systems on the same footing as disciplines like civil and mechanical engineering. Wasserman[WASS80] made this point very clearly:

> "In the 1960's, software developers attempted to design and implement increasingly complex systems...In many cases, developers were unable to construct systems that were suitable for the envisioned application. Even those systems that worked were often unreliable, poorly documented, inefficient, and/or in need of extensive maintenance...To a large extent, their failures may be attributed to the generally ineffective character of software development practices at that time...The result of this situation could be seen in the lengthy delays or even outright collapse of a number of large-scale software projects. Identification of the problems and discussion of possible approaches to their solution took place at several meetings in the late 1960's, two of which were sponsored by the NATO science committee. The term 'software engineering' was coined as the theme for these meetings, and was chosen as a provocative term, to indicate that the design and construction of software should be viewed as an engineering discipline." (Page 664.)

The traditional engineering disciplines are much older than software engineering, and have evolved rigorous methodologies and systems of training. In addition, they are grounded on proven mathematical and physical principles. Software engineering has not yet developed a widely accepted formal mathematical foundation and is still too young a discipline to have established a widely accepted engineering methodology. In this connection, Boehm[BOEH76] said that:

> "Those scientific principles available to support software engineering address problems in an area we shall call *Area 1: detailed design and coding of systems software by experts* in a relatively *economics-independent* context. Unfortunately,

> the most pressing software development problems are in an area we shall call Area 2: *requirements analysis design, test, and maintenance of applications software* by *technicians* in an *economics-driven* context. And in Area 2, our scientific foundations are so slight that one can seriously question whether our current techniques deserve to be called 'software engineering'...Hardware engineering clearly has available a better scientific foundation for addressing its counterpart of these Area 2 problems. This should not be too surprising, since 'hardware science' has been pursued for a much longer time, is easier to experiment with, and does not have to explain the performance of human beings." (Pages 1239,1240.)

Hebden[HEBD86] said:

> "Software engineering is a relatively young discipline. It is somewhat younger than the computer industry itself and, like any evolving discipline, there has to be a delay between the theoretical work which forms its basis, the availability of proven tools which enables the theory to be put into practice and the experience gained from having made use of the tools." (Page 199.)

Research into *formal methods* for software specification and design is an attempt to provide software engineering with a mathematical basis similar to that which underpins the traditional engineering disciplines. There has been great emphasis in the Alvey software engineering programme on projects researching into formal methods. Of the ninety software engineering projects listed in the 1986 Alvey Annual Report[ALVEY87b], thirty six (40%) are in the general area of formal methods. This reflects the desire of some parts of the UK software industry, especially those involved in the production of real-time, 'mission' or safety-critical systems, to establish rigorous, algebraic techniques that can ensure the production of provably correct software. Talbot, who was Director of the Alvey software engineering programme, said[TALB86]:

> "The Formal Methods programme..[within Alvey]..above all aims to raise the professional standards of software engineering. There is a need for at least the same level of confidence in certain forms of software as, for example, in civil engineering where there are techniques and methods applied in the design process to give a quantitative measure of the suitability of a civil engineering design for its environment. The use of Formal Methods is the key to being able to certify software in safety-critical applications." (page 23.)

Ince[INCE88] commented that the optimistic expectations of the Alvey Directorate in the area of formal methods have not been fully justified by events.

> "The formal methods developers are seeing very limited take-up of their ideas, and we are far from the prediction by the Alvey Directorate that, by the end of the programme, 30% of all software engineers would be familiar with the techniques associated with mathematical software development." (Page 21.)

This observation by Ince suggests that the widespread use of formal methods in commercial DP must be regarded as a long-term objective.

The design and development of a commercial DP system involves many of the problems associated with more traditional engineering design. These problems include control and management of a large team of people with diverse but specific skills, budgeting and scheduling human and physical resources, and building complex systems to meet both physical constraints and aesthetic requirements. However, as we have noted, there are two essential differences between software engineering and traditional engineering: the former is a more immature technology as far as development methodologies are concerned, and it does not have a firm mathematical foundation. Formal methods research is investigating potential mathematical foundations for the software process; research into KBS tools for supporting DPSD can help strengthen existing engineering-like development techniques. In addition, the success of KBSs in other engineering design areas suggests that KBS tool support can be an effective way of enhancing the design process.

The similarities between traditional engineering design and DPSD, coupled with the evidence that KBSs are being successfully used in the former domain, lends credence to the idea that KBSs can be successfully applied to the problems of information system development. The detailed nature of the tasks performed in the two domains are fundamentally different and we cannot, therefore, expect that KBSs used in engineering could be used in DPSD. However, KBSs in engineering do suggest generic areas of design where these systems may be applicable in DPSD.

3.6 Summary

The review of reported work into the application of AI to software engineering has shown that the research is in two broad areas: automation-based AI environments for the whole software life-cycle and KBS tool support for specific tasks within current conventional development methods. The former area is characterised by Balzer's[BALZ83] work and the KBSA project[GREEN83] at the Kestrel Institute. KBS research has been more limited and there is little reported research into the use of KBS tools for the design phase in the commercial software development life-cycle. While the AI environment research offers potentially more powerful support for the software engineering process than KBS tool support, it is unlikely that the results will be available in a commercial environment in the near future. In fact the use of environments such as

the KBSA is likely to require fundamental changes in the nature of commercial DPSD, as well as engendering significant managerial and organisational changes. The development of KBS tools with more limited scope, to support particular tasks within the current software life-cycle, could bring more immediate benefits to the commercial DP environment.

The *Intellipse* project is concerned with the development of KBS support tools. However, there is a link between Balzer's view of the automation-based software paradigm and the basic concept of the *Intellipse* system. Like the KBSA, the architecture proposed for the *Intellipse* system is also based on the notion of a knowledge-based support environment for DPSD incorporating discrete knowledge bases relating to the different tasks in the design process. The essential difference between the *Intellipse* concept and the KBSA is that the latter is based on the paradigm of automatic code generation via very high-level specifications - to be achieved through incremental *formal* transformations of a very high-level specification language[BALZ85] which minimises human intervention. The *Intellipse* concept is more limited in scope and is based on the paradigm of expert support for the human designer at specific points in the design cycle via KBS tools, together with some automation of mechanical tasks where this is appropriate[BADER87a]. In addition, the *Intellipse* project is not primarily concerned with the specification stage of the DP life-cycle, except insofaras this is part of the BIS SSD methodology.

There is another important difference between the *Intellipse* research and KBSA-type projects. The *Intellipse* project is concerned with developing tools which could be incorporated into a typical DP installation found in the UK today. This implies that the tools should be compatible with existing tools and development methods, and run on hardware which is acceptable in these installations. As has been said, KBSA-type projects are long-term programmes, unlikely to produce tools which could be used in a commercial or industrial setting in the near future.

The distance from industrial reality of some current research into IKBS applications in software engineering was noted at a joint AlveyDirectorate/British Computer Society workshop held in 1985. A report[OWEN85] of the workshop said that:

> "Not only do the software engineering and intelligent knowledge-based systems communities tend not to speak to each other, but neither group tends to speak to the data-processing managers responsible for running today's mainstream computing systems in business and industry. Hence the directions of software

> engineering and IKBS advances could bear little relation to the real-life needs and problems of the working DP community." (Page 13.)

The *Intellipse* project is attempting to address current problems in the commercial DPSD environment. It offers the prospect of KBS tools which could be introduced into this environment in the very near future. In this respect, the *Intellipse* project is clearly addressing the need, identified by the Alvey Directorate, for KBS research to relate directly to the "real-life needs and problems of the working DP community".

Chapter Four

The Initial Feasibility Study

Faust
Trust honesty, to win success,
Be not a noisy jingling fool.
Good sense, Sir, and rightmindedness
Have little need to speak by rule.
And if your mind on urgent truth is set,
Need you go hunting for an epithet?

Johann Wolfgang Goethe 1749-1832 (Faust/Part One)

Preamble

This chapter introduces the BIS/SSD methodology. It describes the methods used for the initial feasibility study and the conclusions resulting from it. A description of the organisations and people used in the *Intellipse* research is given and the detailed themes of the research are identified.

4.1 The BIS Structured System Design Methodology - SSD

The SSD method is based on the theories of structured systems design formulated in the 1970s[(STE74), (YOU79), (JAC75)]. Stevens et al summed up the philosophy of structured design:

> "Structured design is a set of proposed program design considerations and techniques for making coding, debugging, and modification easier, faster, and less expensive by reducing complexity...Simplicity is the primary measurement recommended for evaluating alternative designs relative to reduced debugging and modification time. Simplicity can be enhanced by dividing the system into separate pieces in such a way that pieces can be considered, implemented, fixed, and changed with minimal consideration or effect on the other pieces of the system." (Page 115.)

The other major advantage of structured design was identified by Wasserman[(WASS80)]:

> "...increased effort in the earlier stages of development would be reflected in reduced costs for testing and maintenance. Early detection of errors (in the specification, for example) could reduce the cost of correcting those errors at a later time. The resulting software would be of higher quality and would be more likely to fulfill the needs of its users." (Page 665.)

The SSD methodology is one of a large number of system development methodologies marketed commercially in the UK. The better known methods are SSADM, LBMS, SADT, INFOREM, CORE, JSP, JSD, IE, DDSS[MADD83), (DOWNS86), (CONN85), (JONES86b)]. Some of the methodologies, such as SSADM and INFOREM, address more than one phase of the software life-cycle and others, like JSP, address the programming phase only. It is beyond the scope of the thesis to discuss the nature and relative merits of these methodologies in relation to SSD. However, SSD has a significant number of major commercial and government users and the Alvey Directorate regarded it as sufficiently important in its own right and representative of other methods, to make it a suitable basis for a project researching into IKBS tools for structured design.

SSD is very similar to the *Structured Systems Analysis and Design Methodology* (SSADM) which is approved and recommended by the government's Central Computer and Telecommunications Agency (CCTA). SSADM was originally developed by the CCTA in conjunction with the company, Learmonth and Burchett Management Services Ltd. (LBMS), whose founders were former employees of BIS and worked on early versions of BIS's structured techniques. SSADM is a structured development methodology which goverment departments, as well as private sub-contractors, are expected to use when building administrative computer systems for government use.

SSD is part of BIS's *MODUS* services for computer-based information system management and development. *MODUS* covers the areas of strategic planning, resource management, project management, system development and production management. The components of *MODUS* are procedures, techniques, documentation, automated aids, training and implementation support.

The *MODUS* framework recognises the following phases for the conduct of a project:

- feasibility study,
- systems analysis,
- systems design (computer and clerical),
- programming,
- system testing,
- acceptance testing,
- implementation, and
- post-live review.

SSD is concerned with the *design* phase which follows systems analysis and precedes programming. The development phases are regarded by BIS as discrete entities in their own right, and the use of SSD need not be in relation to any of the other *MODUS* phases. The input to the SSD phase is a set of documentation, arising from the analysis of a user's system requirements. This analysis usually covers any existing manual or computer systems relevant to the project, and data and files which will be needed by the proposed system. The output from the SSD phase is a set of program and file specifications from which the software for a computer system could be built. Wasserman[WASS80] has described the nature of the design phase in the software life-cycle:

> "...the importance of the design process has become evident, since the output of the design activity is a *software blueprint* that can be used by the programmer(s) to implement the system without having to refer back to the specification and without having to make unwarranted assumptions about the requirements." (Page 672.)

The SSD course manual suggests that any design method should:

- reduce design complexity,
- assist detection of errors at an early stage,
- improve documentation,
- provide for effective project control,
- increase the ability of non-technical staff to contribute,
- enforce the consideration of factors often overlooked, and
- cope with bad input, i.e. functional specifications of poor quality.

The manual also states that SSD incorporates the following features:

- a formal statement of design criteria,
- a formal design method leading to a system with simple interfaces between independent segments,
- design reviews,
- documentation,
- methods for estimating design performance with reference to given performance parameters, and
- a simple interface between design and programming.

The underlying principle of SSD is that the design process should be rigorously divided into two halves: *logical* and *physical* design. In principle, the logical design of the system should be devised without any consideration of the physical constraints the system must meet. This feature of SSD is related to the concept of *simplicity* referred to by Stevens et al above, and is characteristic of most semi-formal development methods. A more detailed description of SSD will emerge in chapters five, six and seven.

4.2 Organisations and People Involved in the *Intellipse* Research

The *Intellipse* research has drawn on the domain expertise of a large number of people outside the immediate project team. Information about the organisations and persons involved is collected here for convenience, and references to organisations or individuals in the rest of the thesis will be with respect to this section.

BIS Applied Systems

A general description of BIS was given in 1.5.

BIS1 - Principal consultant-Birmingham (promoted to Divisional Director during the *Intellipse* project). *Intellipse* project manager. Responsible for BIS's Technical Division (TD) for the Birmingham region.
Specific expertise: several years experience of using *MODUS* methods on development projects; data analysis; expert systems.

BIS2 - Senior consultant-Birmingham. Member of project team.
Specific expertise: training people in the use of structured programming; use of *MODUS* standards on projects; developer of BIS's introductory course on expert systems.

BIS3 - Consultant-Birmingham. Joined the project team and BIS at the end of the first year of the project.
Specific expertise: several years programming experience in a DP environment.

BIS4 - Consultant-Birmingham. Member of project team. Main programmer of *Intellipse* tools.

BIS5 - Senior consultant-London/TD. Used extensively as a domain expert.
Specific expertise: SSD; database systems (DBS) design; data analysis (DA); experience of several major development projects using *MODUS*.

BIS6 - Senior consultant-London/development division (DD). Used as a domain expert.
Specific expertise: experience of two major development projects using SSD.

BIS7 - Senior consultant-Birmingham. Used as a domain expert.
Specific expertise: extensive experience of teaching SSD on BIS courses.

BIS8 - Principal consultant. Head of *MODUS* training division-London. Used in evaluation of *Advisor*.
Specific expertise: Use of MODUS on development projects and tutoring on MODUS training courses.

BIS9 - Senior consultant-Birmingham. Used as a domain expert.
Specific expertise: training in SSD; extensive knowledge of DBS design.

BIS Banking Systems Ltd.

BIS Banking Systems Ltd. (BISBK) are based in Wimbledon, London. They are part of the BIS Group but trade separately from BIS Applied Systems Ltd. BISBK develop and market large-scale software systems (for an *IBM System 38* environment) used by major international banks for their financial administration systems. The software, known as the *MIDAS* system, is highly complex and involves some 87 million lines of code. *MIDAS* has the largest share of the world market in this type of software.

In 1986 BISBK began a thirty man-year development project to build a new module for the *MIDAS* system dealing with futures and options trading (FOT). A senior management decision had been taken to adopt SSD standards for the project and to use BIS/IPSE. The project team had experience in the DP industry as analysts and programmers but none of them had used SSD. The team were inducted into the SSD method using the standard one week BIS training course.

The FOT team represented a typical group of technically skilled DP practitioners, who were inexperienced users of SSD and who did not have access, within their installation, to designers with expert knowledge of SSD, who could help them during the difficult early stages of using the method. The Head of Development at BISBK is particularly interested in the area of KBSs and their potential use in DPSD and he gave permission for the FOT team to be involved in several aspects of the *Intellipse* project.

BSC

The DP installation at BSC, Port Talbot, is large by UK standards and consists of an IBM 3090 mainframe computer with several hundred megabytes of on-line disk storage. The development environment is based on the Cincom Systems Inc. range of DBMSs, *Total*, *Tis* and *Supra*, and the Cincom Fourth Generation language (4GL) *Mantis*. Some of the older applications at BSC are written in Cobol.

BSC1 - Project manager. Member of the project team.
BSC2 - Project manager. Member of the project team.
BSC3 - Database Administrator (DBA) of the BSC installation.

ICI plc - CMS Division

The Corporate Management Services (CMS) division of ICI is responsible for advising ICI's ten divisions on various aspects of information technology (IT). The deputy head of CMS (ICI1) had taken the initiative in contacting the *Intellipse* project after studying the full list of Alvey SE project summaries. ICI1 was critical of the Alvey SE programme believing that few projects addressed current problems within the DP function of companies like ICI. The *Intellipse* project had been selected as one of only three Alvey SE projects which had any potential relevance to ICI's corporate DP activities. ICI have their own DPSD methods and standards which are similar to SSD. The CMS division was used during the initial feasibility stage and during the evaluation of *Advisor*.

ICI1 - Deputy Head of the CMS division. Responsible for facilitating the adoption of DP standards throughout ICI's ten divisions.
ICI2 - Senior manager. Responsible for evaluating CASE tools and standards for potential use within ICI.
ICI3 - Senior manager. Responsible for advising ICI's divisions about methods and standards to be used for DPSD.

National Computing Centre (NCC) - Software Tools Demonstration Centre (STDC)

The NCC/STDC was set up with Department of Trade and Industry funding and is partly sponsored by the Alvey Directorate. The objective of the STDC is to encourage the use of appropriate software tools, and to promote awareness and understanding of SE

methods. The STDC has assembled an extensive range of CASE tools which are available for public evaluation and training. The STDC also offers consultancy services in the area of CASE tools and SE methods. The STDC was used to evaluate some current CASE tools during the initial feasibility stage.

BIS Training Course Attendees

Attendees on two of BIS's public training courses in SSD and database systems design (DBS) were interviewed informally at different times during the project.

Miscellaneous

A significant number of people have been interviewed informally during seminars and conferences which the author attended, particularly during the first 15 months of the project. Information was also obtained from the project's Alvey monitoring officer - ALV1.

4.3 The Initial Feasibility Study

It should be noted that the term *feasibility* used in the thesis is not intended to encompass the idea of a *cost-benefit* analysis, which is the objective normally associated with this term in commercial DP.

It was noted in 2.2.2 that *MODUS* did not have procedures or standards available, at the start of the project, appropriate to KBS development. In addition, there were no detailed specifications for the proposed KBS tools. As was to be expected in a *research* project, the project proposal simply stated some general aims and objectives. This meant that progress at the start was hesitant, particularly at BIS.

Thus, the approach adopted at the start of the project was to examine in more detail the project proposals, and to establish from SSD users firm evidence of their experience of using the method as well as their expectations of an SSD support tool. In addition, a feasibility study and user-requirements analysis was broadly in line with the first two phases in *MODUS's* conventional development standards.

Two main options were considered for the study:

- a questionnaire to be sent to SSD users and other users of similar SD methods and,
- a series of structured interviews with a specific group of individuals and organisations.

The first option was discounted for several reasons:

- BIS were not happy about issuing a list of SSD clients;
- many DP installations are reluctant to admit to the use of a particular methodology, and are even more reluctant to admit to the non-use of any development methodology;
- a questionnaire was unlikely to reveal the kind of detailed information which the project required;
- it would be difficult to identify only installations using structured design methods, which meant that the questionnaire would have to be sent very widely;
- responses to questionnaires in the DP industry are traditionally poor, and BIS's involvement in the project was likely to deter respondents who were sensitive about information relating to their commercial operations;
- conducting a major questionnaire would be costly in resources.

Structured interviews had three important advantages:

- the exercise could be targeted precisely at organisations known to be willing to co-operate;
- more detailed information could be obtained revealing some of the subtleties of using SSD in a live DP environment and;
- the personal contact implicit in interviewing could establish the basis for using the interviewees later in the project to evaluate any tools and ideas developed.

The last point was particularly important since the credibility of the results obtained during the *Intellipse* project would be enhanced, if organisations who were not part of the immediate project team were involved in validating the research.

An attempt was made to solicit information by publishing an article in the STDC's newsletter. The latter had a wide circulation especially amongst the Alvey community. The article invited organisations to contact the author if they were willing to co-operate in the study. This general call for information received a very poor response partly justifying the validity of the decision not to use a questionnaire.

The initial feasibility study involved the following individuals and organisations:

- BIS1, BIS2;
- BSC1, BSC2;
- ICI1, ICI2, ICI3;
- STDC and;
- ALV1.

The initial meetings were followed up by further interviews after Advisor was built. These interviews are discussed in chapter six.

The interviews were designed to elicit information in the two general areas summarised below. If the interviewees were not familiar with the project or the Alvey programme, a short introduction to the main aims and objectives of the project was given.

- The problems of using structured design, in general, and SSD in particular, in a commercial DP environment.

- The advantages and shortcomings of current CASE tools, and the type of facilities that were being sought in the next generation of knowledge-based tools.

4.4 Conclusions of the Initial Feasibility Study

This first set of conclusions are related specifically to SSD although some of the problems identified are endemic to most forms of structured design.

i The introduction of SSD can slow down to a significant extent the design phase of the software life-cycle. Commercial pressures on DP departments to deliver systems quickly often lead to the abandonment of SSD in favour of ad hoc approaches.

The long-term benefits of SSD, of reliability and maintainability, are lost in order to gain the short-term advantage of early delivery.

ii SSD requires the production of large volumes of support documentation which are difficult and tedious to manage manually, but which are central to the SSD method.

iii Naive SSD users find that the step from the training course (or manual) to the application of SSD to a project in their own environment is a difficult one, sometimes resulting in partial reversion to inferior, ad hoc methods.

iv Maintaining the standards and disciplined approach of SSD is very much dependent on the expertise of the designer. Current support tools rely almost entirely on this expertise and on the designer's knowledge of the development methodology. The tools lack any intelligence and can do little more than provide *mechanical* support by recording information, helping with word and diagram processing and carrying out simple consistency checking. The shortage of expert support for the inexperienced designers using SSD leads to a reduction in the standard of results obtained.

v The existing systems and methods of a DP department often represent many man-years of investment. The introduction of SSD, and CASE tools, must take account of this investment and allow essential parts of the old systems and methods to be incorporated into the new development environment.

vi Support tools for a particular methodology should have available examples of how the methodology can be used to solve typical design problems. This facility should be linked to a system for recording past design problems and their respective solutions.

Some of these findings are corroborated by three reported surveys into the use of

structured methods in commercial DP. Lee[LEE86a), (LEE86b] carried out an independent review of industrial and commercial DP organisations in the UK under the auspices of the City University. She obtained replies from 111 organisations. Some of the respondents were the subject of in-depth interviews. Her main observation was:

> "...how few organisations-only 32% of the sample-were using structured methods for systems development, although a further 15% of data processing managers said they were considering adopting one in the near future." (Page 30 - LEE86a.)

Lee quoted some of the respondents' views on the use of structured methods:

> "Project Manager:
> 'One problem is convincing management that the change of emphasis towards design is both necessary and cost effective. (We suffer from the Wisca syndrome - why isn't Sam coding anything?)'
> Systems Development Manager:
> 'Major benefits derived during structured analysis but found to be cumbersome at the design stage, particularly generating masses of paperwork.' " (Page 29-LEE86b.)

Sumner and Sitek[SUM86] investigated whether structured methods for systems analysis and design were being used in the USA. They sent questionnaires to 172 organisations involved in a mixture of "real-time" and "business applications" - only 47 (27.4%) replied. They concluded:

> "One of the major findings of our study is that although most of the respondents acknowledged the benefits of using structured tools in requirements analysis and design, these tools were not being used in actual development projects, largely because of their lack of acceptance by data processing professionals, and the fact that they were perceived as time-consuming to use...Project managers may resist their use because they are being pressured to complete projects within time and cost constraints imposed by users and feel that the use of structured tools would only add to project costs.
>
> Many of the programmers and analysts may not have had formal education in information systems design and may have learned development practices from project managers and peers-on-the-job. As a result they may be unfamiliar with structured approaches and how to use them." (page 23.)

Condon[CON88] analysed the NCC's 1987 annual survey of its members. The survey covered many aspects of DP and had 365 respondents. Condon made these comments in relation to the use of structured methods by NCC members:

> "In the section..[of the survey]..on systems development methods, a third of the respondents said they were using some form of development methodology-a disappointingly low level, although two-thirds of all new business applications

> involve some defined method...The survey found some predominant obstacles in the use of tools. 'The most significant one is finding the tools with the right facilities for the various stages in the development life-cycle.' " (Page 13.)

Some general conclusions were also drawn from the results of the feasibility study.

System failure

The problems of building DP systems on time, within budget and meeting end-user requirements are well documented. A frequently quoted statistic is that 50-80% of the budget for a systems development project is spent "maintaining" the system after it has been delivered to the user[(ALVEY82), (RAMA84)]. However, despite its reputation for failure, the DP industry can point to a number of highly successful systems in widespread, everyday use. The international airline booking system or the many plastic card operations are two highly visible examples. The existence of high quality DP systems is an important observation, since any attempt to build knowledge-based tools for supporting DPSD must be predicated on the assumption that techniques exist which can adequately cope with the problems of DPSD, and that people exist who know how to apply those techniques successfully.

End-user involvement

Another problem in commercial DP is the large backlog of development work in many DP departments encouraging the DP industry to look for ways to improve analyst and programmer productivity. This fact, together with the need for DP managers to deliver systems which satisfy end-user expectations, has led to suggestions that end-users should play a more active role in information systems development[(MUM87), (CORD85)]. If end-users are to play a more active role, development tools are required which do not depend for their success on a high level of design expertise on the part of the user.

Movement of DP personnel

Another long-standing problem in the computer industry, which a knowledge-based approach can address, is the frequent movement of DP personnel between companies. When a person leaves the DP department of one company to join another (s)he takes with him/her not only technical skills but, more importantly, accumulated knowledge of the company's development environment and computer-based information systems. Knowledge-based tools could enable this knowledge to be retained within the company

so that it could be used, for example, to speed up the induction and training of new personnel.

Commonality between systems

Information systems in use in diverse organisations possess a high degree of commonality, especially in relation to their functional specification. A knowledge-based design tool could exploit this commonality by providing the designer with a knowledge base containing generically classified design templates. The template knowledge base could be used as a basis for a new design built up using proven design elements from systems already built and tested.

One further important conclusion was drawn from the feasibility study. Although the DP practitioners interviewed clearly stated their desire for support tools which would address the problems of structured design, none of them offered a coherent view of the nature of the support that such a tool should give. In particular, the concept of a *knowledge-based* tool was an unfamiliar one to those interviewed, and it became clear that it would be desirable to develop some prototype KBS tools as early as possible. These tools could then be used to help clarify, with DP designers, the precise functionality that was required in a knowledge-based support tool.

4.5 Summary of the Research Themes Identified

The essential conclusion from the initial feasibility study was that the successful use of SSD requires a high level of expertise on the part of the SSD practitioner. This expertise was usually attained only after considerable experience of using SSD on actual development projects. Knowledge-based tools designed to support SSD should have, as a central feature, the facility to be able to be used effectively by users who were not necessarily expert SSD practitioners. This meant that the tools would have to encompass the design expertise of expert SSD practitioners.

Current support tools, typified by products like the *BIS/IPSE*, LBMS's *Automate* and James Martin's *IEW* and *IEF* [JONES87b], *do not know how to design.* The ability of these tools to effectively manage the design process, by automating mechanical tasks and facilitating diagram production, word processing and documentation management, is dependent on the expertise possessed by their user. These tools, for example, do not guide the user through the design process but expect the designer to know which tasks to

perform, how to perform them, and the order in which they should be carried out.

The areas of research which were important to the *Intellipse* project were clearly identified by the initial feasibility study. Research was necessary to establish whether knowledge-based tools could be designed which would exhibit some of the following features:

- support for a particular structured design methodology, SSD, which did not necessarily require the user to be an expert in that method for the support to be effective;

- the ability to provide expert support for specific design tasks within the overall development life-cycle;

- knowledge about design procedures and past design practices which could be made available to the user;

- the usability, robustness and reliability demanded in an industrial or commercial environment;

- the ability to integrate successfully with current support tools, like BIS/IPSE, and existing DP development practices.

Chapter Five

The Concept and Design of *Intellipse*

Mephistopheles
What need, dear Sir, this dull life to pursue?
One loses pleasure in the same old view.
It's good no doubt
To try things out;
Then off we go to something really new.

Johann Wolfgang Goethe 1749-1832 (Faust/Part One)

Preamble

This chapter describes the initial development phases for *Intellipse* and introduces the concept and architecture of the system. A broad functional specification is given and the key features of the design are explained. An initial appraisal of the suitability of the DPSD domain for KBSs is also made.

5.1 Initial Development Phases

After considering the results of the initial feasibility study and the outline specification of the proposed tools in the project proposal to the Alvey Directorate (descibed in 2.1), five initial phases of development were identified in order to investigate the potential for building a knowledge-based support tool which would exhibit the features listed in 4.5.

i The definition of a functional specification for the system.

ii The establishment of a unified system architecture possessing a high degree of modularity and allowing for incremental development over a period of time.

iii The identification and classification of the individual tasks performed within SSD and their precise relationship within the overall SSD method.

iv The adoption of a suitable, umbrella knowledge representation

schema for the system.

v The development of a structured knowledge engineering approach able to cope with the demands of a very wide application domain.

The remaining sections will expand on each of these phases and explain why each of them was identified as necessary.

5.2 Functional Specification

A functional specification was required which would provide a framework for developing some prototype tools. This specification was not intended to be a detailed list of functions or actions which the system should perform in precisely defined circumstances. This type of specification is the necessary basis for the development of a conventional DP system, but not for one whose mode of operation was still undetermined. The *informal* functional specification devised was for a system which could:

i advise a designer about the SSD method in general, by answering specific questions about *objects* and *activities* which are relevant in the domain (Advisor-mode);

ii offer *active* support to the designer in the form of intelligent advice about a specific task within design. This advice should actively assist the designer to make design decisions and perform specific design tasks (Designer-mode);

iii where appropriate, automatically execute tasks within the design phase of SSD which are currently executed manually;

iv build and maintain a unified project database and data dictionary which could be linked to any existing systems and proprietary data dictionaries appropriate to a particular DP environment.

5.2.1 Objects and Activities

SSD consists of a number of tasks or *activities* to be performed in a certain order, and

the SSD standards precisely specify the pre-requisites and the output for each activity. The output from an SSD activity is usually an item of documentation, an *object*, which must conform to a pre-defined structure and content. It was decided, therefore, to classify all "things" in the SSD domain as either *activities* to be performed, or *objects* which were the subject or focus of a particular activity.

A general example of this classification would be that *cooking* is an activity and *ingredients* and *meal* are the related objects. In the context of SSD, Figure V.i illustrates some examples of objects and activities.

ACTIVITIES	OBJECTS
DATA ANALYSIS	DATA, PRIME KEY
STRUCTURE PROCESSES	PROCESS SHEET
IDENTIFY BASIC-LEVEL ACTIVITIES	ACTIVITY, ACTIVITY SHEET

Figure V.i - Examples of Activities and Objects

5.2.2 Advisor/Designer Concept

Two basic modes of operation were devised for *Intellipse*: Advisor-mode and Designer-mode - hereafter referred to as Advisor and Designer. Related to this concept is the idea of *passive* and *active* support. Four potential forms of support were identified for a tool intended to help a designer perform a task in SSD:

- remove the decision-making from the designer by completely automating the task - AUTOMATION;

- share decision-making with the designer by using the tool to offer advice about procedures and techniques, *and their specific application to a design problem being considered,* but leaving the ultimate design decision to the designer - ACTIVE SUPPORT;

- general advice about procedures and techniques for a specific task *which does not offer specific advice in relation to a particular design problem* - PASSIVE SUPPORT.

- facilities for *recording decisions* made by the designer and for aiding diagram and word processing, but excluding any advice (general or specific) on design procedures or techniques - MECHANICAL SUPPORT.

Most current tools, such as the BIS/IPSE, fall into the last category. The first three forms of support are related to the KBS roles of *automaton*, *expert consultant* and *tutor* respectively, identified in 3.1.3.

Advisor

Advisor is a passive, knowledge-based, question-answering system dealing with the SSD domain. It is *knowledge-based* because it contains knowledge about the design process which reflects the expertise of expert human designers not to be found in the SSD manual or standards. It is *passive* since it offers advice about SSD in general, and not in relation to a specific application. It is a *question-answering* system because its mode of operation is intended to model the actions of a BIS/SSD consultant answering general questions about SSD for inexperienced SSD designers. Advisor has three basic components:

- a set of knowledge bases containing facts, rules, procedures and techniques relating to the objects and activities in SSD;
- a knowledge base representing the hierarchical relationship of all the activities within SSD;
- a mechanism for navigating the SSD domain.

In Advisor, the designer is able to interrogate the knowledge bases about any of the objects or activities in the SSD domain, but Advisor cannot actively execute, in relation to a specific application, any of the design tasks which it knows about. However, the Advisor knowledge bases could act as an explanation facility while the user is in Designer-mode. A detailed description of Advisor is given in chapter six.

Designer

The objective of Designer is to provide *active* support for the designer carrying out tasks in SSD. Designer should not only be able to execute some design tasks itself but, where the latter is infeasible, it must provide sufficient advice to allow the human designer to make appropriate design decisions. That is, Designer will enable the user to work on a specific design problem by actively invoking the rules and procedures which are passively described in the Advisor knowledge bases.

Where current tools simply *record* the decisions taken by the designer, or facilitate the mechanical tasks of diagram and document production, Designer should contribute its knowledge of the design process, thereby actively promoting good design on the part of the human designer. The Designer system is described fully in chapter seven.

5.3 *Intellipse* Architecture and Knowledge Bases

The architecture of the *Intellipse* system is illustrated in Figures V.ii and V.iii.

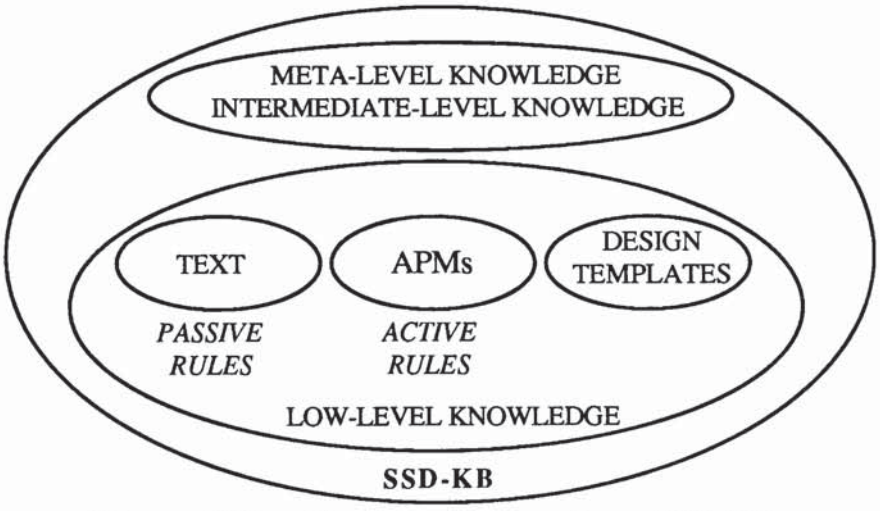

Figure V.ii - Exploded View of an SSD-KB

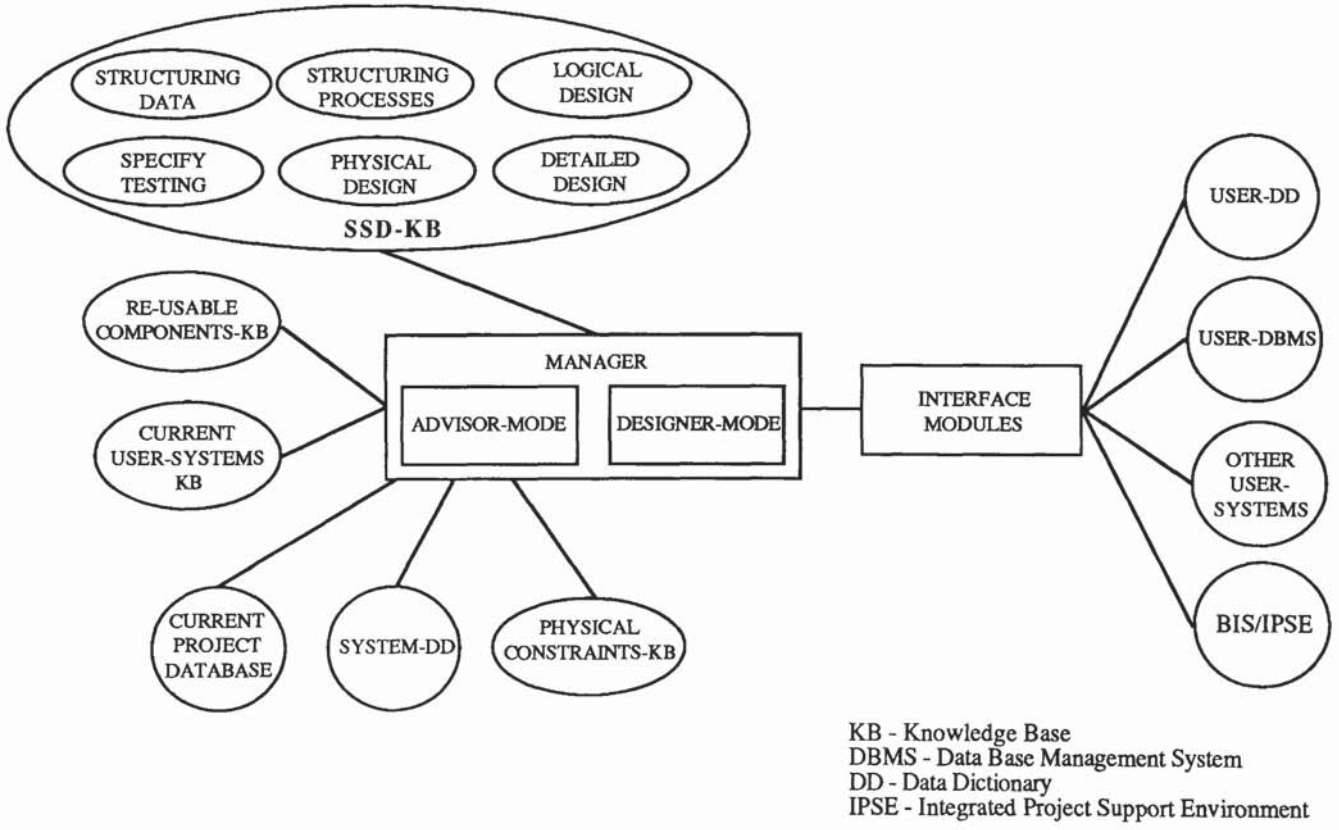

Figure V.iii - Schematic View of the *Intellipse* Architecture

5.3.1 Knowledge Bases

The SSD domain was divided into six separate knowledge bases. This was done for two reasons:

- it corresponded with the six phases of design identified by SSD;

- it was expected that when Advisor came to be implemented, the separation of the knowledge bases would make it easier to fit the system into a personal computer (PC) environment.

The last point indicates that decisions about physical implementation of the *Intellipse* tools had been taken early in the project. This issue is discussed in chapter six.

Each of the knowledge bases is made up of several distinct components.

- Meta-level knowledge describing the objects and activities within a domain and their inter-relationships;

- Intermediate-level knowledge describing the types and categories of knowledge recorded for a specific object or activity;

- Low-level knowledge corresponding to the facts, heuristics, procedures and techniques for a given topic, stored in the system as text or Activity Program Modules (APMs);

- Partially instantiated design templates appropriate for specific tasks within SSD and based on past applications, or the design experience of BIS experts. This KB exploits the commonality which exists between the designs of systems built for different application areas but whose functionality is similar.

The other knowledge bases which the system will require are the Current Systems KB, the Physical Constraints KB and the Re-usable Components KB. The first two will contain knowledge about the particular DP environment in which *Intellipse* is installed. This would include data about the storage and processing constraints which the design had to meet. The Re-usable Components KB, which would be linked directly to the

Current Systems KB, would contain knowledge about programs and other modules available to the designer in his/her installation.

The knowledge bases identified in the last paragraph are one of the key features of the *Intellipse* concept. It is essential that KBS design-support tools, intended for use in a commercial DP environment, have the facility to take account of pragmatic issues, such as available disk storage and performance constraints, when making design recommendations. Many of the AI environments discussed in chapter three emphasise the role of *abstract design* knowledge, and appear to give relatively little attention to *concrete target environment* and *application domain* knowledge.

5.3.2 Activity Program Modules

The architecture of *Intellipse* has been designed with the maximum degree of modularity allowing development of the system to take place in an incremental fashion. Incremental development ensures that discrete elements of the system can be prototyped, evaluated and implemented without waiting for the system as a whole to be completed. In addition, the modular architecture enables individual components of *Intellipse's* knowledge bases to be revised as developments take place in the SSD method.

Central to the modular architecture are the Activity Program Modules (APMs). The APMs will be executable modules which, together with the meta-level knowledge base, will govern the operation of *Intellipse* in Designer-mode. They will be either KBS modules embodying the facts and rules governing a specific design task, or algorithmic programs, if the particular task can be automated. An APM may also be an existing tool such as an editor or compiler which could be invoked by the system.

The integration of KBS and non-KBS modules is another key feature of the *Intellipse* design. This concept makes explicit the need for support tools in complex domains like DPSD, to encompass both heuristic and conventional approaches under one umbrella. In 3.1.4 it was noted that this type of integration is an increasingly important aspect of *operational* KBSs.

The ability to develop *Intellipse* incrementally is crucial. Section 5.4 will indicate that SSD is a very wide domain containing a large number of distinct tasks and the *Intellipse* concept envisages an integrated system of KBS tools capable of supporting a large number of these tasks. The integration of the system will be through the exchange of

data between different tools and by managing the tools under one umbrella environment. Due to the large number of tasks in SSD, the construction of such a system is likely to be unmanageable unless a rigorous *logical* separation between modules is maintained.

Integration of the individual APMs could also be achieved through the use of a project database and data dictionary which would contain details of all design decisions made during a particular project. This concept is used in the BIS/IPSE which generates a *system model* for each project being supported by the system.

5.3.3 External Links

All of the DP practitioners interviewed during the initial feasibility study stressed the importance of support tools being able to link easily to existing systems and target environments. A *target environment* is the hardware system that will run a particular application. The term is used to distinguish it from the *development environment* which can be a separate system used to build the software. There are three main reasons why these external links are important.

- Most proprietary data dictionaries and similar systems used by installations to store data about their system designs and databases run in a mainframe environment. CASE tools, on the other hand, usually run in a PC or minicomputer environment, although these tools can often produce files of data in the appropriate mainframe format. If these files cannot be transferred directly to the target, manual keying of the data is required which can be a very time-consuming and error-prone exercise.

- Certain tasks in SSD may require the exchange of data between the target environment and specific *Intellipse* tools. It is also important that this can be done automatically to avoid manual keying of data.

- The BIS/IPSE provides mechanical support for several tasks in SSD. It is sensible therefore for *Intellipse* to supplement these IPSE facilities with active tools, rather than re-implementing the IPSE support and then adding on active tools. This has been a crucial issue during the project and is discussed in greater detail in

chapters six, seven and ten.

5.4 Is the DPSD Domain Suitable for KBSs?

It is useful at this stage to make some initial observations on the suitability of the SSD domain for KBS support, before going into the details of how the proposed *Intellipse* architecture was validated. In 3.2.1 a list of factors is identified which should be considered when assessing the suitability of a problem for a KBS approach. In this section these factors are considered in relation to the SSD domain *as a whole*. In chapter seven, which deals with the Designer feasibility study, *individual* SSD tasks are considered.

Narrowness of the domain: SSD is a very wide domain and many of the tasks performed early in the design process involve significant general knowledge about the application environment. The width of the domain is evidenced in the first two SSD phases, *structuring data* and *structuring processes*, which contain over thirty different tasks. The width of SSD is a key reason why a structured approach to knowledge engineering is required.

Complexity of the problem: SSD is a highly complex domain involving large quantities of data. However, the structured nature of the method means that the tasks are performed in a highly ordered manner. This natural structure can be exploited when defining problem-solving strategies in the domain.

Nature of the problem: SSD tasks are cognitive in nature and do not require visual or sensory skills. Many of the tasks are executed manually using a pencil and paper technique.

Nature of the experts: In principle, BIS were prepared to make domain experts available who were not necessarily part of the immediate project team. However, this political support for the project had to be balanced against the client work-load at any particular time. This indicated that the involvement of individual BIS consultants would have to be planned well in advance and knowledge engineering sessions organised for maximum efficiency. Since BIS consultants spend a great deal of time working with clients who are having problems with SSD or who require tuition, it was likely that their ability to articulate knowledge would be good.

Training: BIS have well-established training courses for SSD. The courses make extensive use of *syndicates*, where small groups of course attendees work on case studies illustrating various SSD techniques. This suggests that SSD has evolved well-established approaches to specific design problems and this is likely to make knowledge elicitation easier.

Speed of solution: None of the SSD tasks have to be solved in real-time and, although some tasks are dependent on data from other tasks, there are no specific time constraints for individual tasks.

Sensitivity of the problem: Legal and ethical issues are not relevant in SSD.

Conventional solutions: The existing mechanical support for some aspects of SSD can be regarded as conventional solutions to specific SSD problems. The research objectives of the project dictated that KBS approaches should be the focus of attention. The important issue for the *Intellipse* work was to identify which tasks could be, or were already, supported algorithmically, and which ones needed KBS support - and to integrate both types of support into a single environment.

Written material: The training courses and standards for SSD means that a large amount of written material is available on many aspects of SSD. The availablility of written material enables the knowledge engineers to do their homework prior to the initial KE sessions. This can greatly enhance the credibility of the knowledge engineers in the eyes of the experts, who do not then feel as if they were talking to people ignorant of their field of expertise. From the knowledge engineers' point of view, the expert does not appear to be talking a foreign language. Thus the written material can speed up significantly the early stages of knowledge elicitation.

The main disadvantages of SSD with respect to KBS support appeared at this stage to be:

- the width and complexity of the domain;
- the extensive use made of general knowledge;
- the uncertainty regarding the availability of domain experts.

Advantages noted at this stage were that:

- SSD is a highly structured methodology;
- the domain experts are likely to be articulate;
- a large quantity of training/written material is available.

The most difficult potential problem seemed to be the use of common sense knowledge. In the DPSD domain *common sense* can be taken to mean general knowledge about different application domains (finance, manufacturing, banking, insurance etc.) or different target environments (IBM, Honeywell, DEC etc.). Only a closer examination of individual SSD tasks could resolve this question.

5.5 Summary

This chapter has proposed a conceptual design and architecture for a support environment for SSD, based on the integration of KBS and non-KBS tools. The proposed design is intended to provide a basis for building tools which will exhibit the features identified in 4.5, and meet the broad functional specification in 5.2. Seven basic questions to be addressed in order to validate the *Intellipse* concept are:

- Can the advisor-mode system be built and will it meet the objectives set for it?
  (*Advisor* - chapter six.)

- Which tasks in SSD are suitable for KBS tool support?
  (*Designer feasibility study* - chapter seven.)

- Can a structured development method for KBS construction be devised which can cope with the complexity of the SSD domain, and can such a method be an adaptation of existing conventional methods?
  (*POLITE methodology* - chapter eight.)

- Can knowledge-based APMs be constructed which will help a human designer perform tasks in SSD?
  (*ITAM* - chapter nine.)

- Can the system be built in the incremental fashion proposed?

- Can the APMs be integrated under a single umbrella environment?

- Can the tools be linked easily with external systems?

The last three questions are discussed in chapters six, seven and ten.

These questions motivated the future direction of the work and to a large extent dictated the next steps in the project. Figure V.iv illustrates the sequence of work and underpins the structure of the rest of the thesis.

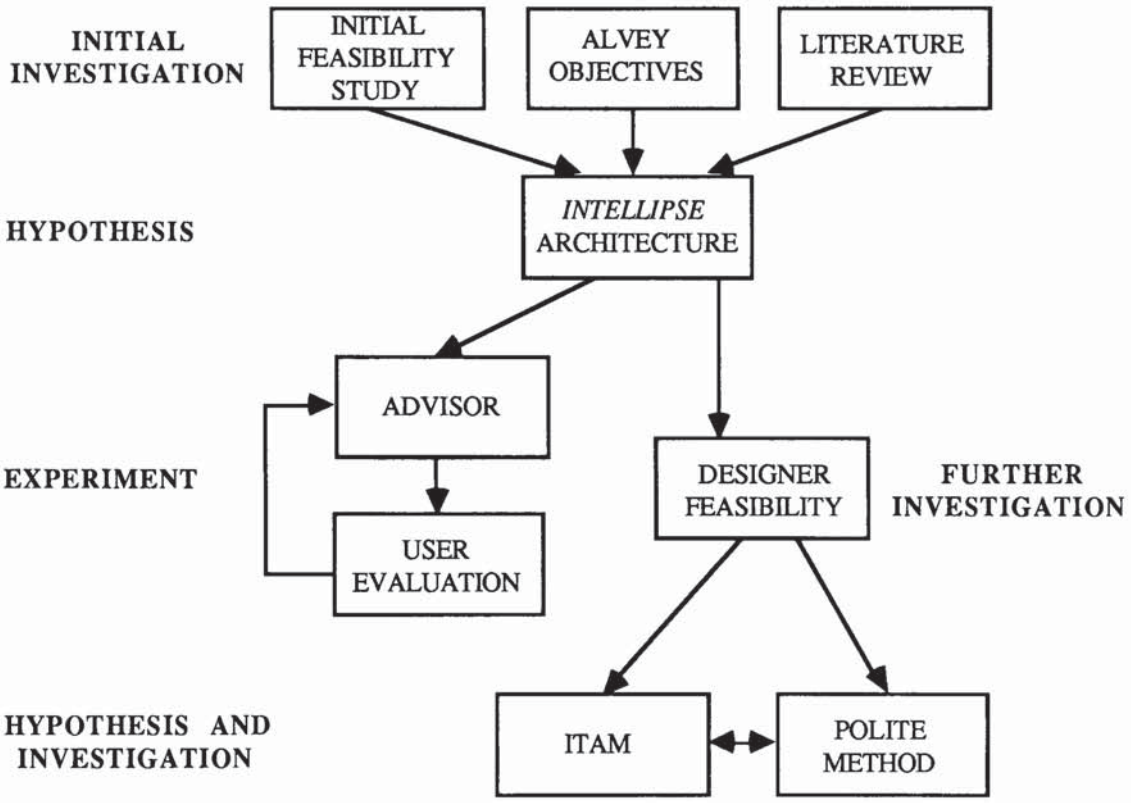

Figure V.iv - Logical Progression of Project Stages

Figure V.iv is not meant to imply that the project work followed a formal *scientific method.* The latter approach would not have been valid since the absence of metrics in the domain precluded accurate quantitative measurement. However, the diagram does indicate that the project exhibited a clear logical progression from one stage to another.

Chapter Six

Advisor

The Nightingale and the Lark
What should we say to the poets who take flights beyond the understanding of their readers?

Nothing but what the Nightingale said one day to the lark. "Do you soar so very high, my friend, in order that you may not be heard?"

Gotthold Ephraim Lessing 1729-1781

Preamble

This chapter describes the design and construction of Advisor. It discusses why Advisor was built first and how Advisor influenced later work on the project. It describes the knowledge engineering methodology adopted at this stage, and the reasons for choosing the tools used to implement the system.

6.1 Why Was Advisor Built First?

At this stage in the project, a significant amount of feasibility work had been done. However, the project still lacked a focal point and it was important to identify some concrete proposals around which the next stage of the project could be organised. There were several reasons why Advisor was chosen as this focal point:

- Of the aspects of *Intellipse* defined in 5.2, Advisor appeared to be the easiest part to build.
- The nature of the proposed Advisor system indicated that it would be necessary to rigorously structure the SSD domain and to identify all its objects and activities. This preparatory work would be useful in familiarising the project team with the domain, and would help to clarify ambiguities and gaps in SSD which were already evident.
- The structuring of the SSD domain would also help identify the different terms and concepts used in the domain.
- Having the Advisor system available would provide an initial basis for the knowledge engineering sessions with BIS experts. It could

be used to display a first pass version of the knowledge bases to the experts, prompting them to suggest amendments and changes.

- Advisor would also help in further interviews with DP practitioners, since it would provide a more concrete basis for them to make suggestions about features they thought desirable in KBS tools for SSD.
- Apart from its role within *Intellipse*, it was felt, particularly at BIS, that Advisor would be a useful tool in its own right - as a stand-alone system. BIS wanted a tool which could supplement the one week training courses in SSD by acting as a mechanised consultant for newcomers to the methodology - see 6.2.
- It was felt that the existence of Advisor was an essential precursor to building the Designer system, as it would identify precisely the tasks which Designer would have to support, as well as providing an initial description of how these tasks were performed. It could also be used in knowledge engineering sessions for the Designer feasibility study.

6.2 How Was the Advisor Project Conducted?

Although some initial attempts were made to conduct the Advisor project according to a conventional software development model, this was found to be inappropriate. This was mainly because the Advisor work was of a prototyping nature and very much therefore an exploratory exercise. In addition, no *detailed* requirements for a system existed which could form the basis for a conventional life-cycle approach. Many of the decisions on what steps should be taken next in developing the system were made at the end of a particular stage, and were based on a subjective assessment of what seemed appropriate at the time. With hindsight the following phases of development can be identified:

- definition of an informal functional specification (5.2.2);
- identification of the intended users;
- definition of a knowledge representation schema;
- definition of a detailed mode of operation;
- definition of an informal specification for a Knowledge Acquisition Module (KAM);
- production of program specifications for Advisor and the KAM;
- knowledge engineering for the Advisor KBs;
- coding of Advisor and the KAM;

(the last two phases were done concurrently)

- instantiation of Advisor KBs using the KAM;
- initial testing and user evaluation;
- re-design of the Advisor user-interface;
- further testing and user-evaluation;
- further instantiation of KBs;
- evaluation of Advisor KBs by BIS experts;
- further re-design of the Advisor user-interface.

6.3 Knowledge Representation Schema

6.3.1 Who were the Intended Users of Advisor?

Advisor was intended to be a more accessible version of the SSD training manual as well as presenting SSD in a more structured and logical manner than the manual. It was to contain knowledge of procedures and techniques which could not be found in the manual, or in any other documentary form. The manual is designed to support the training course and is organised to reflect the way the material is presented on the course. It does not, for example, contain an index. Many attendees of the courses complain that, while the manual is a very good support for the week-long course, it is a very unsatisfactory guide or tutor, once the course is over and the new SSD practitioner is back in his/her own environment facing real design problems. It followed from this that two basic categories of potential user for Advisor were important:

- *the naive SSD user* - course attendees whose only practical experience of the method is the syndicate work during the training course, and whose only support after the course is the SSD manual;
- *the rusty SSD user* - someone who has attended an SSD course in the past and has significant practical experience gained on actual development projects, but who has not used the method for some time and needs reminding of some of the techniques.

6.3.2 What Kind of Information Do the Users of Advisor Require?

It was thought by BIS that some of the difficulties users experienced when first using SSD arose because they failed to appreciate the context within which many of the SSD tasks were performed. This failure was partly the fault of the manual which often did not justify the necessity for a particular task or method. The manual also obscures the

hierarchical relationship of the SSD tasks, and it is difficult to appreciate the position of a particular task within the overall design method. It was therefore decided to classify knowledge about tasks in SSD into five categories.

Descriptive	*what* are the tasks that must be performed?
Justificational	*why* is the task important?
Conditional	*when* should the task be performed and what is its relationship to other tasks?
Procedural	*how* is the task performed? - procedures and techniques.
Illustrative	*example* of the techniques used for the task applied to a particular case.

A similar classification is used for objects in SSD, except in this case only the *what* and *example* categories are normally employed. Examples of this classification applied to objects and activities in SSD is given in section 6.6.

6.3.3 Frame-Like Representation Schema

Minsky[MIN75] proposed a theory of knowledge representation based on the concept of *frames*. The theory was principally concerned with AI systems for the analysis of visual scenes. Minsky defined frames in the following way:

> "A *frame* is a data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party. Attached to each frame are several kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed.
>
> We can think of a frame as a network of nodes and relations. The 'top levels' of a frame are fixed, and represent things that are always true about the supposed situation. The lower levels have many *terminals*- 'slots' that must be filled by specific instances or data...Collections of related frames are linked together into *frame systems*." (Page 212.)

Scripts are a more restricted form of the frame concept. Rich[RICH83] described scripts as a "frame-like structure":

> "A *script* is a structure that describes a stereotyped sequence of events in a particular context. A script consists of a set of slots. Associated with each slot may be some information about what kinds of values it may contain, as well as a default value to be used if no other information is available...

> Scripts are useful because, in the real world , there are patterns to the occurrence of events. These patterns arise because of causal relationships between events. Agents will perform one action so that they will then be able to perform another. The events described in a script form a giant causal *chain.*" (Page 235-6.)

SSD is a domain consisting of a large number of "events" (tasks) which follow each other in a strict chronological sequence. Therefore a *frame-like* representation schema similar to the concept of scripts seemed an appropriate choice for representing the knowledge in Advisor. It is frame-like because it exploits some of the features of frames identified by Minsky, although it must be stressed that Advisor is only a *partial* implementation of a full frame system as envisaged by Minsky. In particular, Advisor does not make use of the concept of *inheritance* whereby a frame, which is a 'child' of a frame at a higher level, inherits the properties of the higher level frame.

The use of the term *frame(s)* throughout the rest of this chapter is always qualified by the proviso made in the last paragraph. One of the main reasons for choosing frames was that it was a useful *conceptual* schema for facilitating discussion and communication amongst the project team, during the knowledge engineering for the Advisor system. Other features of frames which seemed appropriate for Advisor and which led to this choice of conceptual schema are summarised in Figure VI.i.

SSD KNOWLEDGE	FRAME-LIKE SCHEMA
hierarchical structure	network of frames
different topic-types	slots
several knowledge types per topic	multiple slot categories
sequential description of procedures	scripts

Figure VI.i - Correspondence Between SSD Knowledge and Frame-Like Representation

Three types of frames were defined. They are related directly to the description of the *Intellipse* KBs given in 5.3.1.

Meta-level frames - "knowledge about knowledge", that is, the topics in SSD about which Advisor contains information.

Slots	Domain:	e.g. structuring data, logical design...
	Topic:	e.g. data analysis, record data...
	Is:	e.g. activity, object...

Intermediate-level frames - knowledge about the sub-activities, related objects and knowledge-types for a particular topic.

Slots	Sub-activities:	e.g. record data, select key...
	Related objects:	e.g. data, prime key...
	Knowledge-Type:	e.g. what, why...

Low-level frames - the text corresponding to a specific knowledge-type for a particular topic.

Slots	Text:	Data analysis/what. Data analysis is a task.

An example of a meta-level and intermediate-level frame is given in Figure VI.ii.

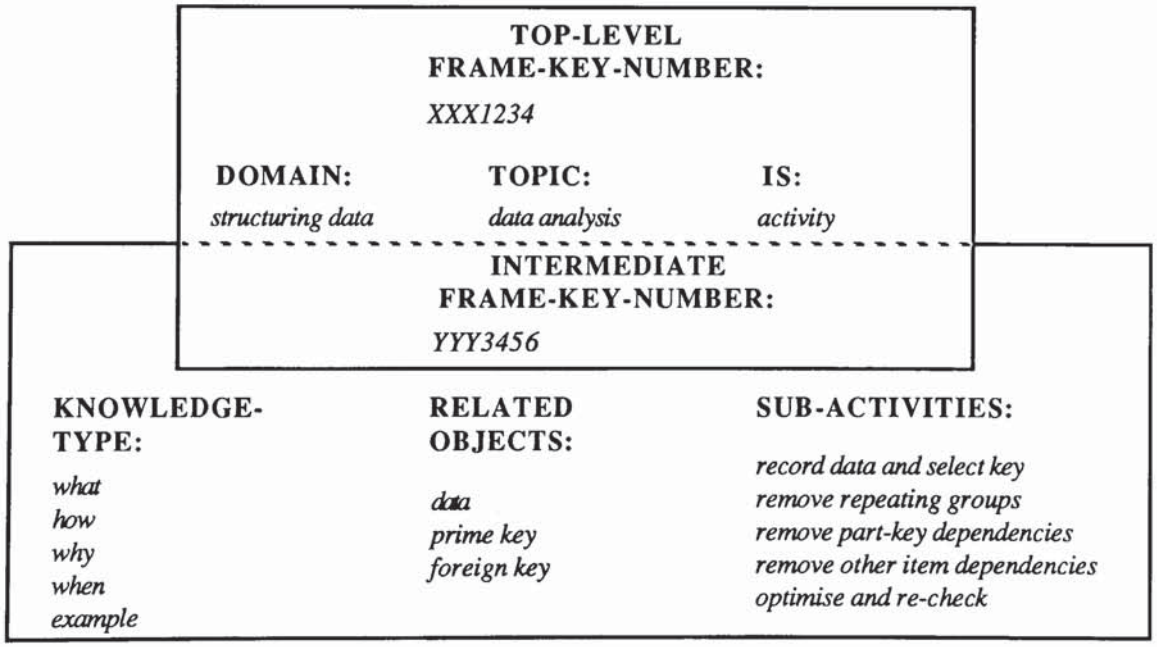

Figure VI.ii - Example of Top-Level and Intermediate-Level Frames Used in the Advisor Knowledge Bases

The frame-key-numbers shown in the diagram are the *physical* means by which the frames were linked together at the implementation stage. They are not an essential element in the *logical* schema and were not therefore listed as slots above. Also, values in these slots are assigned automatically by the knowledge acquisition module (KAM) and cannot be entered by the knowledge engineers (see 6.5). The meta-level and intermediate-level frames together represent the hierarchical relationship between the SSD tasks.

6.4 Advisor - Mode of Operation

After considering the informal specification for Advisor described in 5.2.2, the intended users and the information the latter require, the following list of detailed features was defined:

- the user should be able to select any particular topic (object or activity) in SSD and obtain information about it in conformity with the categories listed in 6.3;
- the user should be able to navigate around the SSD domain and be able to see clearly the relationship between the current selected activity and its parent, child and neighbouring activities;
- the user should be able to step through the SSD activities one at a time if required;
- the system should respond to menu commands, or a highly constrained English-like query language (the *Query Analyser*).

Query Analyser

The objective of the query analyser is to enable the user to enter free-form, English-like queries to interrogate the KBs, allowing the experienced user to bypass the multiple-level menu system. This capability is essential since the Advisor system contains KBs covering over one hundred individual topics - each topic having, potentially, several categories of knowledge. The menu structure reflects the breadth and complexity of these KBs. Although an advantage for the naive user, who can be guided through the domain step by step, using the interactive menu system, this was thought likely to be an inhibiting interface for the experienced user, already familiar with the system or with the SSD method. The query analyser allows the experienced, but rusty, SSD user to bypass the menu system and go immediately to any position within the SSD hierarchy.

Some examples of the free-form queries which can be processed by the Advisor are:

What is a prime key?
How do I perform first normal form analysis?
When do I start the physical design stage?
Show me an example of a transaction profile.

It was found that an effective and practical natural language capability could be achieved without resorting to the construction of a sophisticated set of grammar rules with their accompanying translation and interpretation modules. The latter approach inevitably leads to a severe cost in processing speed and consequent degradation in performance as seen by the user. The analyser built is designed to detect only key words or phrases in the input query and can therefore cope with ungrammatical sentences. This method also means that the user who has become familiar with the system can access the KBs using a fast-path by simply entering queries containing only the appropriate keywords.

The query analyser described above is clearly of a much more limited nature than was envisaged in the original Alvey project proposal. In 2.2.1 it was indicated that even before the *Intellipse* system had been devised, it was thought that a sophisticated natural language capability for the system would be unnecessary. The mode of operation and scope defined for Advisor confirmed these initial views.

6.5 The Knowledge Acquisition Module (KAM)

The Advisor system allows a user to navigate and interrogate the six KBs covering SSD. However, a separate system is required to enable the KBs to be created in the first instance. The KAM is a system for creating the meta-level and intermediate-level frames. It also allows text to be entered into the low-level frames. The KAM automatically assigns frame-key numbers and provides a number of other house-keeping facilities for managing the KBs.

The KAM and Advisor were designed in such a way that very little knowledge specific to SSD is built into the code. This means that, in principle, the KAM and Advisor could be used in domains other than SSD. In this respect Advisor/KAM can be regarded as a "shell" system which could be filled with knowledge about another domain. Of course, the system would only be appropriate for hierarchical domains like SSD, which could be represented using the knowledge representation schema adopted for Advisor/SSD. An application of Advisor to another domain is discussed in 6.10.

6.6 Knowledge Engineering (KE) for Advisor

It was noted in 3.3.2 that knowledge acquisition can be a bottleneck in KBS development and it was thought appropriate that, unlike some of the other Advisor development phases, the KE phase should be structured and planned in advance. The procedure followed is shown in Figure VI.iii.

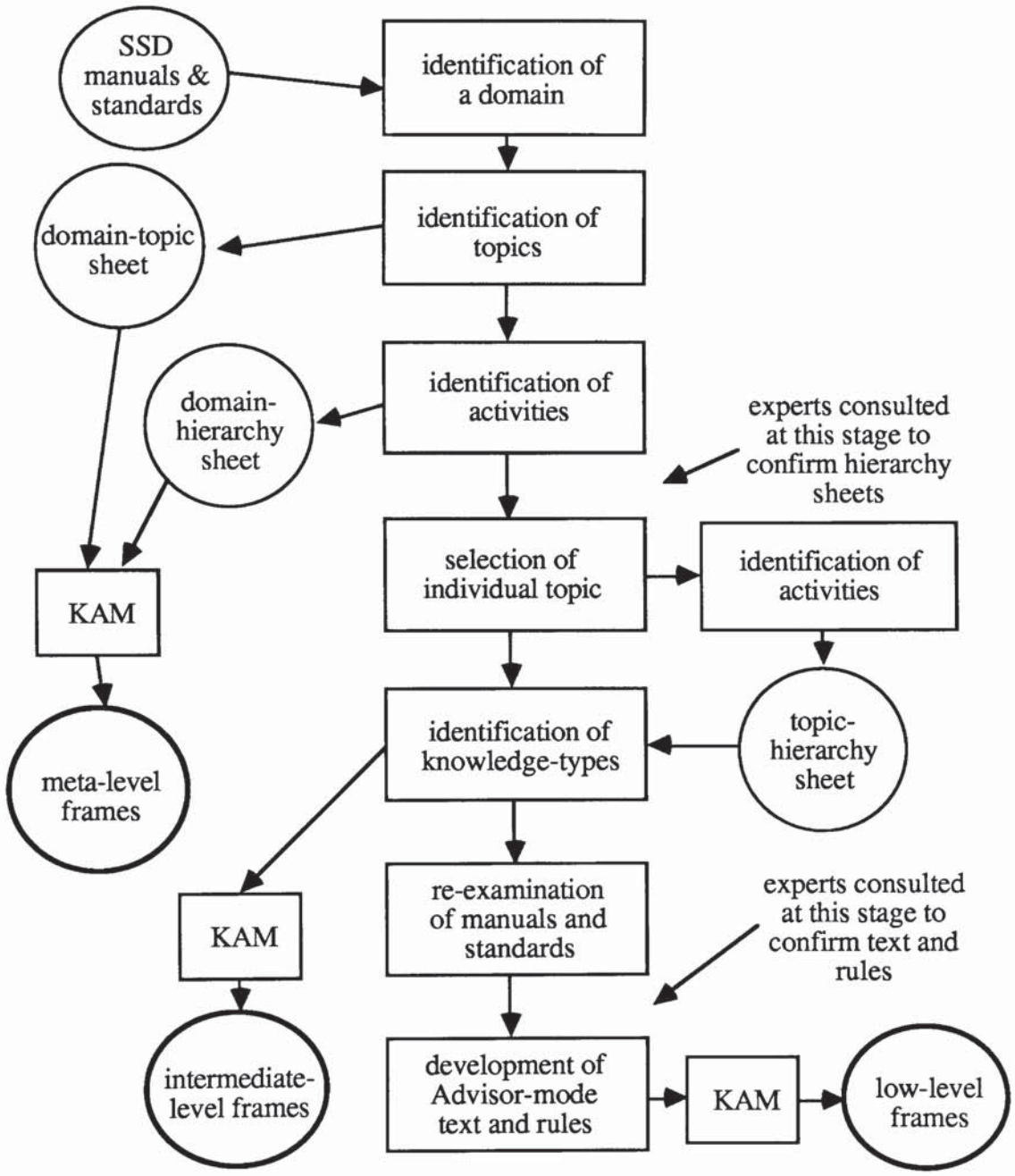

Figure VI.iii - Knowledge Engineering Procedure for Advisor

As Figure VI.iii suggests, a significant amount of the KE for Advisor, especially at the start, was done primarily from documentary sources, without reference to BIS experts. Only after a first-pass version of the KBs was obtained, were experts asked to validate the structure and text and begin the process of enriching the KBs with knowledge not available in the manuals or standards. Although begun at this stage, the enrichment process was largely performed during the Designer feasibility study and will be covered in the next chapter. The domains chosen for the first Advisor system were structuring data and structuring processes. This was because, apart from being the first two phases in SSD, the *MODUS* standards place a great deal of emphasis on the need for a rigorous, formal analysis of the business data at the start of the design process. Thus, *structuring data* is a key phase in SSD. In addition, many of the currently available CASE tools address this phase in the development life-cycle.

The domain-hierarchy sheets and topic-hierarchy sheets referred to in Figure VI.iii are a key product of the KE process. The sheets represent the hierarchical set of tasks which are required to complete a specific design process. The domain-hierarchy sheets deal with activities relevant to a whole sub-domain, like *structuring data*. The topic-hierarchy sheets describe sub-activities within a higher-level activity. The resulting tree-like structure of activities is codified in the Advisor KBs and dictates the way a user can navigate the SSD domain.

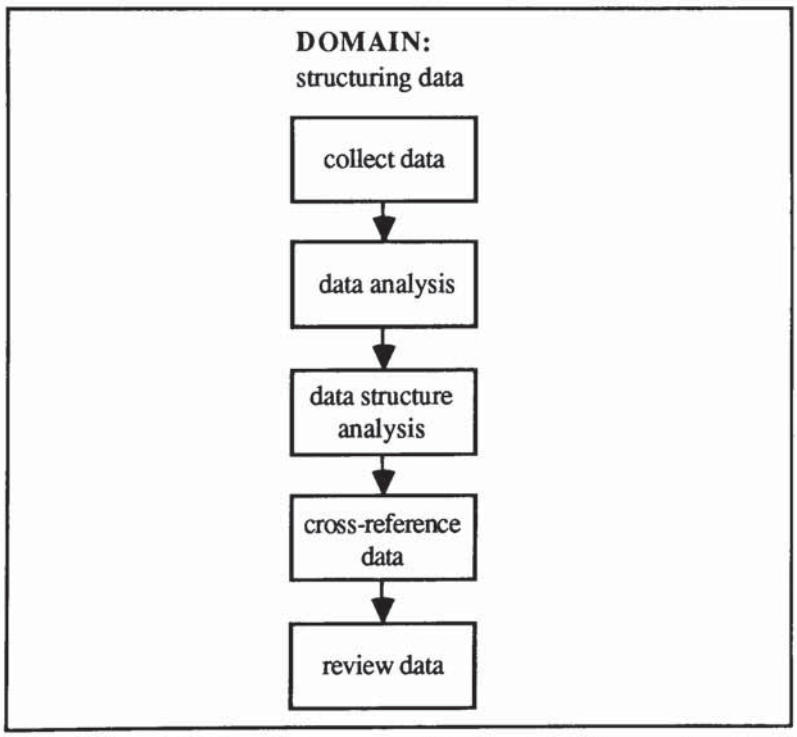

Figure VI.iv - Domain-Hierarchy Sheet for Structuring Data

Figures VI.iv and VI.v give examples of a domain-hierarchy sheet and topic-hierarchy sheet from the sub-domain, *structuring data*. The linear nature of these examples is typical since the sheets show tasks only *one level below* the high-level task concerned.

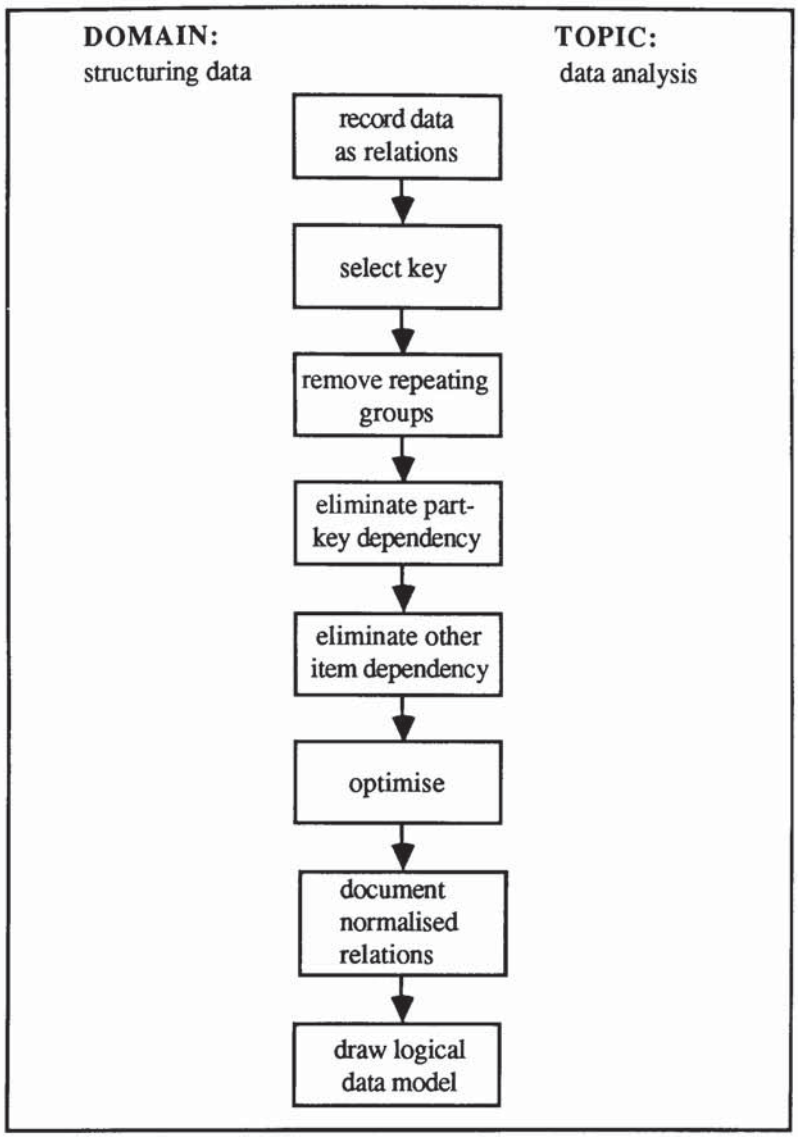

Figure VI.v - Topic-Hierarchy Sheet for Structuring Data/Data Analysis

Figure VI.vi gives an overview of the SSD for the first two sub-domains in SSD and illustrates the complexity of the domain. The rigorous structuring of the KE process was vital for managing the analysis of such a complex domain. For example, the structuring of the KE process allowed the work-load to be spread throughout the project team. Each team member was allocated a group of SSD topics within a given sub-domain. Regular meetings were held to organise and consolidate the separate analyses into a single, paper-based representation of the first two SSD domains, prior to the transfer of the sheets into machine form via the KAM.

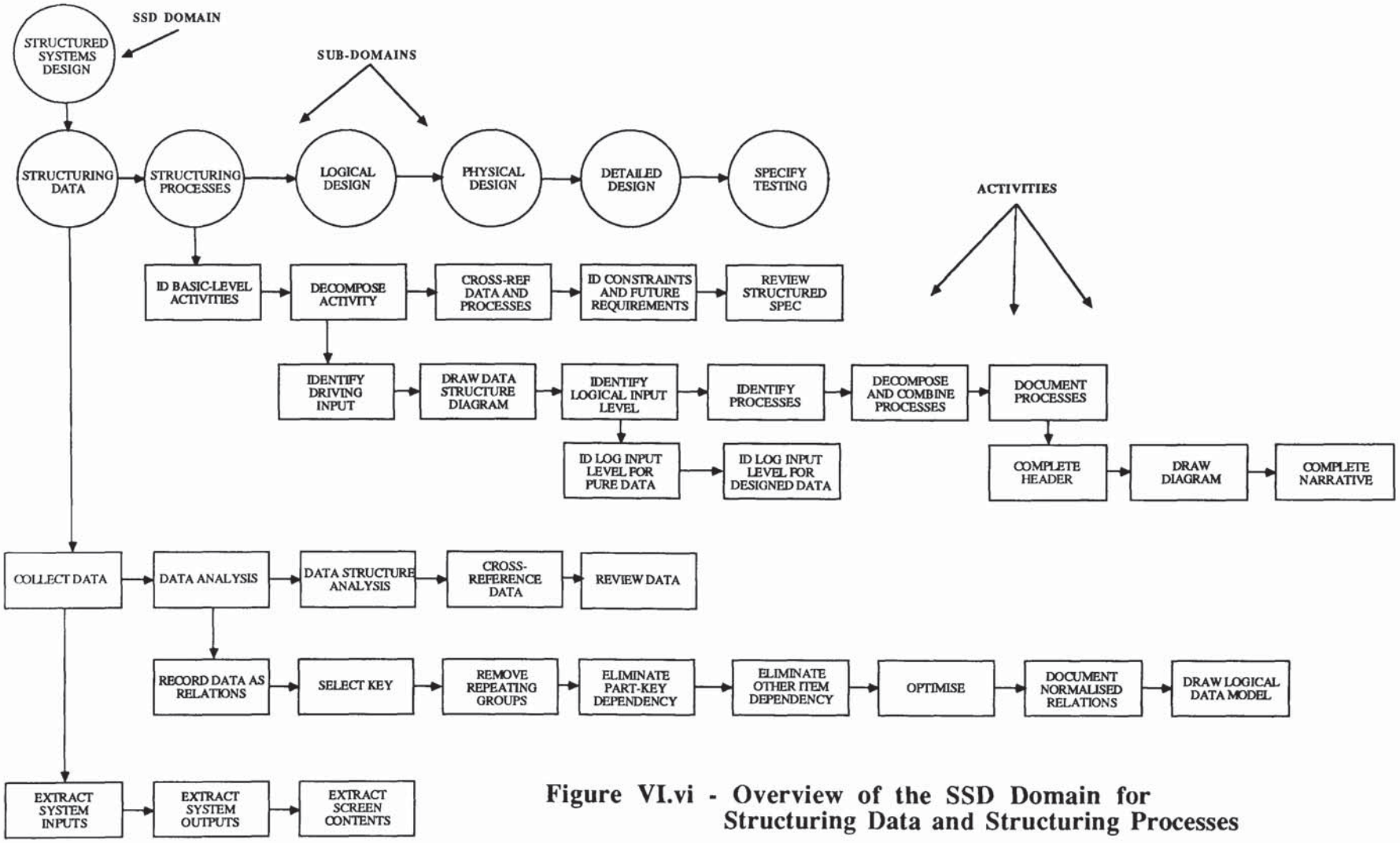

Figure VI.vi - Overview of the SSD Domain for Structuring Data and Structuring Processes

6.7 Implementation of Advisor and the KAM

Advisor and the KAM were implemented using Logic Programming Associates Ltd. (LPA), *micro-Prolog*, running under *PCDOS* on an *IBM PCAT* personal computer (PC). The choice of implementation tools was largely dictated by BIS at the start of the project, before any design and specification work had been started. The reasons for their choice at this stage were mainly pragmatic, rather than based on a logical analysis of what was required. As a result, a number of problems were encountered using the chosen environment. The following sections identify both logical and pragmatic reasons for the various decisions.

6.7.1 Why Use a PC?

- BIS were keen to achieve portability of the tools and clearly the IBM PC route would achieve maximum portability since an IBM PC, or compatible, is the most widely used business microcomputer in the UK.
- LBMS Ltd, a key competitor of BIS, have marketed very successfully the CASE tool, *Automate*, which is PC-based.
- The BIS/IPSE can run in the IBM PC environment under the Unix operating system. The possibilities for linking *Intellipse* tools with the IPSE would be enhanced if the former ran in a similar environment.
- Many other Alvey software engineering projects were using workstation or DEC-*Vax* environments. For example, in a survey of all Alvey projects in 1986, it was reported[HAR86] that, of the 161 project-respondents, 55% were using *Vaxes*, 39% were using *Sun* workstations and only 16 projects (10%) were using IBM PCs. (Some Alvey projects use more than one type of machine.) *Vaxes* and *Suns* are very expensive (> £30k) and are uncommon in a typical DP installation. It was felt by all the team that a key objective of the *Intellipse* project was to make tools available in a format, and at a price, likely to be acceptable in UK DP environments.
- The PC environment was a common factor within the team. Although access to a *Vax* was available at Aston, BSC was an IBM environment, and BIS only had access to PCs.
- A workstation environment would have offered more power and

scope for prototyping work but was beyond the resources of the project.

- An IBM PC was the most convenient medium for exchanging software between BIS, Aston and BSC.

6.7.2 Why Use Prolog?

- In 1985 there was a popular view in the UK, due to the influence of the Japanese "Fifth Generation" AIT programme, which had adopted Prolog as its main programming language, that expert systems were synonymous with Prolog. BIS were therefore keen to enhance their experience of using Prolog, since they believed that this would be a good investment for the future in that it would help them develop their expert system skills.
- The BIS office in Birmingham was also host to the Data Processing Estimator Association (DPEA). The DPEA was a club set up by BIS to build expert systems for software cost estimation. A decision to use LPA *micro-Prolog* for this project had also been taken by BIS. (See 3.4.2).
- By the time Advisor and the KAM were ready to be coded, consideration was given to the use of an expert system shell. A shell was clearly unsuitable since Advisor did not require a rule-base interpreter or editor. Also, the available shells at the time had very poor facilities for linking with other PC systems, and did not provide facilities for the production and incorporation of text or graphics - both of which were essential for Advisor. A PC shell would not be able to represent the complex hierarchical structure of SSD. However, Prolog would be particularly useful in this respect as it allowed complex hierarchies to be coded relatively easily. An AI toolkit running in a workstation environment would have been able to represent the SSD structure, but these systems were beyond the hardware and software resources of the project. The AI toolkits did not run on IBM PCs and would not provide an acceptable delivery environment for commercial DP installations.
- Prolog would also enable the proposed query analyser to be built quickly. The declarative nature of the Prolog language means that a simple pattern-matching parser for analysing the English-like queries could be implemented. The powerful facilities in Prolog for

handling symbolic data structures (words) make it possible to analyse sentences by "cutting them up" into their constituent parts. Prolog rules could then be used to detect key words in the input queries.

- BIS were an official distributor of LPA *micro-Prolog*. In late 1985, there was only one other serious contender in the PC-Prolog market. This was Expert System International's *Prolog2*. This Prolog system had more facilities than LPA *micro-Prolog* but was three times more expensive. The price of *Prolog2*, together with BIS's formal links with LPA, effectively ruled out the use of any other version of Prolog.

6.7.3 Text Entry

LPA *micro-Prolog* did not provide a convenient means for creating the low-level text frames. Microsoft's word-processing package, *Wordstar*, was used to create ASCII-based text pages. A program written in C at BIS was used to access the pages via *micro-Prolog*. The C program was required, as *micro-Prolog* was too slow at reading from disk and displaying on screen the ASCII-based text files.

6.7.4 Graphics

Many of the low-level frames required diagrams but *micro-Prolog* did not have any facilities for creating them. A great deal of time was spent discussing this problem in the project team. Two prototype solutions were explored.

- A character-based diagram editor was written by BIS in *micro-Prolog*. This allowed diagrams to be created using standard ASCII characters. Diagrams were constructed using the keyboard arrow-keys to move around the screen. The ability to edit these character-based diagrams was very limited, and linking them with the low-level text frames was tedious.

- A more sophisticated bit-mapped diagram editor was built in *micro-Prolog* by the author. The system exploited a proprietary graphics library and could be driven by a mouse. It also allowed text and graphics to be mixed easily. However, although this system had far more scope, it was cumbersome for creating and

editing *large* amounts of text and, because it was bit-mapped, could not be linked easily to Advisor. Advisor was based on a 25 (deep) x 80 (wide) ASCII character screen.

Because of the difficulties of linking the bit-mapped diagram editor to Advisor, its more sophisticated facilities were sacrificed in favour of the less sophisticated BIS system, which had the advantage of being character-based and could therefore be linked easily to Advisor. The use of the BIS editor did, however, significantly slow down the translation of the paper-based frame sheets into machine format via the KAM.

6.7.5 Detailed Physical Design

Figure VI.vii shows annotated fragments of the Prolog knowledge bases within Advisor. The diagram gives an indication of how the Advisor frame-like schema was implemented.

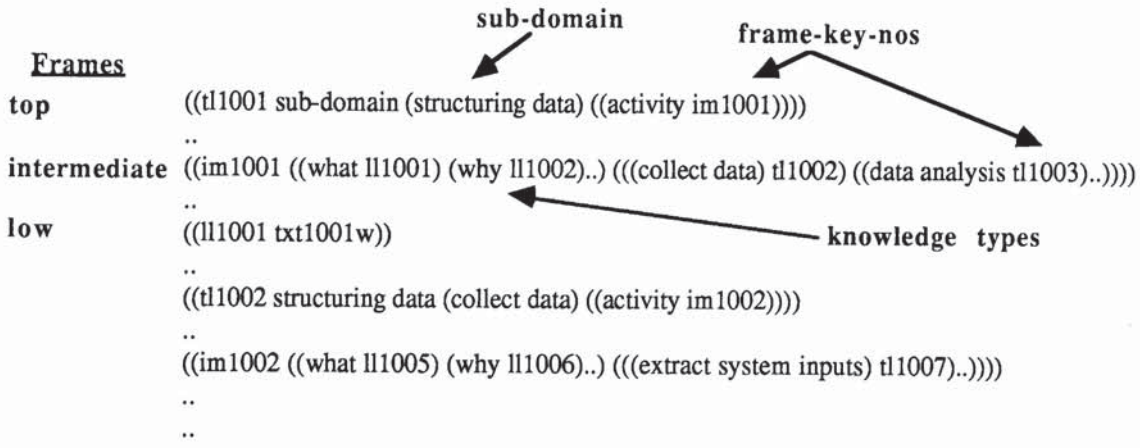

Figure VI.vii - Fragments of Advisor KBs in Prolog Syntax

Much of the detailed design of the KAM and Advisor was done at Aston, and the system was coded at BIS.

6.8 Advisor - User Interface

The first version of Advisor successfully implemented the SSD hierarchy and provided all the specified facilities for navigating the KBs. However, its interface was poor as it obscured from view the SSD hierarchy. The low-level frames could be displayed but it was not clear at what point in SSD the user was situated.

The failure of this first interface was for three main reasons.

- Insufficient effort had been made by the author in specifying clearly what the interface was meant to show.
- No attempt had been made to prototype different interfaces to evaluate their effectiveness or their technical feasibility.
- Too much emphasis had been placed on implementing the SSD hierarchy and the Advisor frame structure, and not enough thought had been given to the best means of allowing the user to navigate the KBs.

These shortcomings occurred mainly because of the experimental nature of the Advisor work referred to in 6.2. It is not unusual in prototype software development for the user-interface to be accorded little attention.

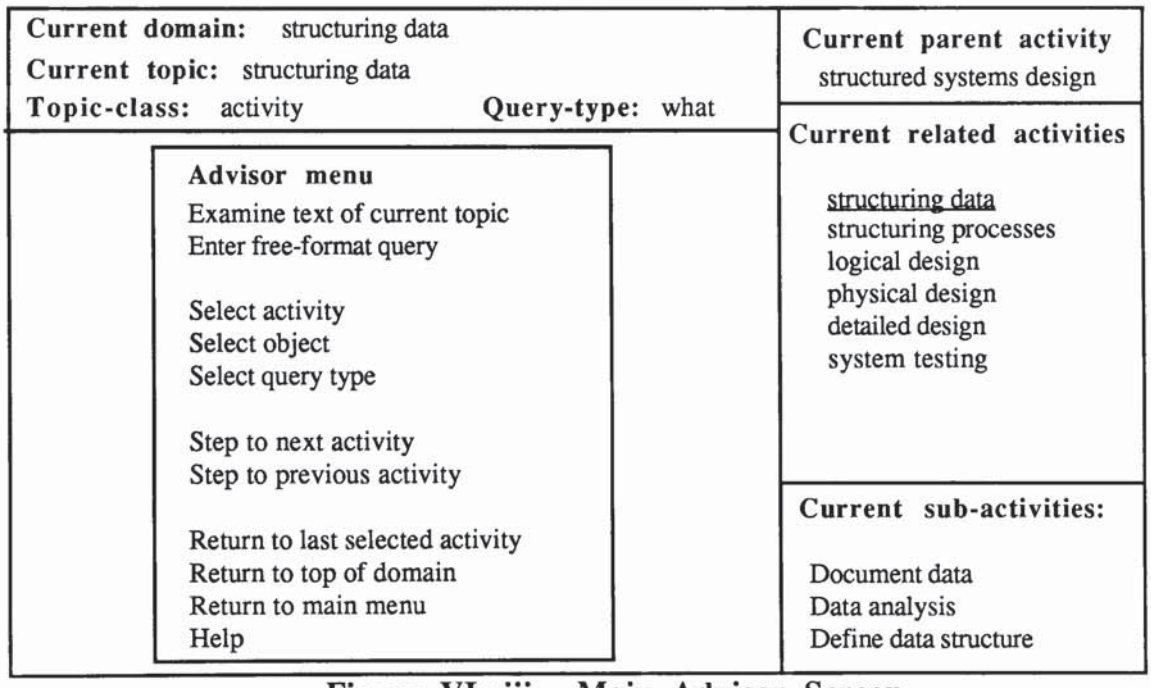

Figure VI.viii - Main Advisor Screen

The problems of the first Advisor interface were overcome by prototyping potential Advisor screens using the desk-top publishing facilities of the *Apple Macintosh.* The author made up several possible screens on paper and these were used as a basis for discussion amongst the project team. After two days work, agreement was reached on new versions of the Advisor interface. These paper-based screens were used as specifications for the programmer. The new interface designs did not require any

changes to the Advisor code which implemented the frame system. This confirmed that the first version successfully represented the overall frame system. It also indicated that there was a logical separation in Advisor's design between the functional and interface components - a desirable feature as far as maintenance of the system is concerned. The second version of Advisor has two basic screens which are illustrated in Figures VI.viii and VI.ix. In fact, Figure VI.viii is a copy of the paper-based specification which was used as the basis for implementing the design.

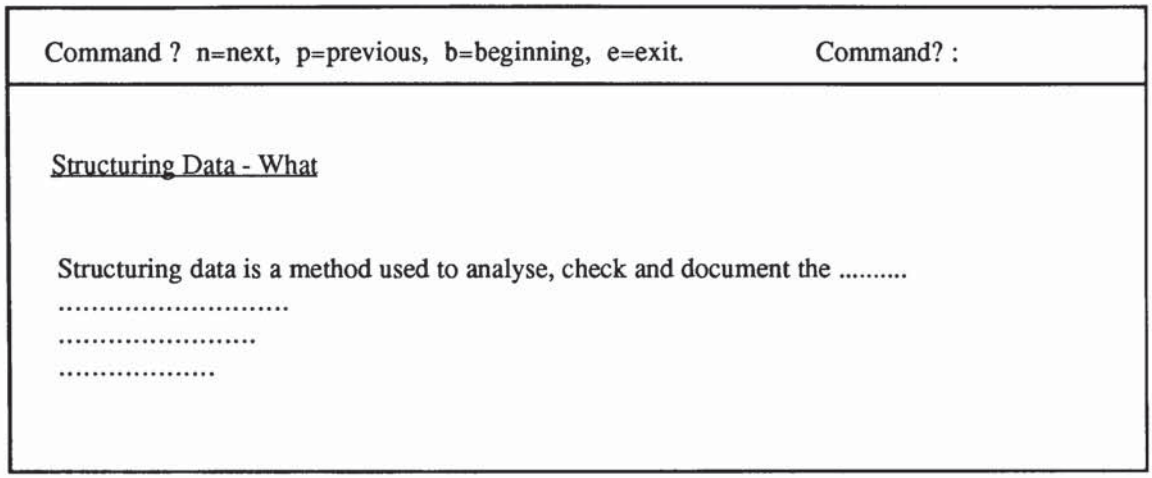

Figure VI.ix - Advisor Low-Level Text Screen

The rest of this section will describe the operation of the system with reference to the menu options shown in the diagram. Advisor was keyboard driven, although the author did experiment with a mouse-driven menu system later in the project. The technical feasibility of this approach was demonstrated and the use of this type of interface is discussed in 6.11.

The main Advisor screen was designed to make explicit the relationship between the currently selected activity and its position within the SSD hierarchy. The "Current related activities" window (CRAW) shows activities on the same level as the selected activity. Activities on the same level denote the fact that the activities all belong to a higher level activity (the parent), and that the execution of the parent-activity involves the completion of all of the child activities. In Figure VI.vi, activities on the same level are shown as a series of horizontal boxes linked by arrows. The first activity in the chain is attached to the parent of the chain. The ordering of the tasks from left to right (Fig. VI.vi), or top to bottom (Advisor screen), indicates the order in which the activities are performed. The currently selected activity is highlighted in CRAW and is shown in its correct position in the chain.

The "Current sub-activities" window (CSUB) shows the sub-activities (child-activities) belonging to the currently selected activity. The "Current parent activity" window (CPAR) shows the owner of the currently selected activity and is also, therefore, the owner of all the activities in CRAW. The CRAW, CSUB and CPAR windows form a "grandparent-parent-child" relationship. The top-left hand window shows the currently selected domain and topic, the topic-class (object or activity) and the current knowledge-type for that topic (what, why etc.).

Examine text of current topic - invokes the text screen for the selected domain/topic/knowledge-type. The text can be paged backwards and forwards. On exiting the text screen, the user is returned to the main Advisor screen.

Enter free-format query - allows entry of a free-format query. The text screen appropriate to the topic/knowledge-type is displayed. The domain is assumed to be the current domain.

Select activity - the Advisor menu is replaced by a menu consisting of the activities displayed in CRAW and CSUB. The user can choose any one of the available activities. After selection, the main screen is redrawn to reflect the new selection and its parent-activity, related activities and sub-activities (if any).

Select object - a list of objects relevant in the domain are displayed and the user may choose any single object. The main screen is redrawn to show the selected object, but the CRAW and CSUB are left blank as an *activity* has not been selected.

Select query type - this option allows the query type to be changed. The screen is redrawn accordingly.

Step to next activity and *Step to previous activity* - these allow the user to move one activity at a time along the list of activities shown in CRAW only - that is, a group of activities which are on the same level in Figure VI.vi, and which contain the currently selected activity. The user is not allowed to jump between levels using these commands. If the commands are used when the selected activity is either the first or the last activity in CRAW, the system defaults to the first activity in CRAW.

Return to last selected activity - often a low-level text frame may refer to an object which is unfamiliar to the user. In this instance the user may select that object via the main menu and examine its text frame. It is useful if the user can then go back to the activity

in the SSD hierarchy which was current when the object was selected. This avoids the necessity to step through the domain. It also overcomes the problem of the users forgetting where they were when the object was selected.

Return to top of domain - returns user to the first activity in the domain.

Return to main menu - makes available some housekeeping commands and allows the user to quit the system.

6.9 User Evaluation of Advisor

Advisor was formally evaluated by BSC and the FOT team at BISBK. In addition, BIS5-9 and ICI1-3 were asked to comment. Advisor was also shown at several exhibitions including the 1987 annual Alvey conference.

6.9.1 How Was the Evaluation Conducted?

The FOT team were given Advisor for a month. They were asked to use Advisor as a support tool in relation to the FOT project which was using SSD. At the end of this trial period, a meeting was held with the team to obtain its observations. Comments were sought on two aspects of the system: the Advisor interface and mode of operation, and the applicability and clarity of the KBs. The meeting was tape recorded and a verbatim transcript made of the tape. Both the author and BIS made summaries of the transcripts.

The same procedure was followed with BSC. However, BSC were not SSD users; nor were they using the methodology in relation to a specific project. Their evaluation of Advisor was therefore somewhat constrained and their comments tended to be of a more general nature. BSC were not able to comment on the text in relation to the BIS courses or manuals. Instead they tended to question some of the procedures and techniques suggested in SSD. These issues were not pursued, as it was not the purpose of the project to assess the validity or applicability of SSD in comparison to other design methods.

The evaluation exercise and the interviews were of a subjective nature. It was not possible to devise formal experiments which could, for example, have compared the performance of the FOT team at BISBK with a control team who did not have Advisor. This type of approach was precluded as it was not clear what metrics could be used to measure the respective teams' performance; besides the project lacked access to other

project teams using SSD.

The subjectivity of the exercise manifested itself in the form and content of the questions posed by the author, during the evaluation interviews and in the way these interviews were directed. For example, where problems were identified by an evaluator, the author tended to suggest possible solutions immediately. By the time the summary of the transcript was made, it was not always clear who had originated some of the suggested improvements. There was also a degree of subjectivity on the part of BISBK and BSC. Since they were not asked to follow a formal assessment procedure, they made their comments on areas which were of particular interest to *them*. This meant that some aspects of the system were given more consideration than others.

Despite the subjective nature of the evaluation exercise, it was felt that the use of semi-structured interviews was the best available option for conducting the evaluation of Advisor.

The user evaluation at this stage was biased towards an assessment of the potential of the Advisor system to fulfil the role proposed for it, as well as the appropriateness of that proposed role. None of the evaluators were expected to validate the SSD knowledge contained in the KBs. It was the responsibility of BIS experts to assess the accuracy of the KBs, and a separate exercise for this purpose was conducted later in the project. The Designer feasibility study was also used to help validate the Advisor KBs.

6.9.2 Summary of the Observations from the Advisor Evaluation

The following is a synopsis of the detailed points raised by the evaluators. None of the points are attributed to individual evaluators, although there was a high degree of correspondence between BSC and BISBK. The summary is primarily concerned with the problems identified; approving comments are not explicitly mentioned at this stage. General conclusions about Advisor are discussed in 6.10 and 6.11.

Selecting Objects Selecting objects via the main menu was cumbersome. A pointing device which could highlight an individual word in the text was required. The desired information could then be overlaid onto the text screen leaving the system at the selected activity.

Paging Text Using a keyboard character followed by "return" was an inconvenient method for paging quickly through the text. A more convenient scrolling mechanism

was required. Connected with this was the idea that all the knowledge for a given topic should be available at once. That is, the knowledge should still be categorised according to query-type, but it should be possible to obtain the other query-types without changing the latter via the main menu. This was another facility which could be handled by a pointing device. There was a consensus of opinion that some of the main Advisor options should be available from the text screen through the use of pop-up and pull-down menus.

Hierarchy The main Advisor screen illustrated the hierarchy well. However, the partition of SSD shown was limited in size. A diagram of the hierarchy showing a greater region around the selected activity would be useful. This would necessitate the use of a bit-mapped display which could overlay graphics on to the text. A paper-based diagram would also be helpful.

Examples The examples were the most useful aspect of the system. More examples were required as well as a system for classifying or indexing them. This was particularly important if a topic had a large number of examples.

Query Analyser This was useful but its effectiveness would be greatly improved if it could process abbreviations and cope with minor spelling errors. Synonyms should also be allowed especially as SSD often used different terms for the same concept.

The Text A number of detailed technical points were made concerning specific issues within SSD. These dealt, for example, with SSD's coverage of on-line development, and the fact that it was not always clear whether a particular task was mandatory or not.

The response to Advisor was favourable and BISBK said it had been very useful for reference and for "jogging the memory" - particularly in the early days following the BIS training course. It was felt by both BSC and BISBK that the system could be useful in other domains.

ICI felt that Advisor could be useful, but that it was a rather expensive system as it would occupy a machine on a 24 hour basis. They also said that the system response was too slow. However, they only had an IBM PCXT system available for the demonstration and this is much slower than the PCAT. This biased their opinion. BISBK are currently using Advisor on the new IBM PS2, which greatly improves its speed of response (see 6.10).

The several BIS experts who studied the system tended to concentrate on the Advisor KBs. They were quick to spot minor inaccuracies, omissions and mis-interpretations of SSD. It became clear that knowledge presented through Advisor seemed to have greater status than the paper-based manual or standards, and the BIS consultants were keen to have their own variants of specific procedures in SSD encoded in the system. This was expected to some extent, and was one of the main reasons why Advisor was built in the first instance. Advisor was certainly a useful starting point for the KB enrichment process mentioned in 6.1.

6.10 Potential Future Roles for Advisor

This section discusses the potential roles for Advisor, the work that would be required to enable it to fulfil those roles, and the interest that BIS and others have expressed in using Advisor in any of the roles identified. Clearly, a key question is the role of Advisor within the proposed *Intellipse* system. This aspect is discussed more fully in chapter ten after more of the investigation into *Intellipse* has been covered.

6.10.1 Advisor as a Stand-Alone System

Both the technical and training divisions within BIS have expressed a firm interest in using Advisor as a stand-alone tutorial system. It would be used *internally* by BIS for supporting BIS consultants, and in relation to the SSD training courses. Advisor/SSD has little potential as a commercial product for use *outside* BIS. This is for three main reasons:

- the market for a PC-based tutor in BIS/SSD is very limited since the number of SSD users is small;
- the Advisor/SSD system running on an IBM PC or compatible would be relatively expensive compared to the cost of a BIS training course, since it would require the purchase or the permanent use of a machine;
- to market Advisor/SSD commercially would require significant further development effort to improve the user-interface, which could not be justified on the basis of the potential sales revenue.

The internal BIS use of Advisor would require work in the following areas:

- the instantiation of the Advisor KBs for the three remaining

domains in SSD - physical design, detailed design and system testing. (The logical design domain was completed in late 1987).

- minor changes to the user interface.

6.10.2 Advisor and the BIS/IPSE

The BIS/IPSE provides mechanical support for a large number of tasks in systems development. However, it *does not know how to design* and while it encompasses well the BIS standards for SSD, its effectiveness is dependent on the expertise of the IPSE user. Linking the Advisor/SSD KBs with the IPSE was suggested by the author very early in the *Intellipse* project. The Advisor KBs would be the basis for providing context-sensitive design knowledge to the IPSE user while the latter was using the IPSE to support various tasks in SSD. This help could be in the form of a knowledge-based help window which temporarily overlaid the IPSE screen.

IPSE/Advisor would begin to address the general problem of CASE tools which do not know how to design. That is, linking the Advisor KBs with the IPSE could be the first step towards a more *active* support environment. BIS have expressed a definite intention of pursuing this potential role for Advisor.

The use of Advisor for this purpose will require extensive validation of the existing machine-based KBs as well as the draft paper-based KBs for the three domains mentioned above. The technical feasibility of linking the Prolog KBs to the IPSE will also have to be examined. This assessment may suggest that the KBs be re-implemented, using a physical design more compatible with the IPSE. However, this would not require any changes to the basic knowledge representation schema.

6.10.3 Advisor and the *Intellipse* APMs

In the same way that the Advisor/SSD KBs could be used within the BIS/IPSE, the KBs could also be used within the *Intellipse* APMs. Since the knowledge in Advisor about specific SSD tasks is being used as a basis for specifying individual APMs, it should be relatively simple to use the relevant part of the KBs as a form of context-sensitive help within the APMs. The KBs could also act as an explanation facility within the APMs.

This idea is being explored as part of the ITAM work covered in chapter nine. The analysis work for ITAM is generating rules, procedures and explanations which are

being systematically recorded. Also, the domain structuring used during the knowledge engineering for Advisor has been used during the ITAM project to produce domain and topic hierarchy sheets. These could be used to build the appropriate Advisor KBs for the ITAM tasks.

6.10.4 Advisor as a Shell

It was noted in 6.5 that the design of Advisor and the KAM enables the knowledge bases to be filled with knowledge from domains other than SSD. The commercial potential for such a system is limited, as the AI workstation environments have powerful facilities for the type of object-oriented representation needed to represent highly structured domains. Although these systems are expensive compared to most PC software, they are likely to become increasingly available to PC users as the PC becomes more powerful. BISBK, however, have begun a feasibility exercise to examine the potential for Advisor to be used for an application domain, other than SSD, related to the FOT system.

The FOT system is a large software system comprising hundreds of individual programs and files. The use of SSD and the BIS/IPSE to support the FOT project has resulted in the production of a large quantity of documentation, a part of which represents the hierarchical relationship between the many parts of the FOT system.

The system will eventually be supported by BISBK's regional offices which are spread all over the world. If a technical problem arises with the system at a client-site, it is the responsibility of the regional office to identify the nature and location of the problem, and to amend the system-code if required. Identifying the particular module which is causing the problem within such a complex system can be difficult, and the objective of BISBK is to use Advisor to represent the FOT system, so that it can aid the location of the faulty module. Advisor/FOT would be distributed on floppy disks for use on PCs located in the various regional offices.

Preliminary feasibility work has been done to assess the suitability of Advisor for this purpose and it appears at this stage that the proposed objectives can be met. The aspect of the Advisor knowledge representation schema, which has had to be reviewed for the FOT system, is the knowledge-types and the categories of "things" in the domain. For example, instead of objects and activities, Advisor/FOT is using the terms *programs* and *reports*. The hierarchical nature of the Advisor representation has been found to be usable without modification, and the structured approach to KE, described in 6.6, was found by BISBK to have been particularly valuable.

6.11 Advisor - the Lessons

This section discusses the lessons learnt from the Advisor exercise. The discussion is related particularly to the reasons for building the system advanced in 6.1, and the methods followed during the development discussed in sections 6.2 - 6.8.

Ease of Construction Advisor meets the informal functional specification defined in 5.2.2. It is a *passive* tool and the basis it lays for a more *active* tool generated a lot of interest on the part of the evaluators. However, this interest also suggests that Advisor falls far short of the active support which is desired by developers in CASE tools. The speed with which it was built confirmed that, of the elements identified in the *Intellipse* system, Advisor is likely to prove, in the long-term, to have been the easiest part to build.

Intended Users BISBK represented "naive SSD users" and they confirmed that the system did assist the FOT project, when team members needed reminding about aspects of SSD covered on the training course. The proposed use of Advisor internally by BIS consultants suggests that the "rusty SSD user" can also benefit from the system.

Knowledge Representation Schema The frame-like schema was successful in representing the SSD hierarchy, although it was sometimes difficult to separate the knowledge into the what, why, how and when categories without repeating some of the information. The distinction between what-type knowledge and how-type knowledge was found particularly useful and, although the Advisor/FOT is not using the Advisor/SSD knowledge-types, it is retaining the concept of object-like and activity-like entities.

Query Analyser The evaluation exercise confirmed that only a simple analyser was needed. The users of Advisor required a method for gaining fast access to lower levels within the SSD hierarchy. It was also clear that users preferred using simple combinations of keywords to access the system, rather than complete, grammatically correct natural language sentences, which are cumbersome to *type* into the machine.

Knowledge Engineering This aspect of the Advisor exercise provided the most important lessons. In 6.2 it was indicated that conventional life-cycle approaches were found to be inappropriate, even though a *structured* approach was adopted in practice. This method was dictated by the width and complexity of the SSD domain.

The rigorous structuring of the domain and the use of the diagrammatic techniques discussed in 6.6 for representing the knowledge were extremely useful. It is difficult to see how the complexity of the domain could have been managed without this approach. Although excellent for managing the process, the structured nature of the approach did take considerable time and involve large amounts of documentation. On balance, the time involved was repaid by the ease with which the knowledge could be organised and maintained. This observation is similar to that made in relation to the use of structured methods for conventional systems analysis, referred to in 4.1. Ease of management and maintenance is often cited as the main reason for adopting these methods in DPSD. After completing the knowledge engineering for Advisor, the project team had gained a clear understanding of the SSD domain. Subsequent knowledge engineering for more specific areas in SSD was, therefore, greatly assisted by this initial exercise. The Advisor/FOT project at BISBK, described in 6.10.4, also confirmed the value of the structured approach to KE employed during the Advisor project.

The lessons learned concerning the applicability of structured methods for knowledge engineering were the first steps towards the proposed, structured life-cycle development model for KBSs, which is introduced in chapter eight.

Use of the PC The use of a PC had been dictated by a number of pragmatic considerations. The PC was convenient and assisted the exchange of software between the collaborators. The commercial wisdom of using a PC is something that can only be assessed in the long-term, if BIS decide to market any of the tools produced.

Micro-Prolog The use of the particular version of Prolog - LPA's *micro-Prolog* - did cause a number of problems.

- The version of *micro-Prolog* used was an *interpreted*, as distinct from a *compiled*, version. This presented performance problems. Current versions of *micro-Prolog* can be compiled and would have been preferable to the version used for Advisor, had they been available at the time.
- The evaluation space of *micro-Prolog* is limited to 64K.
- *Micro-Prolog* offers limited facilities for handling graphics and its file-handling features are primitive. This was overcome by writing file-handling routines in C. A Prolog system which incorporates a graphics library would be a more sensible approach, where significant amounts of graphics are required.

- *Micro-Prolog* has a number of useful features for creating window-based interfaces. However, mouse drivers are not provided and coding more sophisticated interfaces requires the integration of *micro-Prolog* with other software written, for example, in C.

Prolog - in General The use of Prolog confirmed its power for performing complex symbolic manipulation. This allowed the query analyser to be written in less than fifty lines of code. The analyser had no performance problems, due to the limited vocabulary employed by the users, and the consequent small size of the machine-based vocabulary.

Although Prolog is useful for handling symbolic data structures, it suffers from performance and complexity problems, when attempting to code conventional algorithms. The conversion of the frame-based knowledge representation schema was very natural using Prolog. Prolog does not offer the flexibility for building interfaces which is available in languages like C. Where Prolog is required for symbolic processing, a compiled version running on a high powered machine is likely to lead to the best results. Until the performance of the PC is greatly improved, PC Prologs will tend to suffer from operational performance problems.

User Interface The scope for building innovative user interfaces is dependent on the power and flexibility of the implementation tools. Therefore, consideration should be given early in the design process to the user interface, in order to make a realistic assessment of the implementation tools required. Interface designs are likely to be more sensitive to the implementation environment than the design of the processing algorithms required by the system.

Paper-based prototyping of interfaces using desk-top publishing facilities, such as those of the *Apple Macintosh*, was found to be a useful way of specifying and evaluating potential interface designs. This is a much less expensive alternative to using the powerful prototyping facilities found in AI workstation environments.

General Consideration should always be given to the feasibility of using a shell or other packaged software before attempting to code from scratch. Some of the problems with graphics might have been avoided, if earlier consideration had been given to finding a graphics package which met the requirements of the project. There is a delicate balance between pragmatic considerations and ease of implementation, where tools are concerned. Some of the implementation problems could have been avoided only by using hardware and software, whose cost was far beyond the resources available to the

project. This approach was not, in any event, a serious alternative, since the more powerful workstation environments would have been completely inappropriate for use in a commercial DP environment in the UK - at least for the foreseeable future.

The most frequent criticism of the Advisor interface was the difficulty of navigating the KBs *quickly*. It is clear that a mouse-based pointing device would greatly enhance the system, especially if this was combined with the use of pop-up and pull-down menus. The most frequent praise for Advisor was the clarity with which it presented the hierarchical structure of the SSD domain.

6.12 Summary

This chapter has described the design and construction of Advisor. The Advisor project served three crucial purposes:

- It focused the project, at a time when clear direction was needed.
- It played a fundamental part in helping the project team structure and understand the SSD domain and thus laid the essential basis for subsequent work on the *Intellipse* system.
- It provided a useful tool in its own right.

The construction of Advisor necessitated the structuring of the SSD domain in a rigorous and understandable manner from the viewpoint of the project team. Having achieved the latter, it was now possible to begin the Designer feasibility study, whose main objective was to examine individual SSD tasks and assess their feasibility for knowledge-based support.

Chapter Seven

The Designer Feasibility Study

Mephistopheles
...The web of thought, I'd have you know,
Is like a weaver's masterpiece:
The restless shuttles never cease,
The yarn invisibly runs to and fro,
A single treadle governs many a thread,
And at a single stroke a thousand strands are wed.

Johann Wolfgang Goethe 1749-1832 (Faust/Part One)

Preamble

This chapter discusses why a feasibility study was necessary and how the study was carried out. It gives a more detailed description of some of the key areas of design within SSD, particularly in relation to the list of KBS suitability factors given in 3.2.

7.1 Why Was the Designer Feasibility Study Necessary?

The *Intellipse* concept proposed in 5.3 envisages individual APMs to support discrete design tasks in SSD. However, not all the tasks would require expert, that is, KBS support. Some of the tasks required only mechanical support of the kind already provided by BIS/IPSE and other available CASE tools. Of those tasks not supported by tools, it was reasonable to assume that only some of them would be suitable for a KBS approach. The main objective, therefore, of the feasibility study was to identify design tasks which could, potentially, be supported by a KBS. This approach would also make it possible to begin to validate the *Intellipse* concept.

Satisfying the criteria for KBS suitability in 3.2 was a necessary, but not sufficient, basis for the identification of SSD tasks for KBS support. Since the resources available to the project were limited, it was also necessary to select a task, or tasks, for which a detailed investigation into the potential for KBS support, could be started within the remaining resources. It was clear that it was impossible to build the APMs for the complete *Intellipse* system within the lifetime of the Alvey project, or even to identify and specify all those that would be required. However, establishing whether the system could be built in principle, by attempting to specify individual APMs, and assessing the potential

for building others in the future, were realisable objectives.

The criteria which were used during the Designer feasibility study, as a basis for identifying tasks suitable for KBS support, were thus as follows.

- Was adequate, conventional (mechanical) support available for the task already?
- Was the task important, relative to other SSD tasks? (E.g. was it on a critical path?)
- Was the task one which was normally only performed successfully by experts, and which inexperienced designers found difficult?
- Did the task satisfy most of the criteria for KBS suitability in 3.2?
- Was the task sufficiently bounded to render meaningful progress towards a KBS tool possible during the remainder of the Alvey project?
- Would a KBS support tool for the task have more general applicability, particularly to a DP environment, such as BSC's, which did not use the SSD method?

The last point above was important. BSC had remained on the margins during the Advisor project. This was mainly because they were not an SSD user and had limited interest in a support tool specific to SSD. Investigating a KBS support tool of potential benefit to the BSC installation was regarded as one of the more important issues for the remainder of the *Intellipse* project. In addition, if the task chosen for KBS support was relevant to BSC, it was likely that some of their own experts could be involved in the investigation.

7.2 How Was the Designer Feasibility Study Conducted?

The method used for the feasibility study was a series of knowledge engineering sessions, involving three experts from BIS and three experts from BSC. These experts and their areas of expertise are shown in Figure VII.i. The author also attended two of BIS's training courses on SSD and Database Systems Design as part of the feasibility study.

Only the first four domains in SSD, structuring data (SDA), structuring processes (SP),

logical design (LD) and physical design (PD) were considered in the feasibility study. The study was limited in this way for three reasons:

- the first four sub-domains were representative of SSD as a whole;
- by the time the last two sub-domains were to be considered in detail, there was a high degree of consensus upon which SSD area to focus on for KBS support during the remainder of the project;
- the limited resources meant that, in the light of the last point, it was sensible not to extend the feasibility study unnecessarily.

The starting point for the sessions, involving the BIS experts, were the domain and topic-hierarchy sheets and text, generated during the knowledge engineering for Advisor.

EXPERT	AREA OF EXPERTISE	NUMBER OF KE SESSIONS
BIS5	structuring data physical design database design	5
BIS6	structuring processes logical design	5
BIS7	physical design database design	1
BSC1-3	physical design database design	1

Figure VII.i - Experts Involved in Designer Feasibility Study

The approach adopted for the KE sessions is summarised in Figure VII.ii.

The KE sessions were tape-recorded and verbatim transcripts made of the tapes. The transcriptions were done by a professional audio-typist.

This method has two important advantages:

- Taping eliminates the need for note-taking during the session by the knowledge engineer. This is crucial, since it was found to be very difficult to participate properly in the session and, at the same time, take down detailed notes.
- A verbatim transcript provides an accurate record of the session

which can be used as reference material in the absence of the expert. The transcript can be sent to the expert to allow him/her to validate it, and any summaries made, and to prepare for any subsequent KE sessions. The experts appreciated the chance to look over what they had said, although some did not want the transcripts circulated widely.

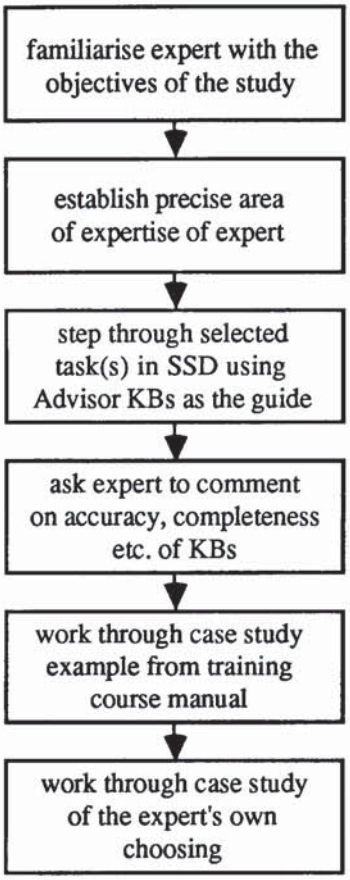

Figure VII.ii - Outline of Procedure Used During KE Sessions

There are also a number of problems with using tape-recording and verbatim transcripts:

- The tapes can be difficult to transcribe, especially if a large number of technical terms are used which are unfamiliar to the transcriber. Making available to the transcriber a paper-based glossary of frequently used, technical terms can alleviate this problem.
- If more than one expert and one knowledge engineer are involved in the session, the transcriber may have difficulty differentiating between the participants. It is important, therefore, for KE

participants to speak clearly, and avoid frequent interjections.

- Verbatim records of human conversation, even where the latter is focused on a specific subject, usually contains large amounts of irrelevant information, badly formulated and unfinished sentences, and interjections which are barely audible. This can make it difficult for the transcriber to interpret the tape.
- Some of the explanations by the expert can involve diagrams and other visual material which is not recorded on the tape. The use of a video tape-recorder can be used, if a visual record is important. Shpilberg et al[SHPI86] have reported on an expert system development project which used video-taping during knowledge engineering.
- Some experts may be inhibited by the presence of the tape-recorder.

In order to avoid some of the difficulties of verbatim transcription, an attempt was made by the author to create summaries of the audio tapes by playing them back and making handwritten notes. This was abandoned, as it was extremely tedious and took inordinate amounts of time to complete even small portions of the tape. When professional audio-typing resources were available, it was found preferable to make summaries of the KE sessions from the type-written verbatim transcript. The transcripts were annotated approximately every half-page with the counter setting from the tape-recorder. This allowed the knowledge engineer to go back later and find quickly parts of the tape which needed clarification.

On balance, the advantages of taping and transcribing KE sessions outweighed the disadvantages identified. The ability of the knowledge engineer to participate, question and otherwise interject, while the expert is articulating his expertise, was found to be crucial to the success of the knowledge engineering process.

The typed verbatim transcripts were used as the basis for future knowledge engineering sessions, and to analyse the individual SSD tasks. In addition to analysing the tasks in relation to the criteria in 3.2, the following information was also required:

- Confirmation of the task and sub-task hierarchies in Advisor.
- Identification of missing tasks.
- Identification of the data exchanged/transferred between neighbouring tasks.

- Identification of type and amount of support available in the BIS/IPSE for specific tasks, as well as any other BIS-internal, or proprietary CASE tools which were relevant.
- Qualitative assessment of the degree of conventional and heuristic techniques employed by the expert to perform SSD tasks.

The ability of the expert to articulate verbally his/her expertise proved to be an important factor in the knowledge engineering sessions. Two of the BIS experts had extensive experience of tutoring on BIS's training courses and, as suggested in 5.4, this was reflected in the productivity of their sessions. These sessions needed minimal involvement of the knowledge engineers once the session had started, as the experts were able to work through the material in a systematic and logical manner. In contrast, where the expert had difficulty in articulating clearly, the author had to continually lead the discussion by asking detailed questions, in order to achieve similar clarity of explanation. Sessions with the articulate experts could last for nearly three hours, but two hours was found to be the maximum useful span in the other cases. The KE sessions resulted in hundreds of pages of type-written transcripts, hand-written summaries, diagrams and other material. The results of analysing this data are given in the next section.

7.3 Summary of the Results

The sub-domains of structuring data (SDA) and physical design (PD) are discussed in detail with reference to the criteria for the designer feasibility study identified in 3.2 and 7.1. The sub-domains of structuring processes (SP) and logical design (LD) are covered more briefly.

7.3.1 Structuring Data

SDA is a critical area in the *MODUS* development phases. There was some ambiguity in BIS as to whether SDA should form part of the *MODUS* Structured Systems Analysis (SSA) phase, or whether it should be the first step in SSD. During the lifetime of the *Intellipse* project, the detailed structure of the SSA and SSD training courses were changed and less emphasis was placed in SSD on some aspects of SDA. The overall importance of SDA to SSA and SSD together was not affected. The transfer of some of the SDA tasks to SSA was partly because BIS's training division felt that there was too much material to cover in the one week SSD course.

The main objective of SDA is the production of a normalised or Logical Data Model (LDM) of the business data relevant to the proposed application area. SSD's approach to SDA is based on Codd's established theory of relational data analysis[(CODD70), (DATE81)]. The specific end-product of SSD is a model of the business data in *third normal form* (TNF). Although Codd's theory has been extended since it was first proposed to enable data to be expressed in fourth, fifth and other normal forms[(DATE81)], SSD does not make use of these extensions.

Some of the objectives put forward by Date[(DATE81)] to justify the relational approach are listed below.

> "To provide a high degree of data independence.
>
> To provide a community of view of the data of spartan simplicity, so that a wide variety of users in an enterprise (ranging from the most computer-naive to the most computer sophisticated) can interact with a *common* view (while not prohibiting superimposed user views for specialized purposes).
>
> To simplify the potentially formidable job of the database administrator.
>
> To introduce a theoretical foundation (albeit modest) into database management (a field sadly lacking in solid principles and guidelines)." (Page 487.)

Relational data analysis (RDA) is a feature common to many other structured information system development methodologies. The basic steps in RDA as it is described in SSD are listed below. The list is from the Advisor KBs.

1 Record data as relations.
2 Select prime key.
3 Remove repeating groups (first normal form).
4 Eliminate part-key dependency (second normal form).
5 Eliminate other item dependencies.
6 Optimise and re-check (third normal form).
7 Document normalised relations.
8 Draw logical data model.

An example, also taken from Advisor, of some data in third normal form, with their associated logical data model (LDM), is given in Figure VII.iii. The notation adopted is that recommended by SSD.

Many of the proprietary CASE tools offer sophisticated mechanical support for RDA. For example, *Automate* (LBMS Ltd.) provides facilities for creating and editing data relations using a mouse-driven user interface. The tool provides a basic framework for carrying out the RDA tasks, and can perform automatically some limited consistency checking and optimisation. *Automate* can also generate a first pass database (DB) schema for a limited range of proprietary DBMSs. Other products, such as *Normal* (Cincom Systems Inc.), and *IEW* (Arthur Young Associates), offer similar facilities.

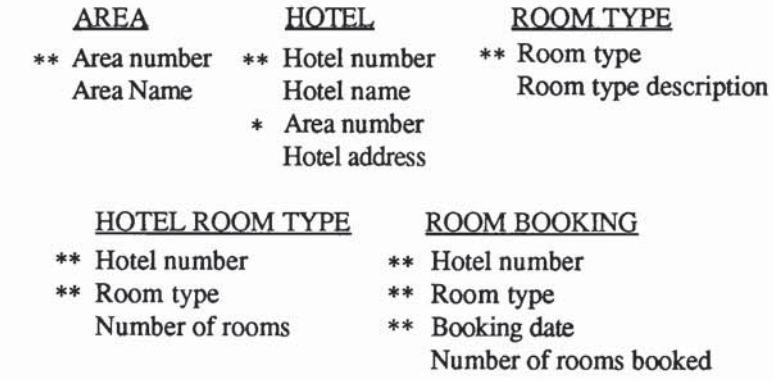

Figure VII.iii - Example of Data Relations in Third Normal Form and the Associated Logical Data Model

While the tools described provide excellent *mechanical* facilities, it must be stressed that none of these tools *know how to perform RDA*. They rely on the designer to make the appropriate decisions as far as the detailed steps in RDA are concerned. Typical mistakes made by inexperienced designers, which were identified by the experts are, for example, the selection of keys and repeating groups, which were *logically* correct, but

totally inappropriate in a *business* context. These sorts of mistakes would not be identified by any of the tools discussed above.

The BIS/IPSE offers less sophisticated facilities in the RDA area. However, unlike *Automate*, *IEW* and *Normal*, which are limited in scope, the BIS/IPSE offers mechanical support for a much larger number of phases in the development life-cycle. Notwithstanding the last remark, the three tools mentioned have achieved some success in the DP market place for PC-based, Wimp-driven CASE tools. The BIS/IPSE is not specifically directed at the PC-tool market, but at the CASE market for more powerful *IPSE* tools. However, its success here was limited and there was significant emphasis in BIS on the need for a CASE tool which would compete more directly with *Automate* and similar products.

It was noticeable, therefore, in the early stages of the feasibility study that BIS were keen to steer it towards those areas in SSD (such as structuring data) which, in their view, were of commercial importance. The *Automate*-type tool represented the approach which many at BIS thought had good commercial potential. The author's view was that a tool which augmented the mechanical facilities available in *Automate* with some knowledge-based features, thereby endowing the tool with *knowhow* about design, was an area with more commercial potential than a tool which merely reproduced the usual mechanical support for RDA. In the event, at about the same time as the feasibility study, the BIS/IPSE team in London were finalising plans to produce a cut-down version of the BIS/IPSE for the PC-based CASE tool market. This meant there was much less emphasis on producing such a tool, as part of the Alvey project.

There are major difficulties in building *rule-based*, KBS tools in the RDA area. (By *rule-based KBSs* are meant systems whose knowledge representation is based on the use of "if-then" production rules, and whose architecture is like the early ESs illustrated in Figure III.iii.) The key tasks in SSD/SDA, listed as 1, 2, 3, 4 and 5 on page 133, rely overwhelmingly on *general knowledge* of the specific business domain involved, related application domains and the accumulated experience of the expert gained from past RDA exercises. Lengthy consideration was given, in conjunction with the experts, to ways in which this knowledge could be generalised and represented in machine form. However, the kind of knowledge being employed by the experts to, for example, select a key in an un-normalised data relation, is being drawn from a human KB, which is far too wide to encompass in a machine. The expert was using knowledge which, in another context, would be regarded as common sense.

As discussed in 5.4, in the context of DPSD, common sense can be taken to mean the general knowledge about abstract design techniques, and detailed domain-specific experience, which the expert designer accumulates over an extended period of time whilst working on a diverse range of systems development projects.

The problems arising from the use of common sense reasoning are exacerbated by the fact that much of the relevant business-specific data, which the expert uses in RDA, is not readily available but has to be obtained by interviewing clients. When the data *is* available, it is normally in a form which cannot be utilised easily in a machine environment.

The other tasks in RDA (6, 7 and 8) are algorithmic in nature and can be performed automatically using conventional implementation techniques.

As far as the criteria in 3.2 were concerned, two features of SDA made it particularly unsuitable.

- The problem is not well-bounded and relies predominantly on common sense knowledge.
- Those tasks, which can be supported, are already adequately addressed by conventional techniques.

Ceri and Gottlob[CERI86] discussed the use of a Prolog system for RDA. They characterised their program as "...suitable for the interactive design of small database applications and as a teaching aid." However, the work reported is confined to a description of how various conventional normalisation algorithms are represented as Prolog rules. The system would not be of use to an inexperienced designer in the context of a commercial system development project, as it does not offer facilities for taking into account data about the application domain.

Bouzeghoub and Gardarin[BOUZ85] described SECSI, an expert system for supporting the production of a set of normalised data relations based on information about the application supplied by the user. The system is also implemented in Prolog. SECSI requires a "sound semantic schema" in order to produce a normalised model. The system interacts extensively with the user, in order to identify characteristics about the application domain which are relevant to the normalisation process, but not covered by the semantic schema. In a typical commercial application this would not be a viable

approach because of the very large number of questions which would have to be asked. The inexpert designer would also have difficulty in answering some of the questions.

Conclusions

- Heuristic tasks in SDA cannot be supported by a *rule-based* KBS due to the level of common sense reasoning used by experts.
- A degree of KBS support for the heuristic tasks could be achieved by incorporating suitably augmented Advisor KBs for the SDA sub-domain into the BIS/IPSE - in the manner described in 6.10.2. In particular, the Advisor KBs could be augmented with an extensive set of generically classified RDA examples which covered most of the common problems found by BIS experts in the course of their work. A limited paper-based version of such a set of examples is already available in BIS, but does not appear to be widely used. Incorporating this into Advisor/IPSE is likely to greatly enhance the usefulness of these examples.
- The BIS/IPSE is the appropriate environment to support those algorithmic tasks in SDA which can be approached using conventional techniques. Other mechanical support could also be provided in the IPSE.

7.3.2 Structuring Processes and Logical Design

The KE sessions for these areas were affected by two main factors.

- The expert involved was not able to articulate as well as the other experts interviewed.
- It was apparent from the start that many of the tasks involved in SP and LD were very unsuited to a KBS approach.

The construction of a DP system involves two fundamental components, the *data* which the system must use, and the *processes* to be carried out on that data. The processes are synonymous with the detailed functions which the system is expected to perform. While SDA is the method used to analyse the data and structure it, in a form suitable for the next stage in the design cycle, SP is concerned with structuring the processes and documenting them in a systematic way. The low-level processes, which a DP system

must perform, are derived from the high-level transactions which future users expect the system to execute. For example, in the hotel application used in Figure VII.iii, an enquiry (a high-level transaction) might be expressed in the following way:

> *Approximately three times per day, the hotel receptionist will want a list of all rooms of a certain type unallocated for a given number of consecutive days, from a given day. This enquiry will be made on-line, and the report is to be printed immediately on a printer situated in the reception area.*

The main object used in SP is the process sheet. An example is given in Figure VII.iv. A process sheet is completed for every low-level process into which the overall system specification has been decomposed. For a large system there will be hundreds of process sheets produced.

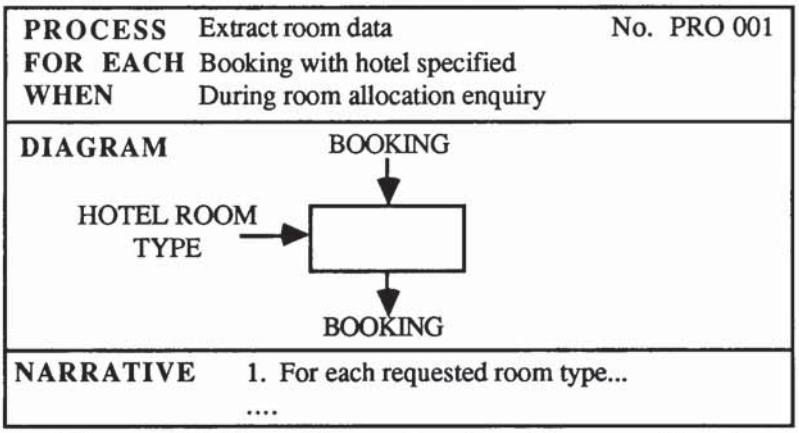

Figure VII.iv - Example of a Process Sheet

At the logical design stage the process sheets are sorted and organised into *transaction profiles*. A transaction profile is a group of low-level processes which form a logical representation of a high-level system function. An example of a transaction profile is given in Figure VII.v.

The final outcome of the LD phase is a Logical System Chart (LSC) which contains, in diagrammatic form, a complete description of the processes, transaction profiles and data which are required to perform the system functions identified for the application. The notation used in the LSC allows the time-ordering of the system processes to be expressed, as well as the static data files involved. The *logical* nature of the LSC means that the design does not reflect any characteristics of the hardware and software

environment, which may be used to implement the application represented by the LSC. The LSC is the basis, therefore, for the second, physical half of the SSD design cycle.

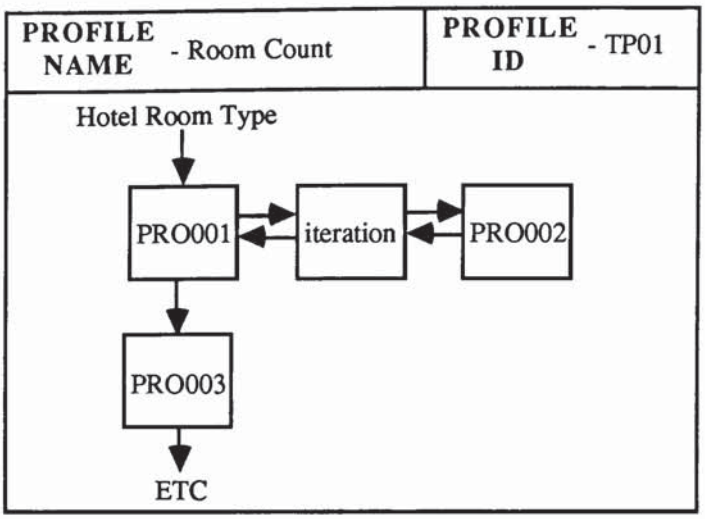

Figure VII.v - Example of a Transaction Profile

Conclusions

The scope for building rule-based KBSs in the near future in the SP and LD areas is limited. The essential difficulty is finding a knowledge representation schema which will allow process sheets, transaction profiles and similar objects to be held and manipulated in a *machine environment.* Some heuristic knowledge is used to identify processes and group them into appropriate transaction profiles, but, unless the SP objects themselves can be represented, it will be very difficult to exploit the heuristics identified using a KBS. The iterative nature of many of the tasks in SP and LD means that mechanical support, which offers sophisticated facilities for editing and creating diagrams easily, is the critical support requirement. This is another area served well by proprietary CASE products including the BIS/IPSE. *The linking of the Advisor KBs for SP and LD with the IPSE is again the best form of KBS support which could be used in these areas in the short-term.* Other ideas for KBS support which were explored briefly with the experts were as follows:

- The use of a KBS tool for performing syntactic and consistency checking of process sheets.
- The addition of a KB in Advisor containing example process sheets organised into generic classes.
- The use of a KBS tool for advising on the completion of process

sheets, and in particular the narrative part of the sheet.

- The construction of a KB of generically classified transaction profiles, which could be coupled with a KBS facility for advising on their use.

These ideas for KBS support in SDA/LD were identified as worthy of further detailed research, but were not pursued during the designer feasibility study. The reasons for this are discussed in 7.3.5.

7.3.3 Physical Design

The objective of the PD phase in SSD is to begin the transformation of the LSC into a set of detailed program and file specifications. These specifications reflect the physical environment chosen to implement the system, and would be used by the programmers to code the system software. SSD emphasises the physical implementation of a DP system using *a high-level language (e.g. COBOL) + program-managed data files.*

This view envisages that the program specifications will define all of the procedures necessary for the creation, storage and retrieval of data in the system. However, although this approach to DPSD is still widely prevalent in the UK, there is an accelerating movement towards a different implementation method which can be characterised as *a high-level or Fourth Generation language (4GL) + DBMS.*

In this approach, all data management is handled by a proprietary DBMS, and the application code is confined to the area of system functions only. The importance of this approach was clearly identified by the BIS experts, and there is also evidence from the DP press(JONES86c), (NORT86), (LAWR87), (STUR87), (MACI87) that DBMSs are becoming a key component of the DP environment. (It should be noted that the MODUS environment, as distinct from SSD, does provide standards for both file-based and DBMS-based systems development.)

The rest of this chapter concentrates on the approach to physical design based on the use of a DBMS. This reflects the way that the KE sessions developed - and was a direction dictated by the opinion of the experts, as well as by the author's own independent assessment.

7.3.4 Database Design

Database design involves two main components, the Logical Data Model generated during the SD phase, and the list of system transactions identified in SP. A first-pass DB design is obtained, by transforming the LDM into the DB schema appropriate to the DBMS being used to implement the system. This first-pass schema must be *sized* and *timed* to check whether it meets the physical constraints laid down in the system specification.

The process of first pass physical design is primarily concerned with analysing the performance of a system based on a pure LDM. This first-pass design is modified to become a second-pass design by making various degradations to the LDM in order to meet specific physical constraints. The main objective is to achieve the required physical performance by adjusting the logical design, whilst retaining the maximum possible degree of maintainability and clarity inherent in a normalised LDM. This process involves the application of heuristic rules for changing the design, followed by a re-calculation of the design's performance. Invariably, a design alteration can produce several interacting effects on the design's performance. This means that design changes must be balanced against each other, in order to gain the desired result. At any point several different strategies may be available to the designer, of which only some are capable of meeting the desired objectives.

The initial analysis of the database design area suggested that it was suitable for a rule-based, KBS approach.

7.3.5 DB Design and the Criteria for the Feasibility Study

This section discusses DB design in relation to the list of KBS suitability factors in 3.2, and the specific criteria for the designer feasibility study given in 7.1.

KBS Suitability Factors (from 3.2)

Narrowness of the Domain The problem is well-bounded. Some general knowledge about the application domain is important but this is well-defined, and the experts are definite about where this knowledge can be obtained.

Complexity of the Problem By concentrating on one proprietary DBMS, the complexity

of the problem can be greatly reduced. The experts use a well-defined procedural approach which involves both heuristic rules and conventional numerical methods. The procedural nature of the experts' method means that the problem can be factored and represented as a hierarchical structure of tasks, similar to that used for analysing other aspects of SSD in the Advisor project.

Nature of the Problem The problem does not rely on visual or other sensory skills. The refinement of a first-pass DB design into a form which meets the physical constraints and objectives of a specific application is a problem normally only carried out by human experts. Furthermore, BIS experts have a well-established record of successfully performing this type of design exercise. The experts are also of the view that the design problem itself is relatively straightforward, providing the appropriate techniques are *known*. BSC also has some established procedures in the area.

Nature of the Experts Two experts were available in BIS, both of whom had extensive experience of DB design in general, and one who had particular expertise in the proprietary DBMS used by BSC. Both had taught on BIS training courses and were known to be highly articulate. Expertise at BSC was also available.

Training The DBS training course promotes a highly structured approach to DB design. This provided a sound basis for the KE required.

Speed of Solution The design problems are solved 'off-line' and take minutes, hours or a few days to solve, depending upon the size and nature of the application.

Sensitivity of the Problem There are no obvious legal or ethical issues involved.

Conventional Solutions Computational tasks are involved and these could be supported easily using conventional techniques. However, there are very few computer-based support tools known to the experts which provide this kind of support. The integration of conventional and knowledge-based support appears to be both feasible and desirable.

Written Material A large amount of documentation is available covering the general design tasks involved, and the specific procedures associated with the proprietary DBMS.

Feasibility Study Criteria (from 7.1)

Conventional Support Conventional support tools were not available as far as the experts and the project team were aware.

Importance of the Problem in SSD As has been indicated above, the use of DBMSs in commercial DPSD is of growing importance. It was also clear from anecdotal evidence given by the experts, based on their work with BIS clients, that DP installations often have significant problems achieving the desired performance from their DBMS applications. This was the case, even when there were no hardware or software-based restrictions to prevent them attaining the required objectives. This assessment was borne out by the author's own independent analysis of the problem.

KBS Suitability The problem appeared to satisfy most of the KBS suitability factors in 3.2.

Scope of the Problem The problem could be bounded in such a way that the feasibility of building an operational KBS could be adequately demonstrated. It was felt that this demonstration of feasibility could include the design and implementation of some elements of the system, within the remaining project resources.

General Applicability and Relevance to BSC By choosing the proprietary DBMS, *Total* from Cincom Systems Inc, the investigation would have direct relevance to BSC who used *Total* as well as other Cincom products. The following factors indicated that an investigation, initially restricted to the *Total* DBMS, would have general applicability:

- The newer Cincom product, *Supra*, a relational database, although different from *Total* in its mode of operation - from an application-user's point of view - was very similar to *Total* from the DB designer's viewpoint. That is, those KBS facilities developed for *Total* would also be applicable to *Supra*. This was important, as *Total* is being superceded by *Supra*. In fact, it was clear from discussions with experts in Cincom and BIS, that *Supra* could utilise a KBS tool to an even greater extent than *Total*. This is because the extra flexibility that *Supra* offers to the DB user, at the logical level, necessitates a much tighter control of the DB at the physical level, if performance constraints are to be met.

- The general techniques used by the expert in the DB design area are applicable to all DBMS-based applications. Only a relatively small number of the *detailed* tasks are specific to a particular DBMS.
- Of the many DBMS products on the market in the UK, Cincom's share of the sites, where DBMSs were in use, is significant. 5.51% of DBMSs presently in use on IBM mainframes in the UK are from Cincom Systems Inc[MACI87].
- The use of a KBS to obtain optimum performance of software, running in a commercial mainframe environment, is of growing importance. Two articles on the subject appeared in 1988 in the DP press[LANG88], [BETTS88] on the subject. In 1988, System Designers Ltd announced their product, *SD Tuner*, which is an expert system for tuning the Vax/Vms mainframe operating system. This product is discussed by Betts. (*TimmTuner* from the General Research Corporation in the USA is another expert system available for tuning the Vax/Vms operating system.) These developments indicate that the use of KBSs, in place of human systems programmers, is a feasible approach, and that the building, tuning and optimisation of a DBMS design is similar to the problem of managing a large mainframe operating system.

There is an additional feature of DB design, which makes it a particularly interesting area: the combination of *conventional algorithmic tasks* and *constraints-based, heuristic refinement* of an initial design. The integration of conventional and KBS components was identified in 3.1.4 as an area of growing importance. Thus, the DB design domain offers a good opportunity to evaluate the extent to which this integration is possible.

On the basis of the criteria for the Designer feasibility study, a decision was taken to begin a more detailed investigation into the feasibility of a KBS tool for supporting the design, implementation, and maintenance of a DP application, running in the *Total* DBMS environment. There were valid reasons for not investigating in detail, at this stage, the further ideas for KBS support identified in the feasibility study.

- The form of KBS support identified for the other SSD areas was principally based on the incorporation of augmented Advisor KBs into the BIS/IPSE. The feasibility of building these KBs had been

demonstrated during the Advisor project. Incorporating them into the IPSE was essentially a technical problem which could be investigated by the BIS/IPSE team.

- Pursuing the ideas in the other SSD areas had limited relevance to BSC who were in favour of the ITAM investigation.
- The use of a KBS support tool in the DBMS area was an innovative approach, but it still needed significant research effort to establish its real credibility.
- Cincom Systems Inc, who were consulted about the proposed ITAM investigation, before a final decision was taken, had responded very positively to the idea.

7.4 The Lessons of the Designer Feasibility Study

7.4.1 Nature of the Designer's Expertise

The extensive knowledge engineering sessions, involving different designers, allowed some conclusions to be drawn about the nature of the expertise possessed by experts in DPSD. The skills identified were as follows:

Analytical - the ability to extract the relevant aspects (or essence) of a proposed application from the mass of irrelevant information, normally collected as part of the DP analysis and design phases.
Experiential - the ability to identify the common factors between a current design problem and a problem solved in the past, enabling features of that solution to be used in the current application.
Innovative - the ability to improvise and find new solutions to design problems, when none of the normally-used techniques appear to work.
Procedural - the ability to apply their extensive knowledge of design practices in a consistently accurate way.
Abstractive - the ability to generalise from particular design solutions and form generalised approaches which worked in other domains.

The DP designer's expertise consists of two major components. Firstly, *abstract*, domain-independent knowledge about the design process itself - represented, for example, by the SSD methodology. However, a key observation made during the designer feasibility study was that designers also make extensive use of a second

component - *application domain knowledge* - that is, their general knowledge about banking, insurance, manufacturing and so on. In fact, the successful use of the abstract design knowledge was largely dependent on the degree to which the designer was able to utilise his/her domain-dependent expertise. *Many of the heuristics used by the experts are based on their domain-specific knowledge, built up over a long period of time.* The expert and non-expert designer can be distinguished by the degree of domain-dependent experience which they possess.

These observations are in line with those made by Adelson and Soloway[ADEL85]. They looked at the role of domain experience in software design and observed that:

> "A designer's expertise rests on the knowledge and skills which develop with experience in a domain." (Page 1351.)

Greenspan[GREENSP86] also identified the importance of domain knowledge:

> "Software systems are developed to solve problems that exist in some application domain, for example in the world of manufacturing, business, or geological exploration. Knowledge of the application domain must play a major role in the software development process, since it provides the context for both defining the problem and for evaluating the validity of solution systems." (Page 61.)

Barstow[BARS87] concluded that:

> "*Software Engineering is a knowledge-intensive activity.* Of special importance are knowledge of the application domain and knowledge about the design and implementation history of the target software itself." (Page 209.)

Ryan[RYAN88] looked at the nature of the software developer's expertise. He identified the need for KBS tools in software engineering to have "domain and target system knowledge", and also the difficulties of adequately representing this knowledge in machine form, using present techniques. This is in line with the conclusions in 7.3.1 and 7.3.2 concerning the difficulty of representing the knowledge used by experts, to perform many of the tasks in SDA, SP and LD.

Sharp[SHARP88] reported on the results of an experiment designed to assess the role of domain knowledge in software design. Fifteen software designers were asked to perform a software design task, based on a given set of documentation. The latter included an appropriate functional specification and other relevant material. Designers spent between 140 and 210 minutes on the task. The tasks were in different application

areas, and the designers were asked to complete questionnaires which allowed them to indicate their degree of domain knowledge about the application tasks.

Sharp concluded from the investigation that:

> "This study has provided no evidence for supporting the hypothesis that general domain knowledge affects the production of a good system design...However, some designers claim that application domain knowledge is necessary for producing a good design, and some claim to have used it during this study, although there is nothing in their design notes to support this claim." (Page 16.)

The discrepancy between the results of Sharp's work and the author's conclusions can probably be explained by two things. Firstly, the experiment by Sharp was necessarily restricted to relatively small application tasks, compared to a typical DPSD project. It would be difficult to conduct a similar investigation in the commercial DP environment, although it would be an extremely valuable exercise. Secondly, the experiment concentrated on the *software design phase* in the conventional life-cycle, whereas the Designer feasibility study took a broader view of the whole DPSD cycle.

Other points which emerged from the Designer feasibility study were:

- Carrying out the Designer feasibility study would have been extremely difficult, if not impossible, without the work previously involved in building Advisor. The Advisor KBs were the foundation for the feasibility exercise.
- The use of tape-recording and verbatim transcripts worked well, although this depended on the co-operation of the experts, and the availability of professional typing services.
- The study had been technology or KBS-driven. The exercise was one, in which a chosen solution was looking around for an appropriate problem. For an R&D project this is an acceptable approach; however, for operational or commercial KBS development, it is important that the feasibility study is focused on a specific problem, for which conventional solutions have been found to be unsuitable.

Finally, the results of the Designer feasibility study suggest that existing mechanical CASE tools can be made more active by augmenting them with knowledge-based

components. In particular, the BIS/IPSE could be augmented with the Advisor KBs to produce an environment capable of offering significant knowledge-based support in the near future. This support need not be *rule-based*, but could provide facilities which give the inexperienced designer information about the critical factors, which an expert designer would consider in relation to specific design problems. This type of support would still leave the initiative with the designer, but it would increase the chance that the less expert designer would consider the relevant factors before making design decisions.

7.5 Summary

This chapter has described the designer feasibility study. DB design was identified as the most appropriate area for further investigation during the *Intellipse* project. In addition, the results of the study indicated that rule-based KBS support for DB design was technically feasible. Other forms of KBS support for tasks in SSD have been identified, although some areas of SSD were found to be unsuited to a rule-based approach. The concept of augmenting the BIS/IPSE with the Advisor KBs, first proposed in chapter six, was again shown to be the best short-term method of providing knowledge-based support for SSD tasks, not suited to a rule-based approach.

Up to this point, the *Intellipse* project had concentrated on the application of KBS techniques to DPSD - the *KBS for SE* theme. However, two factors were inevitably pushing the *SE for KBS* theme into the picture.

Firstly, the experience of using a structured approach to KE in Advisor had been very positive. The type of diagrammatic techniques employed, and the decomposition of activities into sub-activities had similarities with approaches used in conventional systems analysis.

Secondly, the basis of the Alvey project was the SSD methodology, itself an *engineering* approach to computer systems development. Once a specific KBS support tool *for use in an operational environment* had been identified as the focus for the project, the importance of considering the applicability of structured development methods to the development of an operational KBS became evident. In particular, the opportunity was there to assess the suitability of conventional development methods for building an operational KBS.

Chapter Eight

Practical Engineering of Knowledge-Based Systems

> Ask not what your country can do for you, but what you can do for your country.
>
> John Fitzgerald Kennedy 1917-1963

Preamble

This chapter considers the applicability of conventional DPSD methodologies, such as the BIS SSD approach, to the development of operational KBSs. It discusses the differences between DP and KB system development, and proposes some initial steps which could lead to the establishment, in time, of a structured engineering approach for KBS development. The author was in an advantageous position for considering these issues, since the *Intellipse* project work had necessitated a thorough study of both KB and DP system development methods.

8.1 Why Are Structured Development Methods Relevant to KBSs?

The discussion of operational KBSs in 3.2.2 showed that few KBSs are in operational use. The great majority of KBSs, built so far, have been of an experimental or prototypical nature. In addition, those KBSs, which are in operational use, are mainly small, stand-alone systems which do not interact with other computer systems. Worden[WORD87], for example, said:

> "Most knowledge-based systems in current use are small, standalone systems with little connection to the mainstream data processing of an organisation." (Page 60.)

Waterman noted that[WAT86]:

> "Most expert systems never get past the research prototype stage. This is because until recently most were developed in research rather than in commercial environments." (Page 212.)

It follows that most writers on KBS development are from an academic or R&D environment and few have considered the need for KBSs, intended for operational use, to be built to the standards expected in conventional DP systems. Their research

background also means they are frequently unfamiliar with conventional DP practices.

The objectives of using structured engineering methods in commercial DPSD have been discussed in 3.5 and 4.1. This discussion indicated that structured methods are expected to provide the following advantages:

- *Control* - the ability to plan, schedule and run a DPSD project according to a fixed budget, and to meet precise delivery targets.
- *Robustness* - the ability to build a system which will continue to function, even when users do not interact with it in precisely the expected manner.
- *Reliability* - the ability to continue to function on a 24 hour basis, where required, or to be able to be shut-down and re-started in a predictable way.
- *Maintainability* - the ability to add functionality to the system after it is operational, and to fix problems easily where these arise.

This is not a complete list of desirable attributes to be expected from the use of a structured methodology, but is intended to give an idea of what should be expected from an engineering approach.

Although not referring specifically to KBSs, Partridge[PART86a] identified the need to build *practical AI software.*

> "...if we intend to push on with our AI programs into the harsher world of applications software, we will require all of the usual desiderata of practical software:
>
> (1) perceptual clarity;
> (2) robustness;
> (3) reliability;
> (4) maintainability." (Page 130.)

Ford[FORD86] commented on the lack of a methodology for practical AI software.

> "AI has yet to evolve a methodology suited to the production of practical software, largely because most AI software has been of an experimental nature, and thus not developed in a disciplined way to satisfy the needs of users, managers, and others one stage downstream in the lifecycle, but also because it has been unable to draw from the methodological foundations provided by SE." (Page 263.)

If operational KBSs are regarded as a particular type of computer system, and if future

KBSs are to be integrated more fully with existing DP systems, it follows that KBSs should meet the same operational standards expected in the commercial and industrial environment. Furthermore, since structured engineering methods are the most widely established means, at present, of attaining the desired operational standards in conventional DP systems, then it is reasonable to propose that similar engineering methods should be employed in the development of KBSs. This proposition suggests that the following key question be addressed:

> *To what extent are existing conventional structured methods appropriate to the development of an operational KBS?*

This initial question points to two further areas of investigation:

> *What are the essential differences between DP and KB systems, particularly in relation to the methods used for their construction?*

And, if there *are* significant differences:

> *How does the conventional life-cycle development model need to be adapted to meet the particular needs of KBS construction, and the general requirement of KBSs built to commercial and industrial operational standards?*

8.2 Contrasting Features of DP and KB System Development

Waterman[WAT86] considered the differences between data processing and knowledge engineering. Figure VIII.i illustrates his results.

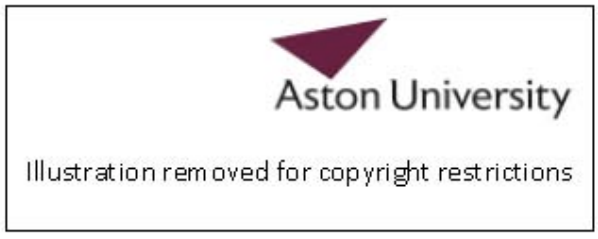

Figure VIII.i - Comparison of DP and Knowledge Engineering (from WAT86)

Gervarter[GERV83] compared AI with conventional programming and his results are shown in Figure VIII.ii.

Artificial Intelligence	Conventional Computer Programming
Primarily symbolic processes	Often primarily numeric
Heuristic search (solution steps implicit)	Algorithmic (solution steps explicit)
Control structure usually separate from domain knowledge	Information and control integrated together
Usually easy to modify, update and enlarge	Difficult to modify
Some incorrect answers often tolerable	Correct answers required
Satisfactory answers usually acceptable	Best possible solution usually sought

Figure VIII.ii - Comparison of AI with Conventional Programming
(from GERV83)

Partridge[PART86b] analysed the differences between the nature of AI and software engineering problems.

> "*AI problems*
> (a) Answers tend to be adequate or inadequate.
> (b) Context-sensitive problems.
> (c) Dynamic.
> (d) Not completely specifiable.
> (e) Performance-mode definition.
>
> *Software engineering problems*
> (a) Answers are correct or incorrect.
> (b) Context-free problems.
> (c) Static.
> (d) Completely specifiable.
> (e) Abstract definition." (Page 31.)

This comparison, however, confuses the *ideal* of software engineering using formal methods, with the *reality* of DPSD in the current commercial environment. This point is discussed again later in the chapter.

Ford[FORD86] considered the differences between the methods used for AI and SE problems.

> "The methods and tools of software engineering are suited to problems that can be characterized as having reliable, static data for which the problem solution space is

> small. Problems with unreliable and dynamic data usually imply a much larger solution space; AI techniques have been developed to deal with this situation." (Page 257.)

The authors cited above have mainly focused on *what* the differences are between problems addressed by AI and those approached using conventional techniques, and they are in general agreement about the nature of those differences. The following analysis concentrates on *how* the construction of commercial DP systems differs from current KBS development practices, in relation to the six key phases in the conventional development life-cycle, namely: feasibility, analysis, design, implementation, testing and maintenance. Inevitably, there is some repetition of points made earlier in the thesis. However, this is essential in order to present a coherent argument.

Feasibility and requirements definition: how much? vs is it possible?

The technology used in building typical DP applications, such as airline booking or credit card management systems, is well established. The balance of debate at the feasibility stage is not concerned with the technical possibility of building the proposed system, but with the time required to build it and the consequent cost. DP projects are often concerned with computerisation of existing manual systems. It is generally an implicit assumption that there will be no intrinsic difficulty in analysing and understanding the manual system, prior to the formal specification of a computer-based alternative. The feasibility study for a KBS must, of course, also address issues of cost and estimated development time. However, other questions which will need to be addressed are quite distinct:

i Problem selection The initial problem analysis will need to consider whether conventional solutions have been adequately considered. A common pitfall in KBS development is the selection of an inappropriate problem. An attempt must be made to classify the complexity of the problem. For example, how much common sense reasoning is involved and how long does the expert usually take to solve the problem? The availability and expected co-operation of the experts must be estimated. Building a KBS with an unco-operative and/or elusive expert will be impossible. An initial technical assessment of the expert's knowledge and expertise will be necessary, in order to establish whether the available knowledge representation formalisms will be able to cope with the problem.

ii Human factors In 3.3.3 the importance of human factors in KBS development was

discussed. In particular, the need to define the prospective users of a KBS was identified. Having characterised as accurately as possible, during the feasibility phase, the users of the proposed KBS, it will be easier during the design phase to take account of human factors when specifying the user-interface for the system. The intention of building a KBS may be to allow non-experts to perform at a similar level of competence as experts in some specific domain. It is essential therefore that the user-interface within the system is tailored as closely as possible to the prospective users. In a conventional DP project, it is rare to give much consideration to the issue of human factors and user-interfaces. This is because the scope for tailoring the interface in a DP system is constrained by the limited facilities available on the "dumb terminals" used in most mainframe computer installations. Also, wider human-organisational factors in relation to a new DP system are normally dealt with at a management level, separate from that of the DP department. The lack of involvement of users in the design of computer systems has been criticised by several authors(MUM87), (LAND87), (CORD85). They point out that this lack of involvement is often responsible for subsequent problems in the operation of some DP systems.

iii Performance objectives A DP system can have a precisely specified set of output requirements or transactions. It is unlikely that the performance of a KBS can be as easily circumscribed. Although the "combinatorial explosion" associated with many AI systems is usually avoided in a KBS, it may not be possible at the commencement of the project to state a semi-formal, closed set of requirements. Some of the requirements may have to be stated in terms of the performance expected from the system. For example, a knowledge-based classification system for analysing rock samples may be specified in terms of the correlation between the system's performance, when used by a 'non-expert', as compared with analysis of the same rocks by an expert. The use of a performance-based requirements statement can also, therefore, serve as a basis for testing a KBS. An example of specifying and testing a KBS, using correlation between expert and KBS performance, is discussed by Edwards and Bader(EDW88).

iv Prototyping It can be seen from the discussion so far that the nature of KBS construction, particularly the lack of a transaction processing-style statement of requirements, means that it is difficult at the feasibility stage to be as certain about development costs as is possible with a conventional DP project. The history of software project estimation in conventional DP shows that, even with a detailed statement of requirements, accurate forecasting of development effort is frequently an illusory target.

Building a small scale prototype of the proposed KBS can be an effective way of assessing the feasibility and likely development costs of a full-scale version of the system. A successful prototype can also serve as an *active*, semi-formal specification for an operational system, since users and managers can use the prototype to ensure that the desired functionality and usability of the system have been accurately identified. However, the scope and methods for a prototyping exercise must be clearly defined. A successful small system at the feasibility stage can lead to the continuation of prototyping towards an operational system. But the latter approach is unlikely to yield reliable or maintainable software. In conventional DP, it is very unusual to "throw away" existing code. Therefore the likelihood that a prototype, built as part of a feasibility exercise, may be subsequently discarded needs to be stated and budgeted for, at the outset of the feasibility phase in KBS development.

v *Hard and soft analysis* Judgements about the desirability or appropriateness of building a KBS, in relation to the estimated financial or other benefits which may accrue to the organisation, using the system, are not discussed here. This type of soft analysis is essential to any proposed computer system project and usually takes place, prior to any decision to build a system. This analysis is confined to the hard design phase, after the judgement on whether to build a computer system has been made. However, a question that arises when building a KBS, which the KBS developer should address if possible, is: how successful is the expert in solving problems in the application domain? This will be discussed below in the section on testing.

Analysis: data analysis vs task analysis.

A conventional DP project begins with an analysis of the existing manual or computerised system. The objective is to produce an accurate, semi-formal model of the business activity being studied. This model is concerned with identifying the data required by the activity, the movement, storage and retrieval of that data, and the processes to be carried out on the data, such that the desired outputs from the activity are achieved. Whereas conventional analysis requires the examination of clerical operations and procedures, KBS task analysis is concerned with the problem-solving activity of human experts. The fulcrum for conventional analysis is the *data*; the fulcrum for KBS task analysis is the *expert*.

Formal techniques, such as Codd's relational model, have been developed to support conventional data analysis[CODD70]. Relational analysis has its roots in set theory and its

correct application can produce a rigorous formal model. Well established formal techniques for analysing the cognitive activity of an expert, who is solving a problem, do not exist as yet. Also, while conventional data analysis can be done largely by external observation of business procedures, KBS task analysis requires a much more intimate involvement with the expert.

As well as modelling the expert's knowledge, the analysis stage during KBS construction may need to perform further the analysis of prospective users of the KBS, begun during the feasibility stage.

Design: complexity of size vs complexity of representation.

Many of the complexities which arise in DP design are due to scale. The use of structured methods to develop the system specification can lead to large quantities of documentation, covering every aspect of a project, from analysis through to coding. The volume of documentation, and the need to link individual items of documentation together, mean that documentation management is a major headache. But the problem is one of management rather than understanding.

Large knowledge bases may also present problems of scale. However, in KBS design complexity mainly arises because of the need to analyse cognitive activity, represent human expertise in a structured form and formalise ill-defined problems and solutions. The problems of knowledge representation are very complex. The data structures employed in a conventional DP system will not be rich enough for the requirements of a KBS. On the other hand, the representation structures commonly used in AI - such as objects, frames or production rules - are themselves only crude approximations to the knowledge structures, apparently employed in human intelligence. In addition, the sophisticated inferencing techniques, employed by an expert, must be modelled using more restrictive mechanisms, such as forward or backward chaining.

The main tasks during the design phase of a DP system are concerned with defining logical and physical file structures and specifying the logical and physical processes, which must access these files. The main output of this phase is a set of detailed specifications for programs which, when executed in conjunction with the data files defined, will generate the output required by the system specification. The design phase for a KBS may also require the generation of program specifications for conventional software components. In addition, it will be necessary to devise an executable

knowledge representation schema and inferencing structure, which can adequately model the expert's activity.

Implementation: program debugging vs iterative refinement.

In the traditional *waterfall* model[(BOEH76), (AGRE86)], the development phases are executed sequentially. The implications of design decisions, made in the early phases, are not reviewed until the code has been executed much later in the development cycle. The waterfall model assumes that, by the time the coding phase is reached, the analysis and design stages are complete. The main objective of the latter two phases is to generate a set of program specifications, which are complete with respect to a set of requirements for the system. Any iteration at this stage should simply be between program coding and program de-bugging. According to the model, after the analysis and design phases have been completed the de-bugged programs should meet user requirements without further modification.

Theory and practice in conventional DP systems development have a habit of diverging. The delay between taking a design decision and reviewing its implications often leads to expensive mistakes, since errors made early in the design cycle, but not discovered until later, can be very costly in development time to rectify[(BOEH76)]. Validation of early design decisions through rapid iteration between design and implementation phases, via the animation of prototype designs, is not encompassed in the traditional waterfall life-cycle. This weakness in the model has led to amended versions being suggested and, in particular, to a proposed life-cycle model based on some form of prototyping[(GLAD82), (HEKM86), (AGRE86)].

KBS construction must include a strategy for incremental development. The waterfall model needs to accommodate iteration between the various development phases, since in KBS development a significant amount of iteration during analysis, design and coding will be necessary in order to identify, specify and represent the range of problem-types and special cases, which the expert can solve in the application domain. It may be necessary to animate intermediate designs of the system, in order to facilitate validation of the analysis and design by the expert and prospective user. This is why prototyping techniques are particularly relevant to KBS development - especially during the feasibility phase. The prototyping process can be a very useful method of attaining the desired performance objectives for a KBS. The crucial point is that this iterative process is managed as part of a developmental model which ensures that the requirements of

reliability, robustness and maintainability are met.

Testing: acceptance testing vs validation of performance.

In DP, acceptance testing is designed to establish whether the system meets the set of user requirements agreed by developers and users at the start of the project. The developers are primarily concerned with building a system which meets those requirements. It is the users or owners of the system who must judge the extent to which the specified system will meet the wider, business objectives which prompted the desire to build the system in the first place.

In KBS development, since we are modelling specific human-cognitive activities, developers must be concerned with the performance of the system beyond the purely mechanical replication of certain requirements. Not only must the system be tested to establish whether it successfully models that part of the expert's activity, which it was designed to duplicate or support, but, if practical, an attempt must also be made to validate the performance of the expert in the application domain. For example, where experts with conflicting opinions have been involved in the construction of an operational KBS, the problem owner must arbitrate and decide, on the basis of the desired business objectives, how the KBS should operate.

Maintenance: Bug fixing vs Knowledge-Base upkeep.

Maintenance in DP is a euphemism often used to describe the process by which delivered systems are gradually adapted to remove obvious bugs in the software and to add essential functionality, which was not envisaged at the time the system specification was finalised. Maintenance can absorb a large proportion of the life-time costs of an operational DP system[ALVEY82].

In KBS development the use of prototyping during the feasibility phase and the animation of intermediate designs may avoid some of the typical problems requiring maintenance. Of course, software bugs may also be present in a KBS and will have to be rectified. However, unlike a conventional system, a KBS is modelling human expertise in a specific domain. Very few fields involving experts remain static; the medical domain is a good example. The knowledge-bases and inference models inside a KBS will have to be continually under review to ensure that the system does not become outmoded.

This analysis has identified some key differences between DP and KB systems development. One of the most significant differences is the importance of prototyping throughout the KBS development life-cycle.

8.2.1 Prototyping in KBS Development

The objectives of prototyping in KBS development, identified in the analysis above, can be summarised as follows:

- To establish the technical feasibilty of acquiring and representing the expert's knowledge in machine form, and to illustrate the key functional requirements expected of the system.

- To aid the evaluation and validation of the machine-based rule-bases, using the expert and problem owner as arbiters.

- To help specify the user-interface by animating possible designs.

- To evaluate and validate the inferencing mechanisms chosen for the system.

- To ensure the range of problem types addressed by the expert are covered by the KBS.

8.3 Current KBS Development Methodologies

Hayes-Roth et al[HAYES83] defined five stages in the evolution of an expert system:

> "Identification: Determining problem characteristics
> Conceptulization: Finding concepts to represent knowledge
> Formalization: Designing structures to organise knowledge
> Implementation: Formulating rules that embody knowledge
> Testing: Validating rules that embody knowledge." (Page 24.)

This does not represent a structured methodology but simply a statement of the general stages associated with KBS development. In particular, it does not identify detailed tasks to be performed in KBS construction. This was not the intention of their description since, at the time the book was written, the development of KBSs was still in

its infancy.

Hayward[HAY86] discussed an Esprit project, looking at structured methodologies for KBS development. He suggested that "rapid prototyping" was "commercially foolhardy" and proposed an alternative development method for expert systems.

> "The alternative taken in our research is to adopt the philosophy of structured development methodology, as familiar from conventional software development. This suggests a partitioning of the development process, with the specification of structured descriptions according to a well-defined process as development proceeds." (Page 197.)

Hayward went on to propose a four phase life-cycle comprising knowledge acquisition, system design and implementation, testing and operational use. The acquisition stage is broken down into three more steps:

- analysis of static domain knowledge;
- analysis of inferential relationships;
- analysis of expert problem-solving strategies.

The work led to a system known as KADS and is implemented in Prolog on *Sun* workstations.

Breuker et al[BREU86] also reported on the work of the KADS project and concluded that:

> "The results of our work to date lead us to believe that the goal of a theoretically well founded methodology is achievable and that with suitable support tools it will be useful and practical." (Page 782.)

Davoudi[DAV87] also reported on the KADS methodology. The KADS project has looked at the way KBS development could be *structured* but, although it has linked its work with conventional software development models, the KADS system does not yet offer a detailed framework of procedures which can be adopted easily in the commercial DP environment.

Partridge[PART86b] characterised the prevailing AI methodology as RUDE:

- **R**un-**U**nderstand-**D**ebug-**E**dit.

He said that:

> "Traditionally, AI programs have not been developed by implementing a *completely specified problem* [My emphasis-JLB]. AI program development has usually been characterised by the run-understand-debug-edit cycle (the RUDE cycle), which can become random code hacking, in the worst case, or incremental analysis and redesign in its better manifestations." (Page 33.)

The RUDE paradigm is synonymous with iterative prototyping and a simplified version of it is illustrated in Figure VIII.iii.

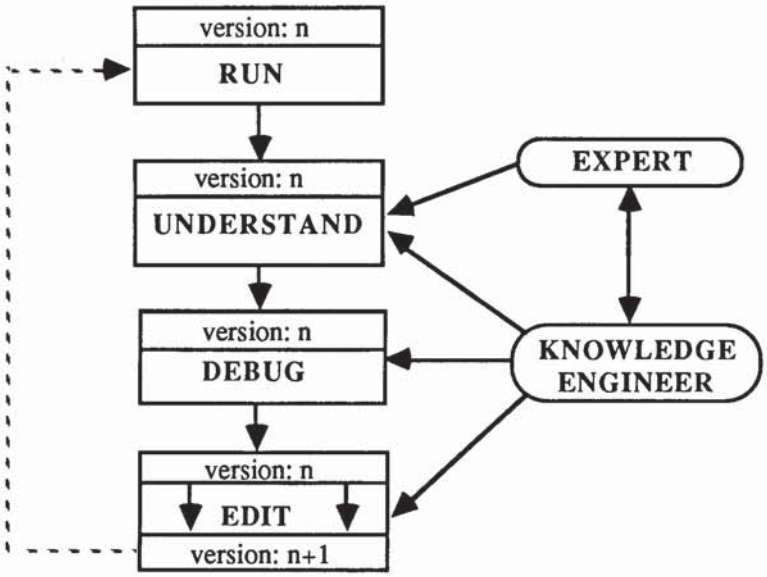

Figure VIII.iii - Simplified RUDE Cycle

Partridge clearly stated the need for improvements in the RUDE paradigm:

> "...the RUDE-based program development methodology is in dire need of development to yield a disciplined derivative that may be used to construct commercial AI software." (Page 34.)

Partridge went on to propose a "complete life-cycle environment" for AI software based on the RUDE paradigm. The proposed environment is discussed more fully in Partridge's book[PART86a] and is based on a model of "controlled code modification" of "AI program abstractions".

> "Code modification should be systematic and based upon the abstract representations that underlie a given inadequate implementation. We have to work with the machine-executable description but we use the underlying specification and less abstract intermediate representations (a possible design sequence) as the

basis for guiding implementation development." (Page 122.)

Partridge did not discuss his RUDE life-cycle explicitly in relation to commercial KBS development and his approach should not therefore be criticised for failing to provide a detailed methodological approach. Partridge, however, identified some of the fundamental issues associated with producing practical KBSs.

The main weakness in Partridge's analysis concerns his characterisation of conventional software engineering which he associates with the notion of "completely specified problems". For example, he said[PART86a]:

> "In software engineering there is the opportunity, which should be exploited to the full, of capturing, independent of any particular persons, a complete and rigorous specification of the problem. That is not to say that human wishes can be ignored, on the contrary that must be taken into account. But this contrasts sharply with typical AI problems wherein certain humans are the only known embodiments of adequate implementations." (Page 120.)

Complete problem specifications in commercial DP are simply a statement, *at a particular time*, of the problem, as it is perceived or understood by developers, users and problem owners. This perception is just as liable to change, re-statement and re-design as many of the problems being approached by AI and KBS developers. The problem of iteration in DPSD was referred to earlier and Boehm[BOEH76] identified its potential dangers to successful DP development. Iteration is sometimes unavoidable, or even desirable, in order to accommodate changes in the perceptions and requirements of those concerned with a particular application. Commercial DP uses the structured conventional life-cycle to control any iteration and, above all, to enable the results of iteration to be *documented* and *maintained*. Thus, the lack of a "completely specified problem" does not necessarily mean that AI or KB software development cannot benefit from some of the techniques evolved for managing conventional system development.

The RUDE paradigm is an appropriate method for doing research and development into KBS systems, and the RUDE cycle encompasses well the iterative refinement which is necessary for KBS construction. However, an *engineering* methodology for KBSs needs to provide a method for constructing an initial system from which the RUDE cycle can begin, as well as encompassing some form of *control* of the iterative process of development. The use of a KBS development methodology, without these last two features, is likely to lead to problems of documentation, maintenance and testing, all of which must be easily accomplished in an engineering methodology.

Thus, in order to engineer a KBS, the iterative features of the RUDE paradigm need to be incorporated within a structured and controllable framework.

In 8.1 it was concluded that conventional development models such as SSD, itself based on the waterfall model, were the best available basis for evolving an engineering model for KBS development. It follows that we should attempt to combine the RUDE paradigm with the waterfall model. In short, the RUDE cycle must be augmented, so that it becomes POLITE:

Produce **O**bjectives - **L**ogical/physical design - **I**mplement - **T**est - **E**dit.

The central feature of conventional development methodologies for commercial software development, which provides the basis for a *controllable framework,* is the definition of detailed design tasks and procedures with accompanying checkpoints and documentation standards for carrying out those tasks. The proposed POLITE model, based on this type of approach, is described below.

8.4 A POLITE Engineering Methodology for KBSs

A schematic view of the proposed POLITE life-cycle is shown in Figure VIII.iv. Each of the development phases is divided into two. The left side is related to conventional components and the other to the knowledge-based or cognitive elements in the system. The following sections will give an overview of the tasks associated with each of the development phases identified. For maximum clarity, each phase in the POLITE model is illustrated with a diagram. Two points should be noted, however:

- The tasks identified reflect the experience gained during the *Intellipse* project, as well as the methods for KBS development described in chapter three.

- To avoid unnecessary repetition, the level of detail is restricted.

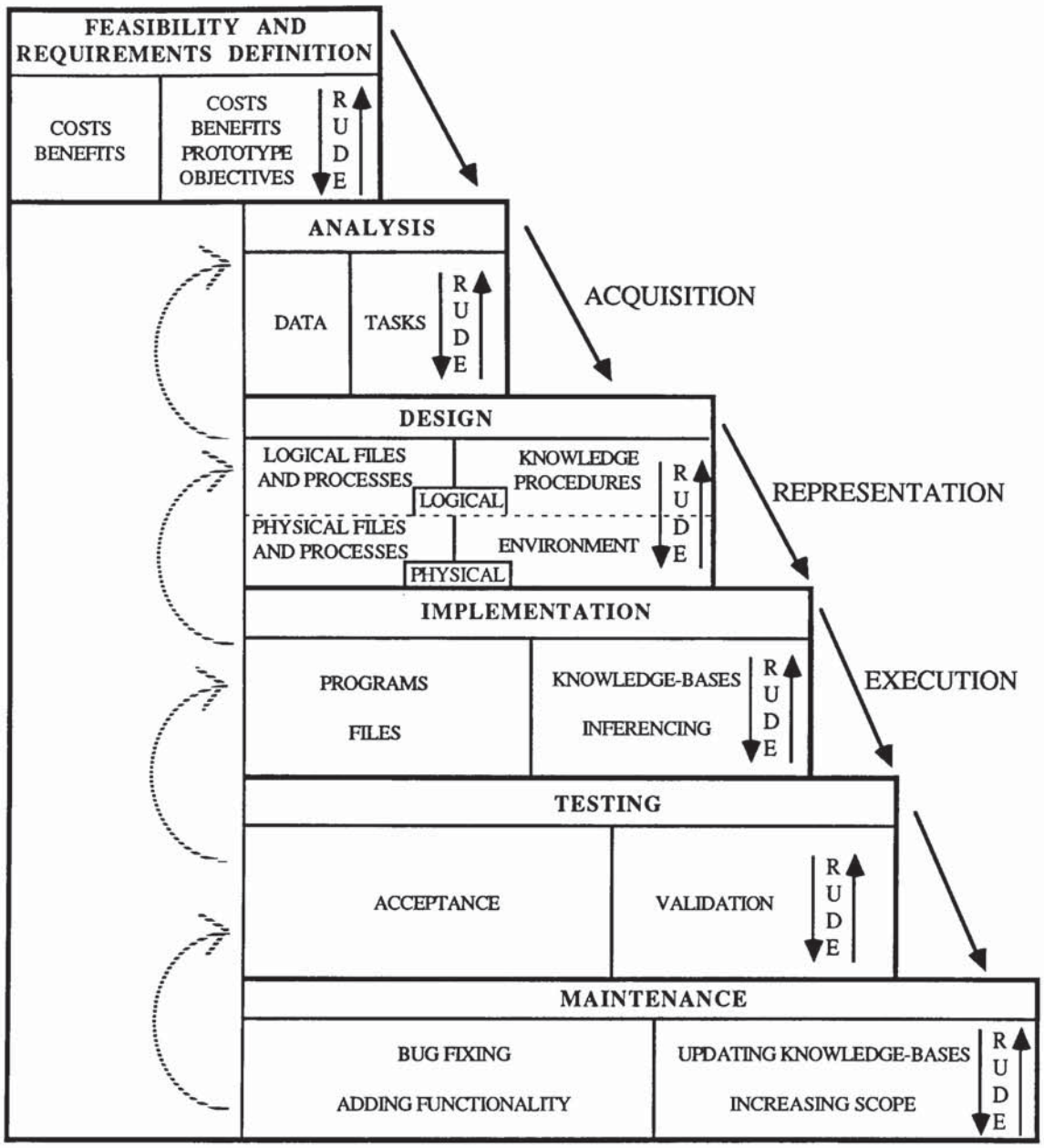

Figure VIII.iv - The POLITE Life-Cycle

The description of the POLITE model which follows deals with the development of hybrid systems comprising both conventional and KBS components. In the diagrams illustrating each phase of the model, those aspects mainly associated with KBS components are shaded. Conventional tasks are not discussed in detail as it is assumed that models like SSD are being used in these areas.

8.4.1 Performance Objectives

A key feature of the POLITE model is the production of performance objectives at the start of any project intended to produce an *operational* KBS. Performance objectives are a statement, in the problem owner's terms, of how the KBS should operate in the chosen domain.

A clearly stated set of performance objectives are essential, in order to facilitate validation of the KBS. Performance objectives are a complete statement of requirements for a KBS *at a particular time* and they provide a framework within which any RUDE development can be controlled. The performance objectives can be used as the basis for evaluation and validation at each stage in the POLITE cycle where RUDE development is appropriate.

8.4.2 Feasibility

Figure VIII.v identifies the main issues that need to be considered during a KBS feasibility study and those involved in the exercise.

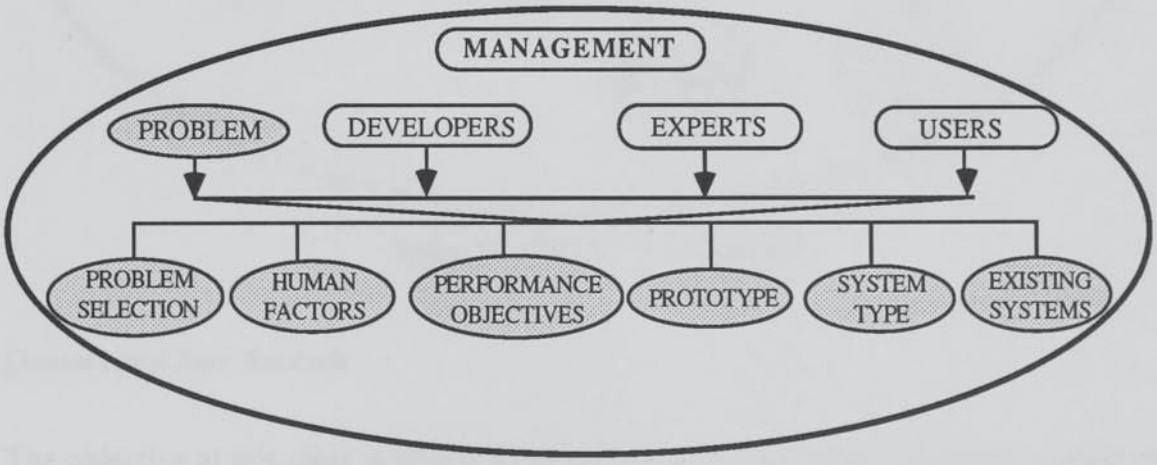

Figure VIII.v - Feasibility and Requirements Definition

Tasks and procedures are not given for this phase as these have been covered in 8.2. The diagram highlights the fact that co-operation between several discrete parts of an organisation is crucial to a successful evaluation. It is important that management oversees the feasibility exercise, since they will have to take the decision on whether to proceed to a full operational system.

8.4.3 Analysis

Figure VIII.vi shows the tasks involved in the analysis phase of the POLITE life-cycle. The main tasks during this phase are domain and task analysis, secondary feasibility, cognitive task analysis (CTA) and data analysis. (Tasks begun in the CTA phase may overlap into the logical and physical design stages - see 3.3.2.) There are many parallels here with conventional systems analysis. The main differences arise when the task being examined is a cognitive activity carried out by a human expert.

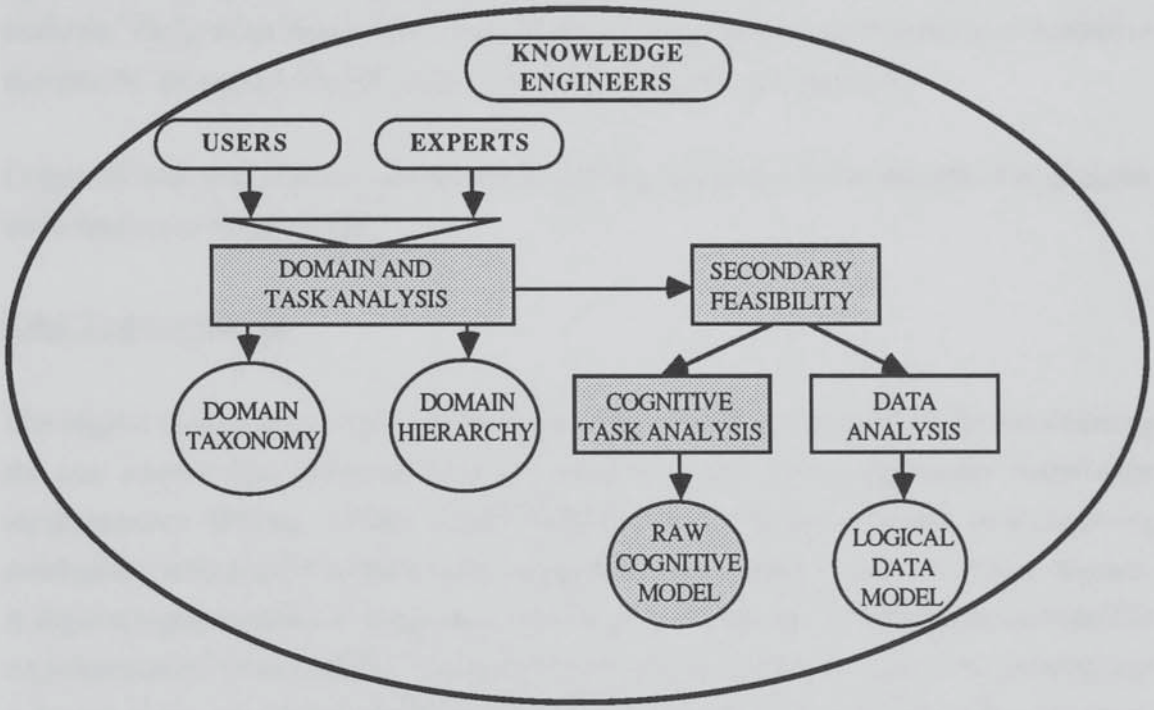

Figure VIII.vi - Analysis

Domain and Task Analysis

The objective at this stage is to take a broad look at the application domain, in order to identify and label the main tasks which the expert performs. This high level picture is successively refined by analysing each of the high level activities. Finally, a sufficient

level of detail is reached to provide an accurate guide to the expert's approach to the problem domain. The resulting description can be represented using a network structure, showing the hierarchical relationship between the activities. This exercise is particularly useful in familiarising the knowledge engineers with the application domain. This early analysis may not require much involvement of the expert, since a great deal of the information needed can often be obtained from existing manuals or text-books. In addition to the high level description of activities obtained, a domain taxonomy can be produced, which identifies and describes the key terms and concepts in the domain.

Secondary Feasibility

Depending on the level of complexity of the problem being studied, a varying proportion of the list of tasks identified will be non-heuristic and could be modelled using conventional procedural algorithms. Since the machine execution of conventional algorithms is relatively well understood, these tasks should be subjected to conventional analysis. Only those tasks which appear to rely on judgemental reasoning or cognitive analysis by the expert should be the subject of cognitive task analysis.

Cognitive task analysis and conventional analysis have been fully described in chapters three and seven respectively.

8.4.4 Logical Design

The logical design phase is shown in Figure VIII.vii and is concerned with transforming the raw knowledge, obtained from the analysis stage, into a particular knowledge representation schema. At this stage it will also be necessary to devise an inferencing mechanism which reflects the expert's approach to problems in the application domain. A *logical* representation is required at this stage, which does not depend on any specific implementation environment. This will provide flexibility in the choice of hardware and software at the physical design stage, as well as providing the basis for maintaining the KBs in the future. In an operational or commercial environment this logical or *intermediate representation* (INTEM REPRN) should be paper-based (or held as machine-based documentation within a project management environment) - another important maintenance consideration. These tasks have been described in detail in 3.3.2.

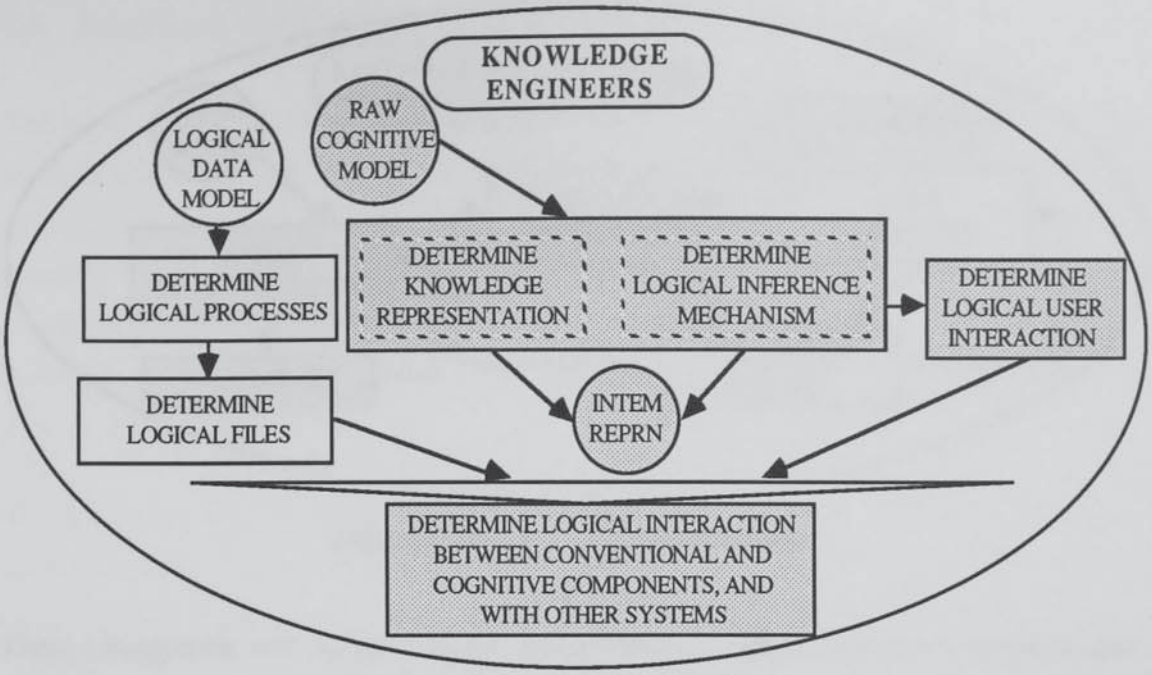

Figure VIII.vii - Logical Design

8.4.5 Physical Design and Implementation

The physical design and implementation phases are shown in Figures VIII.viii and VIII.ix respectively. This is the stage when the intermediate representation is translated into the chosen implementation environment - the *cognitive specification* (COGN SPECN). Sections 3.3.3 and 3.3.4 have discussed the tasks involved and, in particular, the need to define the user-interface and explanation facilities required. Extensive RUDE development is likely here and the logical representation of the system, obtained during the last phase, allows this iterative process to be kept under control.

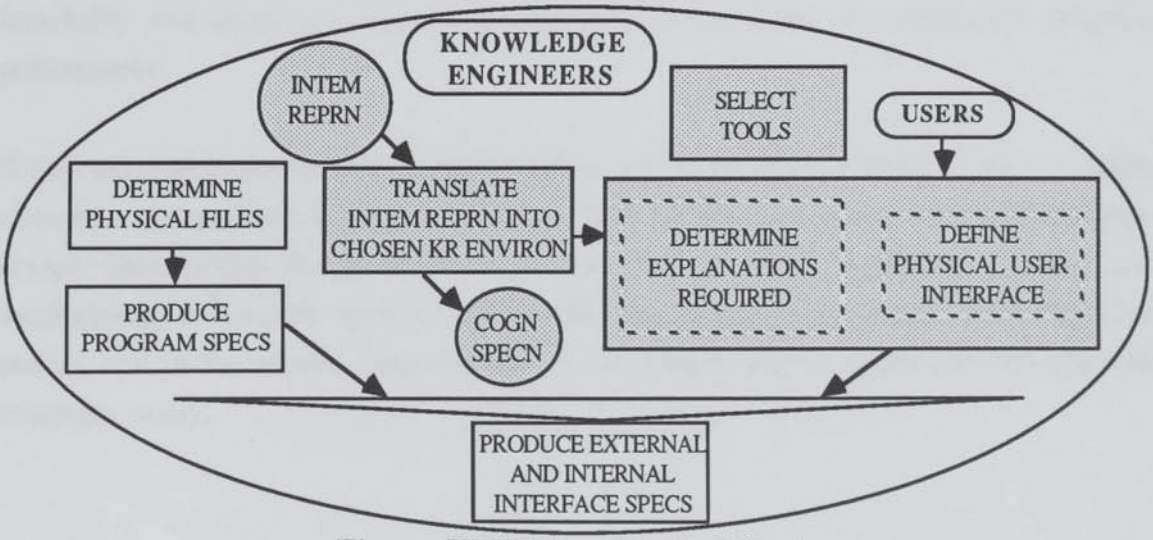

Figure VIII.viii - Physical Design

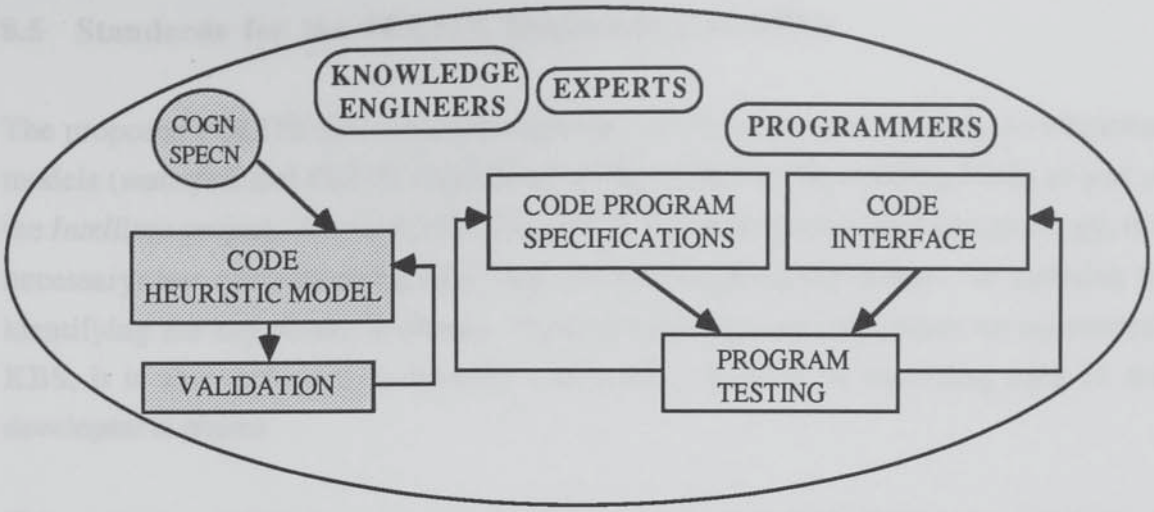

Figure VIII.ix - Implementation

Three components will require coding: the cognitive specification, conventional program specifications and any programs needed to provide interfaces with external computer systems. If shells or AI toolkits are used to implement the system, it is both possible and desirable for the expert and users to be involved at this stage. Incremental validation of the executable model and interface can then be performed.

8.4.6 Testing and Maintenance

The testing and maintenance phases in KBS development were discussed in 8.2. Formal acceptance testing of the KBS, including all external interfaces, will differ from the testing which is performed on a DP system. The probable absence of a DP-style statement of requirements means that the performance objectives, defined at the feasibility and analysis stages, must be used as a basis for measuring the system's performance.

If the rules and procedures in the application domain of the KBS are subject to regular change, a maintenance regime is needed which ensures that experts and management review periodically the performance of the system. The adoption of a structured methodology to build the system should ensure that additions or changes to the cognitive model, and in the overall functionality of the system, can be made and documented relatively easily.

8.5 Standards for the POLITE Engineering of KBSs

The proposed POLITE life-cycle paradigm is a synthesis of two existing development models (*waterfall* and *RUDE*) formulated by the author while building KBSs as part of the *Intellipse* project. A sound life-cycle model, based on a structured methodology, is a necessary, but not sufficient, first step towards engineering KBSs. In addition to identifying the key phases and tasks required to design and implement an operational KBS, it is also necessary to identify a *practical* method of executing each of the development phases.

The evolution of KBS *development standards*, based on the POLITE model, is required to assist developers in the practical construction of KBSs. The standards which are currently used to support the structured development of DP systems are the product of practical experience of many hundreds of projects. POLITE standards, *and the validation of the POLITE model itself*, can only be achieved through the use of the model on many real development projects. Figure VIII.x summarises the origins of the POLITE model and its potential evolution in the future.

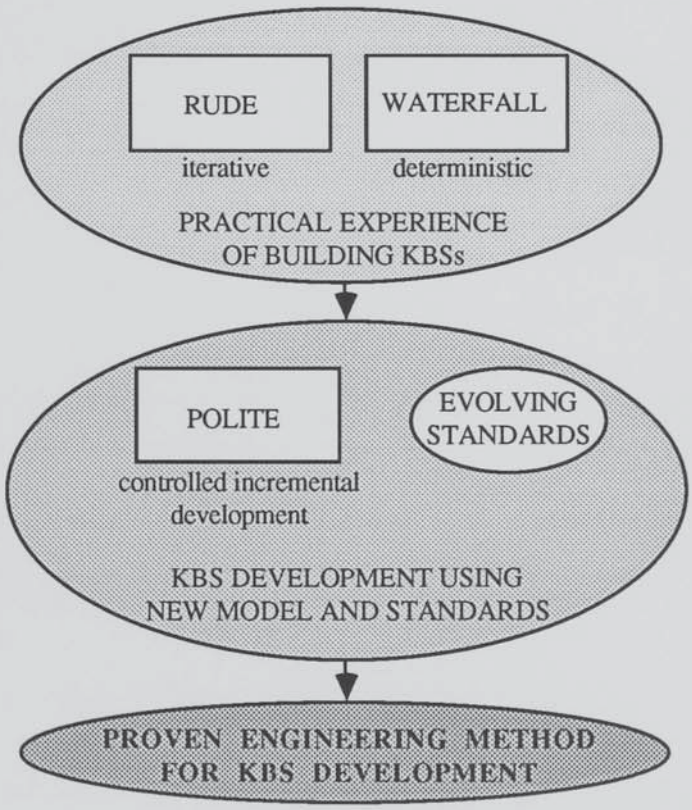

Figure VIII.x - Evolution of POLITE Engineering

8.6 Summary

This chapter has discussed the applicability of structured development methods, such as SSD, to the design and implementation of operational KBSs. It has analysed the differences between DP and KB system development and used this analysis as a basis for proposing an alternative paradigm for KBS development, based on an engineering approach - the POLITE life-cycle.

The POLITE model has the status of a hypothesis, based on some empirical evidence and limited theoretical foundations. Substantial effort is therefore required, over an extended period of time, to evaluate and validate the proposed model, and to evolve the standards vital for its practical application in an industrial or commercial environment. It should be noted that conventional DPSD models, and their accompanying standards, have evolved to their present state only after many years of use in the commercial DP environment. The next chapter describes the first steps taken to evaluate the POLITE model, in the context of the ITAM investigation identified in chapter seven.

Chapter Nine

The ITAM Investigation

Proteus
...But here's no need for musing on our part;
In the wide ocean you must make a start.
There you begin with small things of the seas,
Rejoicing even the tiniest to devour,
Until you compass growing by degrees,
The high achievement of a loftier power.

Johann Wolfgang Goethe 1749-1832 (Faust/Part Two)

Preamble

This chapter describes the initial steps taken in the ITAM investigation identified in chapter seven. In the context of this investigation, some early results of employing the POLITE model are given.

Both the ITAM investigation and the evaluation and validation of the POLITE model are longer-term studies which will need to continue beyond the Alvey *Intellipse* project. This chapter cannot therefore give as detailed or rounded a picture as the previous chapters. However, the material included gives a clear indication of how the results of the *Intellipse* project can be taken forward in the future. In addition, it allows some initial conclusions to be made about the appropriateness of the proposed POLITE life-cycle for commercial KBS development.

The main objective of this chapter is to examine the results of applying the POLITE life-cycle model to the feasibility, requirements definition, analysis and design phases of the ITAM project.

9.1 The General Objectives of the ITAM Investigation

The initial objectives of the ITAM investigation were as follows:

- To confirm the initial findings of the Designer feasibility study in the domain of DB design.
- To specify and design a KBS tool for supporting the design, optimisation and maintenance of DBMS applications in the *Total*

environment.

- To begin the evaluation and validation of the POLITE model.
- To begin the evolution of practical development standards for the POLITE model.

9.2 Approach Adopted for the ITAM Investigation

The first step taken was to conduct two separate KE sessions with BIS9 and BSC1-3. These were intended to confirm the initial findings of the Designer feasibility study, and to establish an overall set of functional requirements for a KBS support tool. As a preliminary exercise, the knowledge engineers studied the extensive documentation available on DB design in general, and the *Total* DBMS in particular. Some early domain hierarchy sheets were prepared on the basis of these sessions.

9.2.1 Initial Observations and Conclusions

The points which emerged from the first two KE sessions are summarised below:

- Acceptable performance of *Total*-based applications is achievable, in principle, using information available in the Cincom manuals, data from appropriate hardware and operating system manuals, and heuristic knowledge about DB design in general, and *Total* in particular.

- In practice, installations do not usually achieve optimal performance from their *Total* applications. Those installations, which can afford it, resort to Cincom or other external consultants to sort out their performance problems. It is possible for a *Total* expert to identify and recommend a solution to a performance problem in a few hours, on average. The use of consultants for this purpose is very expensive.

- The failure of installations to achieve optimal performance unaided is due to five main factors:

  i. The complexity and size of the paper-based manuals.
  ii. The reluctance of DBAs and others to grapple with the mathematical and statistical content of the Cincom tuning manual.

iii. The absence of a disciplined or structured method for designing database applications and tackling optimisation problems.
iv. Insufficient understanding of the general mode of operation of Cincom's products. This undermines the ability of installations to relate performance problems to potential solutions.
v. The absence of the required heuristic knowledge in the installation. This knowledge accumulates over extended periods of time and an individual end-user of *Total* is unlikely to have the experience of a DB consultant or Cincom expert.

- There is an absence of mainframe or PC-based tools to support the design and optimisation of *Total* applications.

- Significant improvements in the performance of *Total*-based applications could be achieved through relatively simple techniques. In addition, it was clear that, if a few simple design principles were adhered to at the 1st and 2nd cut physical design stage, many installations could avoid most of their performance problems.

Conclusions

Optimisation problems with *Total* can be solved by a human expert in a few hours, using a small set of heuristic rules. The heuristic knowledge is well understood and could be represented in machine form.

A PC-based tool for optimising *Total* applications, able to simulate the function of a Cincom expert or external consultant and usable by a typical *Total* installation, would be a cost-effective solution to a problem which occurs in a very wide user-base. The design, optimisation and maintenance of DBMS applications is a key operational problem in commercial DP.

It appeared at this stage to be technically feasible to represent the heuristic knowledge of the expert, the inferencing techiques employed and the external data required. The main problem would be to acquire this knowledge and structure it ready for transfer to the machine environment.

9.2.2 Detailed Objectives

The general objectives for the ITAM investigation stated earlier were re-defined as follows:

- To produce a stand-alone, PC-based KBS tool which could assist a typical *Total* installation to achieve significant performance improvements in a live *Total* application, without resorting to external, expert support.

- To assist *Total* DB designers to achieve an optimal physical design, from a normalised logical model, prior to the installation of the database, using a KBS support tool.

- To design ITAM so that it integrated conventional and knowledge-based components.

- To ensure that the knowledge-based components in ITAM required the minimum amount of keyboard input on the part of the user.

- To conduct the knowledge engineering for ITAM, with a view to generalising the knowledge about DB design, so that a basis is laid for KBS support tools for other proprietary DBMSs.

- To determine task-lists, documentation standards, check-points and deliverables enabling the ITAM investigation to be conducted as closely as possible to the approach envisaged in the POLITE life-cycle.

The prospective users of ITAM were determined as:

- Database administrators (DBAs) in *Total* installations.
- Database designers in *Total* installations.
- Cincom product consultants.

The early decision that the target environment for ITAM should be the PC was based on the considerations of portability and appropriateness for the DP environment, as well as a number of other factors, all of which were discussed in detail in 6.7.1.

9.3 The Main Functional Requirements for ITAM

The following is a high-level set of functional requirements determined for the ITAM system based on the initial KE sessions with the BIS and BSC experts:

- Passive, advisory support on translating a logical database design into an optimised 1st and 2nd cut *Total* physical design. This advice should help the user check that the logical design does not contain any obvious errors or inefficiencies.

- Facilities for displaying graphs of the various statistical formulae used for *Total* tuning which are described in the Cincom manuals. This function should also allow the user to interpolate values from the graphs and to invoke the formulae, when using other ITAM functions.

- A facility to download the output from the Cincom statistics package and other mainframe statistics packages to the PC enabling the data to be used by various ITAM functions.

- A knowledge base containing data about various proprietary hardware (e.g. disk capacities, transfer rates, block transfer sizes etc.) which can be accessed by the appropriate ITAM facilities.

- Facilities for creating user-definable "warning files" which could be used in conjunction with the *Total* statistics to check for specific problems arising in the database.

- Facilities for determining the optimum values for *Total* system parameters, based on a given logical data model, and a set of expected user-queries, prior to the installation of the application.

- Facilities for supporting the tuning of live *Total* applications using a combination of conventional and heuristic analysis of the *Total* statistics. This facility should recommend changes to system parameters with explanations of why particular changes are being suggested.

- Facilities for storing and retrieving data on specific applications or projects which have been partially designed or tuned using ITAM. This function should allow the user to analyse historically the performance of a specific *Total* file and the changes which have been made to it over a given period of time. It should also be possible to annotate this historical data with comments, indicating the reason for changes which have been made.

As the main purpose of this chapter is to give some early results of employing the POLITE model, no further details of the ITAM system are discussed.

9.4 POLITE in Action

The feasibility and requirements analysis phase established a detailed set of objectives for the ITAM investigation, together with a statement of high-level functional requirements for the proposed KBS support tool. An overall domain hierarchy sheet had also been produced which identified nine high-level tasks, to be supported by ITAM. The next step was to produce a plan and schedule for the analysis and design phases of the investigation, based on the POLITE model described in chapter eight.

It was important for the credibility of this initial test of the POLITE model that people, other than the author, attempted to employ it in a practical context. Any development methodology must, in principle, be usable by a DP environment, without the constant support of those who formulated the method in the first place. BIS expect users of SSD to be able to employ it successfully after a one week training course and some additional consultancy. Of course, chapters four, five, six and seven have shown that SSD does need, at least, some tool-support to be successful. However, the principle that an engineering methodology should be usable by different people, in a variety of installations, with the minimum of human-expert support, is a valid one.

Thus, BIS took the main responsibility in drawing up a detailed schedule of tasks and checkpoints, which were required to complete the analysis and design phases, based on the POLITE model. A part of this schedule is shown in Figure IX.i for the analysis phase (the numerical details have been left out of the diagram). Similar schedules were prepared for the logical and physical design phases.

SCHEDULE	Project: ALVEY																			Sub-Project: ITAM	
Phase: Analysis	Initials: BIS2													Date:						Page: 1 of 3	
	Total Effort	Staff	Time Intervals: Weeks																	Units: Days	
Description																					
Discuss and agree initial domain analysis																					
Task analysis for tasks 1 to 9																					
Consolidate task analysis																					
Expert review of task analysis																					
Determine KB or not (secondary feas.)																					
Define logical rules for KB																					
Conventional analysis for non-KB tasks																					
QA of analysis																					
Review meeting																					

Figure IX.i - Schedule for the Analysis Phase of ITAM

The analysis phase involved four further KE sessions with BIS9 and one session involving BSC1-3. The same basic procedure, followed during the Advisor project and Designer feasibility study, was used. The interviews were recorded and transcribed as before. Extensive domain and task analysis was done and the resulting documentation was iteratively refined in close consultation with the experts. Since BIS9 was heavily committed to BIS clients, careful planning and preparation was essential to ensure that KE sessions were productive, and that BIS9 was given the maximum notice of dates for the sessions. One of the main objectives of using a structured approach is to allow this type of planning and scheduling to be done. Figures IX.ii and IX.iii show how the planning and scheduling of the knowledge engineering were accomplished.

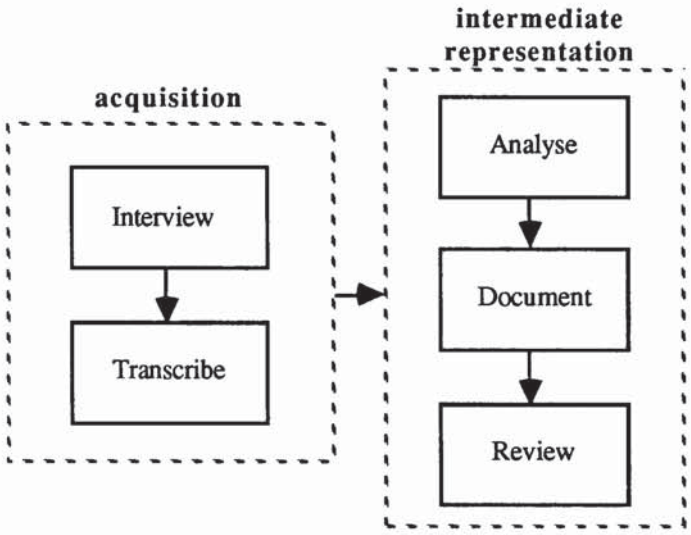

Figure IX.ii - Planning of Knowledge Engineering Sessions

TASK	KE(s)	CLERICAL	EXPERT(s)
preparation	1/2	-	-
interview	1/2	-	1/2
transcription	-	1	-
analysis	2-3	-	-
documentation	1-2	-	-
review	1/2	-	1/2

(units of days)

Figure IX.iii - Scheduling of Knowledge Engineering Sessions

Figure IX.iii illustrates the need for metrics to enable knowledge acquisition effort to be estimated. The metrics employed are based on empirical evidence gained during the *Intellipse* project.

9.5 Evolution of POLITE Standards

The KE sessions with BIS9 involved the extensive use of case studies to determine the accuracy of the diagrammatic representations produced. The sessions necessarily overlapped with the logical design phase during which rules, explanations and potential user-interactions were identified and documented. Figure IX.iv shows an example of the documentation used to record the decomposition of a high-level task, identified during the analysis phase.

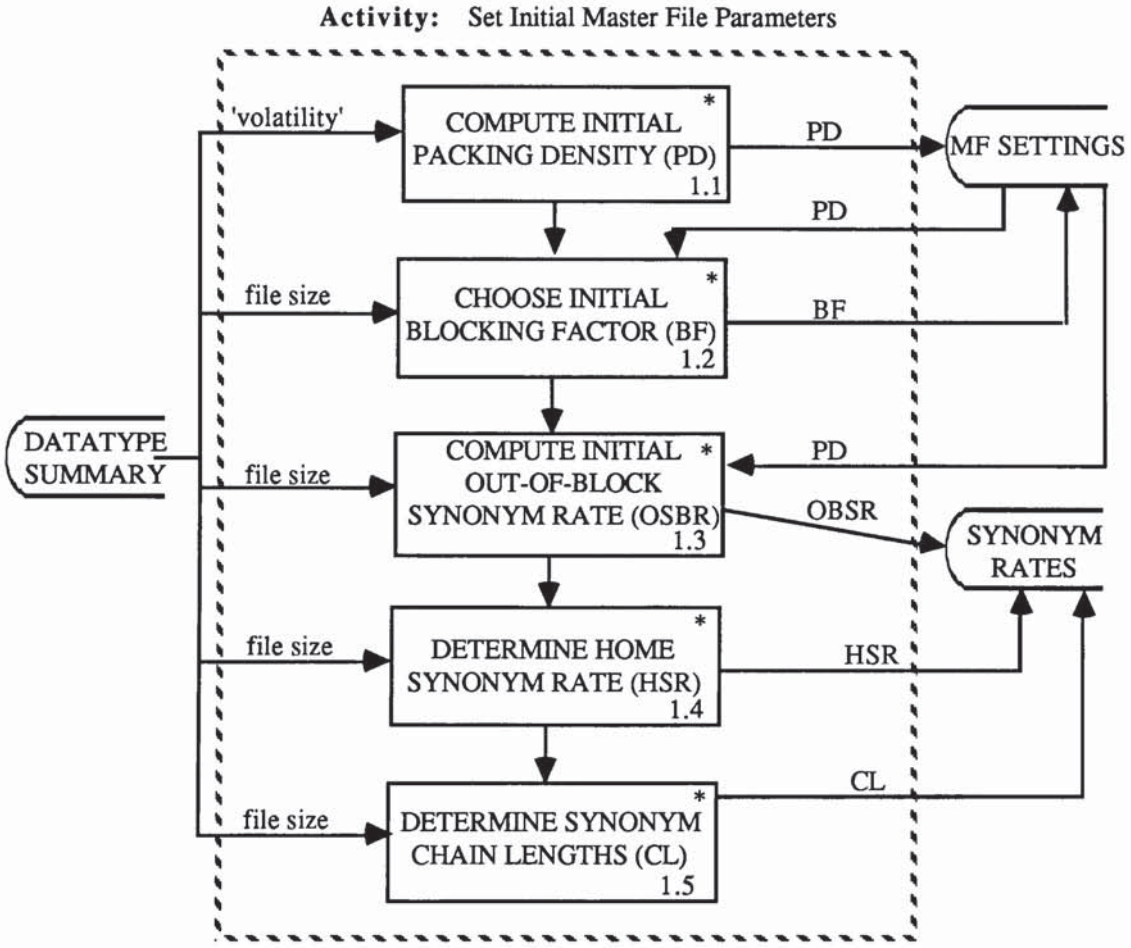

Figure IX.iv - Example of Documented Task Decomposition

The document illustrated in Figure IX.iv makes use of notation recommended in the BIS *MODUS* standards for *Structured Systems Analysis*. Martinez and Sobol[MART88] have

reported on the use of conventional systems analysis techniques for expert systems development. Martinez et al commented that the use of these techniques "...provide a way to document processes so that the experts can review them to see if they have accurately described what they do." This is in line with the experience of using these techniques throughout the *Intellipse* project, and has been commented on elsewhere in the thesis.

Addis[ADDIS85] discussed the use of conventional relational data analysis techniques (RDA) for AI systems. His discussion is mainly concerned with a comparison of conventional database theory and AI. Although techniques from RDA theory are identified, which could be used for knowledge representation in KBSs, Addis does not relate his work to the requirements of an operational KBS development methodology.

Figures IX.v and IX.vi give examples of the documentation which has evolved for recording activity descriptions and rules. This documentation, although based on existing *MODUS* standards, has had to be amended to meet the particular needs of KBS development. This adaptation of conventional standards to create POLITE KBS development standards is the process which was envisaged in chapter eight.

ACTIVITY SHEET **NO** 1.2.3 **NAME** set initial mf parameters
For Each: **When:**
Input: **Output**
Explanation:
Procedure:

Figure IX.v - Example of Documentation for an Activity Description

RULE DESCRIPTION	RULE NO 6
System: ITAM	
Activity Name: Compute initial MF parameters	**Activity Ref:** 1.2.3
Data required:	
Rule context:	
Rule:	
Source:	
Explanation:	

Figure IX.vi - Example of Documentation for a Rule Description

It should be noted that much of the adapted *MODUS* documentation illustrated has been developed by BIS independently of the author. This is an important observation, as it adds credibility to the proposition advanced in chapter eight, that the use of the POLITE model on a live KBS development project was necessary to evolve the required practical development standards.

9.6 Initial Lessons of the ITAM Investigation

The documentation evolved so far is limited and the ITAM project is a relatively small exercise, compared to a typical DPSD project. Caution is therefore necessary in drawing any firm conclusions. However, the following is a list of initial observations made on the basis of the ITAM work:

- The findings of the Designer feasibility study in the area of DB design were confirmed. The technical feasibility of designing KBS

tools to support the building, optimisation and maintenance of a DBMS application was demonstrated.

- Complex design tasks involve both heuristic and conventional elements necessitating the use of hybrid support tools in this type of domain.

- Structuring the knowledge engineering again proved successful; and the diagrammatic representation of the results of the expert task analysis was found to be valuable, not only for recording the results, but also for confirming the accuracy of the analysis with the expert. The diagrams and accompanying documentation were also valuable as a basis for iteratively refining the analysis with the expert. It should be noted, however, that the experts used were familiar with the techniques of conventional systems analysis, which also makes extensive use of diagrammatic techniques. Thus, the usefulness of this type of documentation with experts, who are unfamiliar with such approaches, needs to be investigated. This cautionary note is applicable, only where it is intended to make use of the diagrammatic material as the basic means of communication with the expert.

- The POLITE life-cycle model provided a basis for planning and scheduling the feasibility, analysis and design phases of the ITAM investigation. The model enabled BIS to draw up detailed task lists and check-points for managing the work.

- Practical standards and documentation have already begun to emerge from the ITAM work, much of which *has* been adapted from existing *MODUS* standards. The fact that it has been adapted from *MODUS*, however, also confirms that conventional techniques do need to be *adapted* to suit the particular requirements of KBS development.

9.7 Summary

This chapter has described some early results of work arising out of the main *Intellipse* project. This work has been addressing both the *KBS for SE* and the *SE for KBS*

themes. The feasibility of building KBS support tools in the area of DB design has been confirmed, and the initial results of applying the POLITE life-cycle model to this investigation suggests that the model will have practical applicability in the future. Development standards have also begun to emerge, adding extra weight to the validity of the POLITE model. BIS are satisfied with the results to the extent that they will continue to pursue the development of commercial standards for operational KBS development based on the POLITE life-cycle.

Chapter Ten

Discussion and Conclusions

> ...Rule XI
> If, after gaining intuitive knowledge of several simple propositions, we are to draw some further inference from them, it is useful for us to run through them in a continuous and uninterrupted movement of thought, to reflect on their interrelations and to form, so far as we can, distinct conceptions of several at once. For this adds much to the certainty of our knowledge, and it greatly increases the scope of our mind.
>
> Descartes 1596-1650

Preamble

The final chapter will discuss some general issues relating to both the *KBS for SE* and *SE for KBS* themes. It will also indicate to what extent the author's work has contributed to existing knowledge in these areas. Potential directions for future work are also identified.

10.1 KBS for SE

In chapter three, an analogy was drawn between DPSD and the traditional engineering disciplines, like civil engineering. Figure X.i summarises some of the common features of these two domains. The most significant difference between the disciplines, which is not evident from the diagram, is their difference in maturity. Since the engineering disciplines have been evolving over a greater historical period than DPSD, they have many more established traditions. Other fields, like medicine and the law, show a similar domain maturity. For example, we would not expect a trainee engineer to be given the responsibility for designing a building, or an articled clerk to handle a major criminal prosecution. In medicine, a solid hierarchical training structure attempts to ensure that young doctors are not expected to make judgements on patients, which ought to be the province of their more experienced colleagues. However, in commercial DP, even the most inexperienced programmers and analyst/designers find themselves in a position of significant responsibility in relation to major application projects. No other industry seems to expect such inexperienced practitioners to show such a degree of competence so early in their careers.

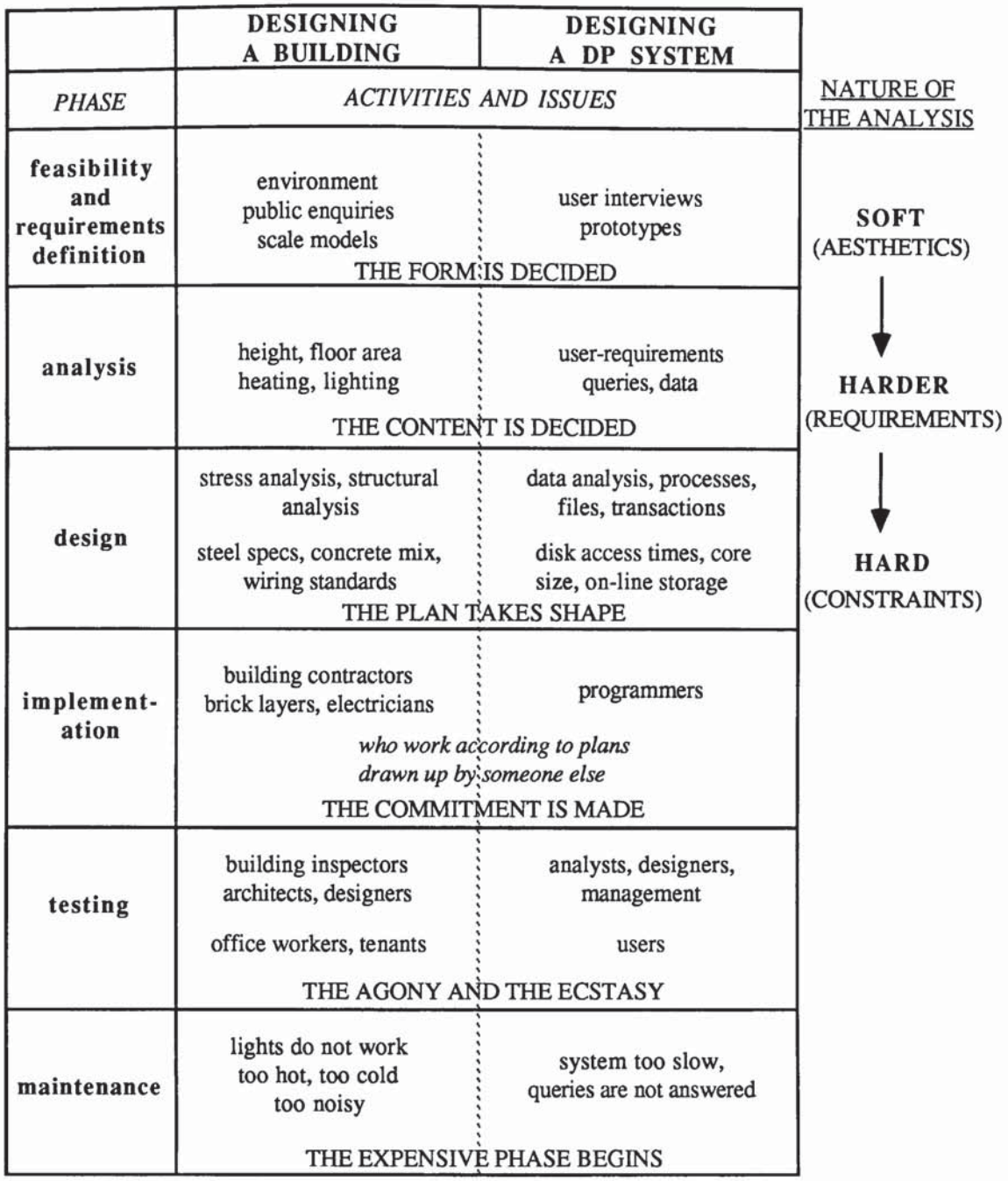

Figure X.i - A Comparison Between Civil Engineering and DPSD

This situation arises because commercial DP does not yet have the tools, methods or the number of experienced analysts and designers to cope with the demands being placed on the DP functions of many companies and organisations. Hence the two motive forces in the industry: the drive for more productivity through the use of powerful CASE tools; and the drive throughout the history of the DP industry to automate as much of the process of systems development as possible. Research into the application of AI to software engineering and the development of formal algebraic methods for specifying

software address both of these concerns.

The common factor linking the AI and formal methods research is the idea that the systems development process can be represented in some abstract formalism which would be applicable to any application domain - a representation which would only contain knowledge about design as a *generic activity*, rather than as a process always applied to specific areas of human activity. Yet the traditional engineering disciplines do not use such a formalism, except in so far as mathematics is used to solve specific problems like stress analysis, aerodynamics and so on. But even in these areas the mathematical notation is instantiated very early in the design cycle with data relating to the particular domain of application.

The research reported in this thesis has been based on extensive interviews with different DP designers. In every case, application domain knowledge has been the decisive influence in most of the design activities identified and analysed. It is clear therefore that tools to support DP systems development need to encompass as fully as possible application-specific design knowledge. It was also observed that the designers did not manipulate their knowledge using a formal mathematical algebra. The notation used was a form of structured English, whose vocabulary was loaded with technical terms specific to computer systems design. Extensive use was also made of diagrammatic notations. This leads to another major problem of DP: communication between users of the technology, and those who design and implement application systems.

End-users employ natural language loaded with their own domain-vocabulary. The problem for the designer is to re-formulate this *natural* specification into one which can be used as a basis for implementing a DP system. Because of the unavoidable time-lag (while coding takes place), between completing a design and specification and executing the code based on that specification, two problems arise. Firstly, users cannot see early enough the implications of their requests for functionality - expressed in their language , but communicated to the programmer in another notation. Secondly, by the time the programs are executed, the users' requirements may have changed, because of changes in their operational domain.

AI research is addressing the problem of communication between users and developers by looking at techniques for parsing and analysing natural language, so that that this can be used as the main basis of communication. AI research seeks specification languages as close as possible to the users' natural means of communication. If these *natural*

specification languages could also be used as implementation languages, by generating executable code automatically from this natural specification, then systems could be implemented very quickly. This objective is the link with formal methods research which seeks *algebraic specification languages* which could be used to animate system designs (to allow users to see prototypes early), and as a basis for automatic code generation.

Formal specification languages would have another potential advantage: if code is generated automatically on the basis of algebraic specifications, then the executable code can be mathematically proven to be a "correct" implementation of the specification. This does not solve the problem of ensuring that the original specification was appropriate from the users' point of view, and hence the importance of AI-based natural specification languages.

The introduction of higher level languages into DP like Fourth Generation tools, application generators and advanced relational DBMSs, is an attempt to bring the specification language closer to the users' language. However, these tools are still based on a semi-formal syntax. There remains the need for systems to be designed to achieve operational requirements of speed and size, and for programmers to code the systems. 4GLs can enable the implications of system requirements to be looked at earlier in the development cycle - in terms of screen layouts, query layouts and so on - *but they do not help to measure accurately the timing and sizing of a fully operational and loaded commercial database.* These tools therefore do not remove the necessity for systems to be *designed*, nor do they enable the inexperienced designer to build systems to commercial and industrial standards.

Figure X.ii gives a speculative view of the way commercial DP may develop in the future, if present trends continue. The emphasis on the automation of, and on AI support for, DPSD has historically been at the implementation-end of the life-cycle spectrum, as distinct from the analysis-end. As these techniques have become more sophisticated, the emphasis has shifted towards the analysis phase of the life-cycle. This reflects the ultimate objective of AI in software engineering, identified in chapter three, that of generating executable computer systems automatically from user-specified natural language requirements statements. This is clearly a long-term goal and depends on very significant advances in both natural language processing and formal specification languages. Figure X.iii summarises the trends in the development of CASE tools and methodologies towards this long-term objective.

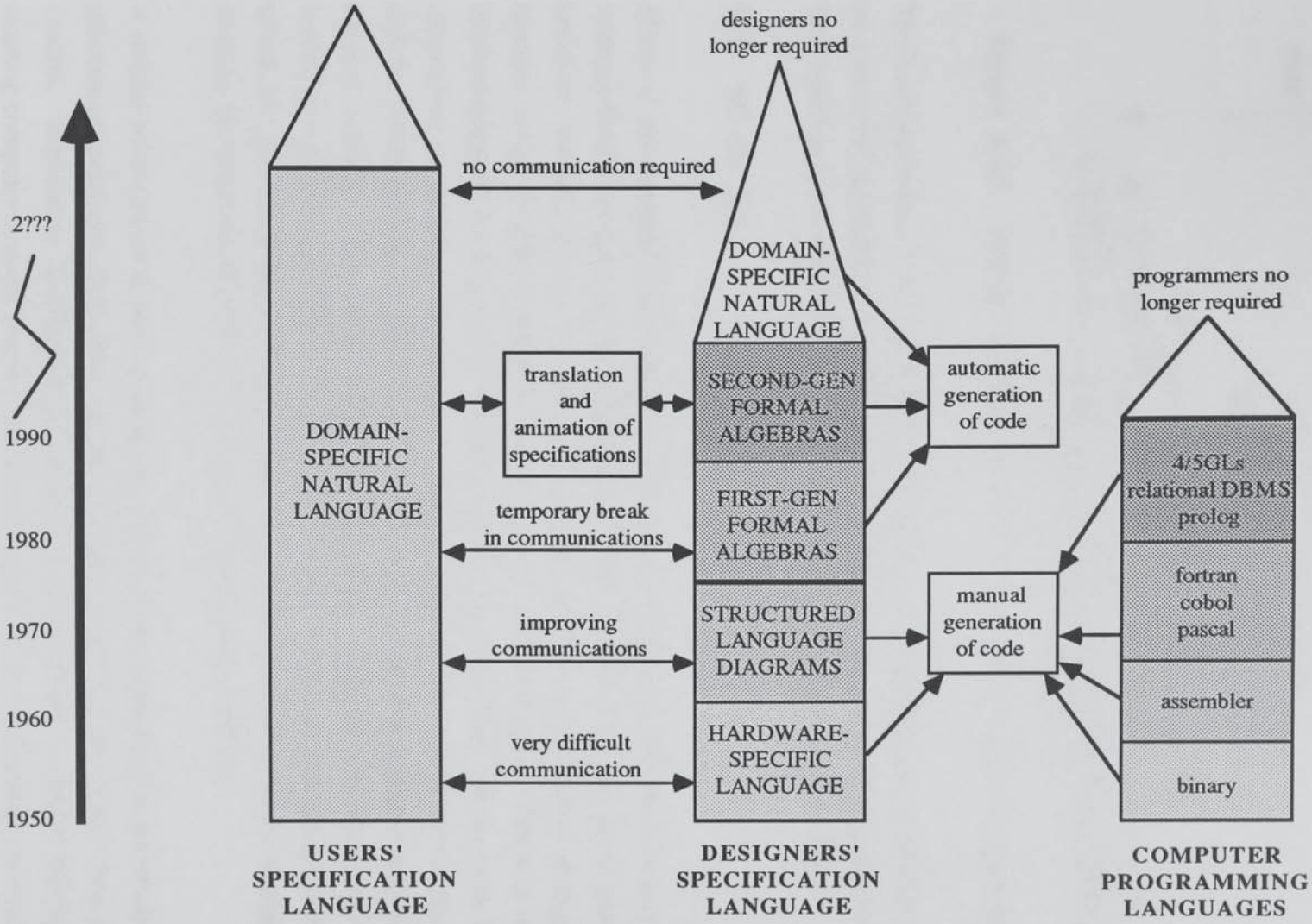

Figure X.ii - A Speculative View of the Future of DPSD

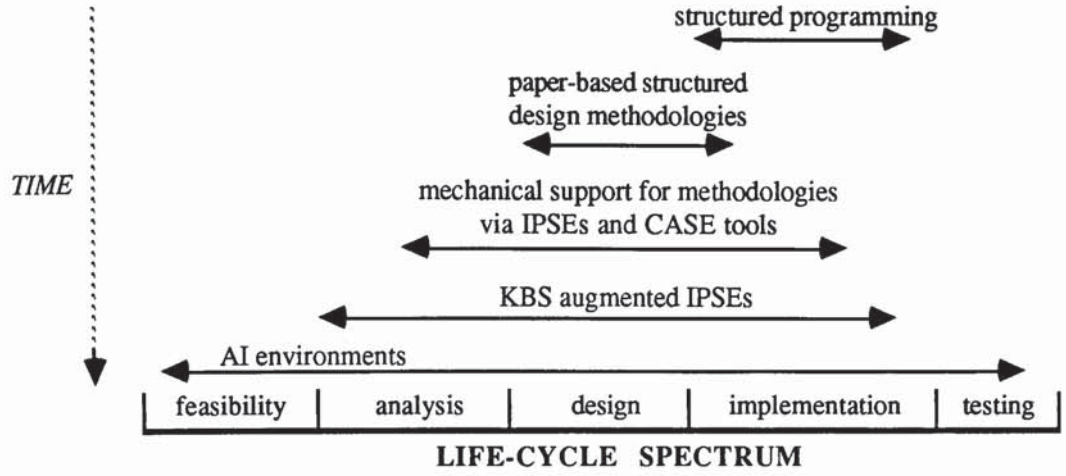

Figure X.iii - Trends in Development of DP Tools and Methodologies

The *Intellipse* research has addressed the short-term goal of KBS tools for DPSD, based on a model of development which is still largely dependent on human practitioners. The final conclusions in relation to this *KBS for SE* theme are summed up in 10.3.

10.2 SE for KBS

Much of the reported KBS work has been of a research or prototype nature. The systems built have been small, stand-alone KBSs. There is a clear trend emerging, however, towards the use of KBSs in complex application domains, to support a broader range of tasks currently carried out by human experts. There is also an increasing awareness that operational KBSs need to integrate fully with existing DP and other computer systems. For example, an important area for development in the future will be the integration of KBSs with management information systems and decision support systems. This will enable company and organisational *knowhow* to be combined with the large databases of sales, marketing, technical and other information, which now exist in commerce and industry. This process will allow senior management to make far more effective use of their strategic information resources.

A related development is the increasing awareness, in the traditional manufacturing and processing industries, that KBSs can be a powerful tool for use in real-time process control. Significant developments are already taking place to embed KBSs inside existing computer systems used for monitoring continuous manufacturing processes.

The trends described require that operational KBSs be built to the same standards as

those expected in other operational computer software. This, in turn, means that KBS development projects need to be managed, using similar techniques to those used in the construction of conventional computer systems. An engineering basis for KBS development will also enhance the credibility of the technology. This is important, since there is still significant doubt in industry and commerce about the applicability and reliabilty of KBS techniques.

As operational demands for the use of KBSs increase, there will be a move away from the situation in which much of the work in the field has been *technology-driven.* The technology-driven nature of much KBS work is another manifestation of the academic origins of the technology. Many KBSs are the result of a process which can be characterised as a *specific technological solution looking around for a problem to solve.* It will be critical to the future success of KBS technology that its operational use is *problem-driven*, not technology-led. The drive for operational systems may also help dispel the widely held view that KBSs are synonymous with small PC systems using "if-then" production rules for their knowledge representation. There can be a role for KBSs, even where the human expertise involved cannot be represented using a simple production rule schema.

10.3 Contributions Made to Knowledge in the Field

The *Intellipse* project work reported in the thesis has contributed to knowledge in the following areas:

KBS Support for DPSD The concept of, and architecture for, a knowledge-based environment to support a widely used structured design methodology has been proposed. Although the *Intellipse* system is related specifically to BIS's SSD methodology, the fact that SSD is similar to other commercially used methods suggests that the system will have general applicability. The *Intellipse* concept envisages supporting the design process, through the use of an integrated set of KBS and non-KBS tools each addressing specific tasks in the design cycle.

An Interim Report[ALVEY88] of the Alvey Information System Factory (ISF) project, discussed in 3.4.1, was published in January 1988. The report proposes an "Architecture for Information System Factories".

Section 6.1 of the report states that:

> "The proposed architecture for ISDFs..[Information Systems Development Support Facilities-JLB]..is based upon the concept of an activity in the ISF and the provision of explicit support for that activity in an ISDF activity...Each activity can itself be broken down into sub-activities, which are themselves activities within a method that implements the overall activity. Activities may be structured, in which case they are explicitly treated as a network of activities at a lower level; or terminal, in which case they constitute terminal nodes in the decomposition of activities. A terminal activity may be complex, but its structure is not controlled by the ISDF; rather it is seen as an individual tool." (Page 15.)

This type of tool-support, proposed for individual activities in the ISF, bears great similarity to the role of APMs in the *Intellipse* system described in 5.3.2. Thus, the ISF report offers a degree of independent validation of the *Intellipse* architecture.

Chapters six and seven describe work done to begin the process of evaluating the appropriateness of the *Intellipse* concept. In particular, a study has been made of the SSD methodology, to identify individual design activities and assess the degree to which they could be supported by knowledge-based techniques. This investigation has identified potential areas for *rule-based* KBS support. In addition, it identified the SSD areas where this type of support is *not* feasible, and indicated how other types of KBS support could be developed. These results may help work directed towards the long-term goal of the Alvey Directorate (discussed in 1.2) to develop a third generation IPSE incorporating knowledge-based components.

The Designer feasibility study described in chapter seven also proposed that existing CASE tools, like the BIS/IPSE, could be improved by augmenting them with knowledge-bases similar to those developed for the Advisor system. In this way, formerly *mechanical* support tools could begin to offer more *expert* support. In particular, a proposal to incorporate the Advisor knowledge bases into the BIS/IPSE has been discussed. IPSE/Advisor would represent an intermediate system between existing mechanical support environments and the large-scale knowledge-based support environment, represented by the *Intellipse* system. Since the *Intellipse* system can only be built in the long-term, due to the very large number of tasks involved in the complete design life-cycle, an intermediate system is a necessary objective.

Chapter seven also identified the DB design domain as a key problem in commercial DP. The chapter demonstrated, however, that DB design provides a sound technical basis for

the development of KBS support tools. Chapter nine went on to discuss the analysis and design phases for the ITAM system, which is intended to provide KBS support for the design, optimisation and maintenance of a particular proprietary DBMS.

There has been some reported work on research into KBS support for software engineering in 1988. For example, Symonds[SYM88] discussed the CASE/MVS project, which is looking at the use of knowledge-based techniques to build a CASE environment to support the IBM *MVS/XA* operating system environment. He observed that:

> "Synthesizing the knowledge-engineering and software engineering disciplines can lead to a powerful CASE environment...A practical approach is to implement the knowledge-engineering technology incrementally rather than try to implement it all at once." (Page 56.)

Paolo Puncello[PAO88] et al reported on the ASPIS Esprit project. This project is also researching into knowledge-based CASE environments. Their work has been focused on the analysis end of the life-cycle. The ASPIS research has addressed the problem of domain knowledge.

> "The most useful heuristics...are those that relate to the application domain. For example, it is surely more useful to have alternative functional decompositions of the system at hand instead of just general domain-independent heuristics. Even more useful are alternative decompositions based on parameters such as data and results. The domain knowledge, then, is an enhancement of the methodical knowledge." (Page 60.)

Both of the observations quoted above corroborate key conclusions made on the basis of the *Intellipse* project work.

Development of Operational KBSs The *Intellipse* project necessitated a thorough study of both DPSD methods and the methods used to build KBSs. Chapter eight discussed the applicability of conventional DPSD techniques to the design and implementation of operational KBSs. The key differences between the methods used for DP and knowledge-based systems development have been identified. Using this analysis as a basis, an adapted conventional life-cycle model for KBS development has been proposed - the POLITE model. There is very little reported work on engineering methodologies for KBS development and, although the POLITE model is based on limited empirical evidence and theoretical foundations, it does provide a basis for taking some initial steps towards a life-cycle for KBS development which can produce KBSs meeting industrial and commercial operating standards.

Chapter nine described the initial results of evaluating the POLITE model, in the context of a real KBS development project, and it illustrated some of the development standards which have begun to emerge. However, it should be borne in mind that the long-term value of the POLITE model can only be assessed, after it has been used on a large number of development projects.

10.4 Future Work

There are five areas of future work suggested by the results of the *Intellipse* project so far:

i. Continued development of the Advisor knowledge-bases and, in particular, validation and enrichment of the knowledge using the expertise of BIS experts.

ii. Incorporation of the existing Advisor knowledge-bases into the BIS/IPSE and the development of further Advisor knowledge-bases containing process sheet and transaction profile templates, as envisaged in 7.3.2.

iii. Further research into the potential for knowledge-based support for SSD tasks, particularly in the areas of structuring processes, logical design, detailed design and testing.

iv. Continued development of the ITAM-KBS tools for supporting DBMS applications, and generalisation of this work to support other proprietary DBMS products.

v. Continued evaluation and validation of the POLITE life-cycle for KBS development, with a view to the evolution, in time, of a comprehensive set of development standards for operational KBSs.

The first three areas of work are necessary, of course, to provide a basis for implementing the full *Intellipse* system in the future.

10.5 Final Summary

The overall objectives[BIS84] of the *Intellipse* project were quoted in 1.3. They were to:

> "- demonstrate the potential for the use of IKBSs in systems development,
>
> - provide a basis for tools for systems design,
>
> - promote industry and academic awareness of the capabilities of IKBS in systems development,
>
> - produce a framework of techniques and tools which could be incorporated in the development of an IPSE." (Page 3.)

The work on the *Intellipse* concept, Advisor and the Designer feasibility study have addressed the first, second and fourth objectives above. Although only limited progress has been made to *implement* the proposals for "IKBSs in systems development", the potential of using KBSs has been clearly demonstrated, and a sound basis has been laid for the implementation of KBS support tools in the near future. A framework also now exists for incorporating KBS techniques into the BIS/IPSE.

The four published papers[BADER87a], [BADER87c], [BADER87d], [BADER88] reporting the work of the *Intellipse* project have helped to promote "awareness of the capabilities of IKBS in systems development" - the third of the objectives above.

The specific aims of the thesis were expressed in 1.9. They were to show that:

i. CASE tools for DP system design will need to offer more than mechanical support to be effective. CASE tools augmented with *knowhow* about DP design can be built to provide more active support to inexperienced designers;

ii. KBS techniques can be applied successfully in the domain of DP systems design to support the activity of expert and non-expert practitioners, but there are key areas within DP systems design which are not amenable to KBS approaches, based on the use of if-then production rules;

iii. KBSs and conventional software should be fully integrated in future computer-based systems designed to solve problems in highly complex domains;

iv. Conventional software engineering techniques should be used to enrich KBS

technology and help the latter become an established technique within the DP community;

v. KBSs can and should be built using an amended conventional SDLC and by adopting an appropriate structured development methodology. KBSs built in this way are much more likely to have the robustness and reliability demanded in an industrial or commercial environment.

The discussion in this chapter indicates that these aims have largely been fulfilled. Of the two themes which have dominated the thesis, the *KBS for SE* work has provided a basis for development of KBS tools in the future. However, it will require significant investment by BIS in order to turn the results of the *Intellipse* project into deliverable CASE tools for the commercial DP environment.

The *SE for KBS* theme did not take up as much time in the project, and it was never the main priority of the work. However, since the relevance of KBS technology to modern commercial and industrial problems is being increasingly appreciated, the proposed POLITE life-cycle model, and accompanying development standards, may prove to have the greater significance in the short to medium-term.

The synthesis of software engineering and artificial intelligence has been heralded as the solution to the age-old problems of the DP industry. Ironically, the application of traditional software engineering methods to KBS development may turn out to solve many of AI's problems, but it may be some time before AI solves as many of the problems of its less glamorous partner.

REFERENCES

The number(s) below each reference indicate the page(s) where the reference is cited in the main body of the thesis.

ADEL85 Adelson, B., Soloway, E., *The Role of Domain Experience in Software Design*, IEEE Transactions on Software Engineering, Vol. SE-11, No. 11, November 1985, pp 1351-1360.
146

ADDIS85 Addis, T. R., *Designing Knowledge-Based Systems*, Kogan Page Ltd., London, 1985.
181

AGRE86 Agresti, W., W., *New Paradigms For Software Development*, IEEE Computer Society Press, Washington, USA, 1986.
65, 157

ALPE87 Alperin, L., B., Kedzierski, B., I., *AI-Based Software Maintenance*, Proceedings of the 3rd Conference on Artificial Intelligence Applications, Kissimmee, FL, USA, February 1987, pp 321-326, IEEE.
64

ALVEY82 *The Report of the Alvey Committee*, Her Majesty's Stationery Office London, 1982.
15, 17, 18, 20, 82, 158

ALVEY83 *Software Engineering Strategy*, The Alvey Directorate London, 1983.
18

ALVEY85 *Alvey Programme, Annual Report 1985 - Poster Supplement*, The Alvey Directorate London, 1985.
20

ALVEY87a *ISF Alvey Study-Project Brochure*, The Alvey Directorate, London, UK, 1987.
57

ALVEY87b *Alvey Programme Annual Report 1987, Poster Supplement*, The Alvey Directorate, London, October 1987.
67

ALVEY88 *Alvey Information Systems Factory Study, Internal Working Note, ISF/15.1, Architecture for Information Systems Factories,* (Author Gavin Oddy), The Alvey Directorate, London, January 1988.
191

AVIG87 Proceedings of the 7th International Workshop on Expert Systems and Their Applications, Volumes I and II, Avignon, France, May 1987.
44

BADER87a Bader, J., Hannaford, D., Cochran, A., Edwards, J. S., *Intellipse: Towards Knowledge-Based Tools for the Design of Data Processing Systems*, Information and Software Technology, Vol. 29, No. 8, October 1987, pp 431-439.
69, 195

BADER87b Bader, J., *IKBS in Support of Software Engineering and Vice Versa*, Alvey Conference Report, The Alvey Directorate, London, UK, 1987, pp 10-12.
58

BADER87c Bader, J., Hannaford, D., Cochran, A., Edwards, J. S., *Intellipse: A Knowledge Based Tool For an Integrated Project Support Environment*, Proceedings of the International Conference On Automating Systems Development, Leicester Polytechnic, UK, April 1987.
195

BADER87d Bader, J., Cochran, A., Edwards, J., Hannaford, D., *Intellipse: A Knowledge-Based Tool to Support the Design of Commercial Data Processing Systems*, Proceedings of the 3rd International Expert Systems Conference, London, June 1987, Publ. Learned Information (Europe) Ltd., UK, pp 363 - 375.
195

BADER88 Bader, J., Edwards, J. S., Harris Jones, C., Hannaford, D., *Practical Engineering of Knowledge Based Systems*, Information and Software Technology, to appear in Vol. 30, June 1988.
195

BALZ78 Balzer, R. M., Goldman, N., Wile, D., *Informality in Program Specifications*, IEEE Transactions in Software Engineering, SE-4(2), pp 94-103, 1978.
56

BALZ83 Balzer, R., Cheatham, T. E., Green, C., *Software Technology in the 1990's: Using a New Paradigm*, Computer, November 1983, pp 39-45.
56, 68

BALZ85 Balzer, R., *A 15 Year Perspective on Automatic Programming*, IEEE Transactions in Software Engineering, SE-11(11), November 1985, pp 1257-1268.
69

BARB86 Barbacci, M., *The Software Factory Project at the Software Engineering Institute*, AFIPS Conference Proceedings, Vol. 55, 1986, National Computer Conference, Las Vegas, NV, USA, June 1986, pp 94-95.
58

BARR82 Eds. Barr, A. and Feigenbaum, E. A., *The Handbook of Artificial Intelligence Volume II*, Pitman Books Ltd., London, 1982.
34, 38, 54

BARS79 Barstow, D., *Knowledge-Based Program Construction*, Elsevier, Amsterdam, 1979.
54

BARS87 Barstow, D., *Artificial Intelligence and Software Engineering*, Proceedings of the 9th International Conference on Software Engineering, Montery, USA, 1987, pp 200-211.
146

BART81 Bartels, U., Olthoff, W., Raulefs, P., *APE: An Expert System for Automatic Programming from Abstract Specifications of Data Types and Algorithms*, Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, Canada, August 1981, pp 1037-1043.
56

BAS84 Basden, A., *On the Application of Expert Systems*, in Developments in Expert Systems, Ed. Coombs, M. J., Academic Press, London, 1984, pp 59-75.
41

BASI85 Basili, V., R., Ramsey, L., Arrowsmith-P: *A Prototype Expert System for Software Engineering Management*, Proceedings of an Expert Systems in Government Symposium, McLean, VA, USA, October 1985, pp 252-264, IEEE.
63

BENN88 Benner, K., T., *The Knowledge-Based Software Assistant*, Proceedings of an International Workshop on Knowledge-Based Systems in Software Engineering, University of Manchester Institute of Science and Technology (UMIST), Manchester, UK, March 1988, pp C1-1 to C1-11. Publ. by The Information Systems Research Group, Department of Computation, UMIST. Proceedings will also appear in Knowledge Based Systems (Butterworths Publications Ltd., Guildford, UK) in 1988.
57

BERR86a Berry, D. C., Broadbent, D. E., *Expert Systems and the Man-Machine Interface-Part One*, Expert Systems, Vol. 3, No. 4, October 1986, pp 228-231.
49

BERR86b Berry, D. C., Broadbent, D. E., *Expert Systems and the Man-Machine Interface-Part Two: The User Interface*, Expert Systems, Vol. 4, No. 1, February 1987, pp 18-27.
49

BETTS88 Betts, B., *Vax Tuning Software Grows from an Expert System Shell*, DEC User, January 1988, p 69.
144

BIS84 *Proposal to the Alvey Directorate for Development of IKBS Software Tools*, BIS Applied Systems Ltd., Birmingham, November 1984.
19, 22, 26, 27, 195

BIS86 *The BIS/IPSE*, BIS Applied Systems Ltd London, 1986.
21

BLUM86 Blum, B. I., Sigillito, V. G., *An Expert System for Designing Information Systems*, Johns Hopkins APL Technical Digest, Vol. 7, No. 1, 1986, pp 23-30.
59

BOB86 Bobrow, D. G., Mittal, S., Stefik, M. J., *Expert Systems: Perils and Promise*, Communications of the ACM, Vol. 29, No. 9, September 1986, pp 880-894.
41

BOEH76 Boehm, B., W., *Software Engineering*, IEEE Transactions on Computers, Vol. C-25, No. 12, December 1976, pp 1226-1241.
66, 157, 162

BOEH81 Boehm, B., W., *Software Engineering Economics*, Prentice-Hall, NJ, USA, 1981.
62

BOUZ85 Bouzeghoub, M., Gardarin, G., *Database Design Tools: An Expert System Approach*, Proceedings of the 11th International Conference on Very Large Databases, Stockholm, Sweden, 1985, pp 82-95.
136

BRAT75 Bratman, H., Court, T., *The Software Factory*, Computer, May 1975, pp 28-37.
57

BREU86 Breuker, J. A., Wielinga, B. J., Hayward, S. A., *Structuring of Knowledge Based Systems Development*, Esprit '85: Status Report of Continuing Work, Elsevier Science Publishers B.V. (North-Holland), 1986.
160

BUCH69 Buchanan, B. G., Sutherland, G. L., Feigenbaum, E. A., *Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry*, in Machine Intelligence 4, Eds. Meltzer, B., Michie, D., Edinburgh University Press, pp 209-254.
38

BUCH86 Buchanan, B. G., *Expert Systems: Working Systems and the Research Literature*, Expert Systems, Vol. 3, No. 1, January 1986, pp 32-51.
44

CAR83 Carey, T., T., Mason, R., E., A., I*nformation System Prototyping: Techniques, Tools, and Methodologies*, INFOR-The Canadian Journal of Operational Research and Information Processing, Vol. 21, No. 3, 1983, pp 177-191. Also in Agresti, W., W., New Paradigms For Software Development, IEEE Computer Society Press, Washington, USA, 1986, pp 48-57.
65

CERI86 Ceri, S., Gottlob, G., *Normalization of Relations and Prolog*, Communications of the ACM, Vol. 29, No. 6, June 1986, pp 524-544.
136

CHAP86 Chapman, P., Seiler, H., *Estimating Software Development Costs Using Expert System Shell and COCOMOx Knowledge Bases*, Proceedings of the 5th Annual International Conference on Computers and Communications: PCCC'86, Scottsdale, AZ, USA, March 1986, pp 568-591, IEEE.
62

COCH81 Cochran, A. J, *Vocational PhDs: Aston's IHD Scheme*, University of Aston in Birmingham, May 1981.
20, 21

CODD70 Codd, E., F., *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM, Vol. 13, No. 6, June 1970, pp 377-387.
133, 155

CON88 Condon, R., *Is There Madness in the Method?*, Computer News, January 14, 1988, p 13.
81

CONN85 Connor, D., *Information System Specification and Design Road Map*, Prentice Hall.
71

CORD85 Corder, C. R., *Ending the Computer Conspiracy*, McGraw Hill, UK, 1985.
82, 154

CRI86 *The CRI Directory of Expert Systems*, Learned Information, UK, 1986.
44

CUEL87 Cuelenaere, A., M., E., Genuchten, M., J., I., M., Heemstra, F. J., *Calibrating a Software Cost Estimation Model: Why and How*, Information and Software Technology, Vol. 29, No. 10, December 1987, pp 558-567.
62

DAGA85 D'Agapayeff, A., *A Short Survey of Expert Systems in UK Business*, R & D Management, Vol. 15, No. 2, 1985, pp 89-99.
35, 37, 49

DAGA87 D'Agapayeff, A., *Report to the Alvey Directorate on the Second Survey of Expert Systems in UK Business*, Publ. IEE on behalf of The Alvey Directorate, London, August 1987.
37, 41, 44

DATE81 Date, C., J., *An Introduction to Database Systems*, Addison-Wesley, 3rd. Ed., USA, 1981.
133

DAV87 Davoudi, M., KADS: *A Methodology for KBS*, Proceedings of KBS in Government, Gatwick, UK, November 1987, Ed. Paul Duffin, pp 19-36, Online Publications, Pinner, UK.
160

DIG84 Dignan, A., *Software Engineering/IKBS Strategy for Knowledge Based IPSE Development Tools*, The Alvey Directorate, London, November 1984.
18, 157

DIG87 Dignan, A., The Alvey Directorate. Personal Communication.
30

DOWNS86 Downs, Ed, *System Design Development Methodologies*, Computer Systems, February 1986, pp 53-55.
71

DUDA79 Duda, R., Gaschnig, J., Hart, P. E., *Model Design in the PROSPECTOR Consultant System for Mineral Exploration*, in Expert Systems in the Micro-Electronic Age, Ed. Michie, D., Edinburgh University Press, pp 153-167.
38

DUFF87 Duffy, A., *Bibliography - Artificial Intelligence in Design*, Artificial Intelligence in Engineering, Vol. 2, No. 3, 1987, pp 173-179.
66

DUNN85 Dunning, B. B., *Expert System Support for Rapid Prototyping of Conventional Software*, Proceedings of Autotestcon 1985, IEEE International Automatic Testing Conference, New York, USA, October 1985, pp 2-6.
64

DURH87 Durham, T., *Moving Experts Forward One Step at a Time*, Computing, September 17, 1987, pp 20-21.
43, 44

DYER84 Dyer, C., A., *Expert Systems in Software Maintainability*, Proceedings of the Annual Reliability and Maintainability Symposium, San Francisco, CA, USA, January 1984, pp 295-299, IEEE.
63

EDW88 Edwards, J. S. and Bader, J. L., *Expert Systems and University Admissions*, Journal of the Operational Research Society, Vol. 39, No. 1, January 1988 pp 33-40.
30, 49, 154

EDWA87 Edwards-Shea, P., Hannaford, D., Harris-Jones, C., BIS/Estimator: *An Expert System for Estimating Data Processing Projects*, Proceedings of the 3rd International Expert Systems Conference, London, June 1987, Learned Information (Europe) Ltd., UK, pp 395-405.
62

ELI86 Eliot, L. B., Scacchi, W., *Towards a Knowledge-Based System Factory: Issues and Implementations*, IEEE Expert, Winter 1986, pp 51-58.
58

FORD86 Ford, L., *Artificial Intelligence and Software Engineering: A Tutorial Introduction to Their Relationship*, Artificial Intelligence Review, Vol. 1, No. 1, 1986, pp 255-273.
150, 152

FREN85 Frenkel, K. A., *Toward Automating the Software-Development Cycle*, Communications of the ACM, Vol. 28, No. 6, June 1985, pp 578-589.
53, 55

GERV83 Gervarter, W., B., *An Overview of Artificial Intelligence and Robotics, Vol.1 - Artificial Intelligence*, NASA Technical Memo NAS 1.15:85836, June 1983.
152

GLAD82 Gladden, G. R., *Stop the Life-Cycle, I Want To Get Off*, ACM Sigsoft, Software Engineering Notes, Vol. 7, No. 2, April 1982, pp 35-39.
157

GREEN77 Green, C., *A Summary of the PSI Program Synthesis System*, Proceedings of the 5th International Joint Conference on Artificial Intelligence, 1977, pp 380-381.
54

GREEN83 Green, C. et al, *Report on a Knowledge-Based Software Assistant*, Technical Report No. KES.U.83.2, Kestrel Institute, Palo Alto, USA, August 1983.
56, 68

GREENSP86 Greenspan, S., J., *On the Role of Domain Knowledge in Knowledge-Based Approaches to Software Development*, ACM Sigsoft Software Engineering Notes, Vol. 11, No. 4, August 1986, pp 61-65.
146

GRIN86 Grindley, K., *Applying Expert Pronciples to Computer Systems Development,* Data Processing, Vol. 28, No. 1, Jan./Feb. 1986, pp 10-14.
61

HAR85a Harandi, M. T., Young, F. H., *Template Based Specification and Design*, Proceedings of the 3rd International Workshop on Software Specification and Design, London, UK, August 1985, pp 94-97, Publ. IEEE.
59

HAR85b Harandi, M. T., Young, F. H., *A Knowledge Based Design Aid for Software Systems*, Proceedings of Softfair II, San Francisco, USA, December 1985, pp 67-74, Publ. ACM.
59

HAR86 Hartley, K., *Survey of Equipment Used by Alvey Projects*, Alvey News, December 1986, pp 16-18, Publ. The Alvey Directorate, London, UK.
110

HART86 Hart, A., *Knowledge Acquisition for Expert Systems*, Kogan Page, London, 1986.
44, 47

HAY86 Hayward, S., *A Structured Development Methodology for Expert Systems*, Proceedings of KBS '86, Pinner, UK, 1986, Online Publications, UK, pp 195-203.
160

HAYES83 Eds. Hayes-Roth, F., Waterman, D. A., Lenat, D. B., *Building Expert Systems*, Addison-Wesley, USA, 1983.
35, 38, 39, 41, 44, 45, 47, 65, 159

HEBD86 Hebden, C., *Software Engineering for AUTOCOM IV*, in Software Engineering, The Decade of Change, Ed. Ince, D., Peter Peregrinus Ltd., 1986, pp 199-213.
67

HEKM86 Hekmatpour, S., Ince, D., *Rapid Software Prototyping*, Oxford Surveys in Information Technology, Vol. 3, Oxford University Press, pp 37-76.
157

HUR85 Hurst, R., S., Frewin, G., D., Hamer, P., G., *A Rule-Based Approach to a Software Production and Maintenance Management System*, in Esprit '84: Status Report of Ongoing Work, Eds. Roukens, J., Renuart, J., F., Elsevier Science Publishers B.V. (North Holland), 1985, pp 127-144.
63

IES87 Proceedings of the Third International Expert Systems Conference, London, 1987, Publ. Learned Information (Europe) Ltd., Oxford, UK.
44

INCE86 Ince, D., Woodman, M., Hekmatpour, S., *The Application of Some Artificial Intelligence Tools and Techniques in Software Engineering*, in Software Engineering the Decade of Change, Ed. Ince, D., Peter Peregrinus, 1986, pp 81-99.
53

INCE88 Ince, D., *Prototyping Gets a Formal Boost from Alvey Work*, Computing, January 28, 1988, pp 20-21.
64, 67

ISM87 *The ISM Project: Towards a Knowledge-Based IPSE*, The ISM Project Consortium, Software Sciences Ltd., Farnborough, Hampshire, UK, 1987.
58

JAC75 Jackson, M. A., *Principles of Program Design*, Academic Press, New York, USA, 1975.
71

JONES86a Jones, R., *Commercial Expert Systems*, Data Processing, Vol. 28, No. 3, April 1986, pp 115-119.
41

JONES86b Jones, R., *Engineering the Best Possible DP Solution*, Computing, June 19, 1986, p 26.
71

JONES86c Jones, R., *An Automation Revolution Hits System Design*, Computing, July 17, 1986, pp 16-17.
140

JONES87a Jones, R., *Business Gets on the Trail of Expert Advice*, Computing, December 10, 1987, pp 18-19.
41

JONES87b Jones, R., *Time To Automate the Automators*, Computing, February 12, 1987, pp 18-19.
83

KAMP85 Kampen, G., R., *Expert Specification Tools*, Proceedings of the 3rd International Workshop on Software Specification and Design, London, August 1985, pp 120-121, IEEE.
62

KANT79 Kant, E., *Efficiency Considerations in Program Synthesis: A Knowledge-Based Approach*, Doctoral dissertation, Computer Science Department, Stanford University, USA, 1979.
54

KUNZ78 Kunz, J. et al, *A Physiological Rule-Based System for Interpreting Pulmonary Function Test Results*, Heuristic Programming Project, Report No. HPP-78-19, Computer Science Department, Stanford University, USA, 1978.
38

LAND87 Land, F., *Social Aspects of Information Systems*, in Management Information Systems: the Technological Challenge, Ed. Piercy, N., Croon-Helm, London, 1987.
154

LANG88 Langley, N., *Slaves to the System*, Computing, February 11, 1988, pp 18-19.
144

LAWR87 Lawrence, A., *Cleaning Up the Market*, Datalink, September 1987, p 8.
140

LEE86a Lee, M., *Slow Response to Formal Methods*, Computing, April 24, 1986, p 30.
81

LEE86b Lee, M., *Structured Methods: Users State Their Case*, Computing, May 1, 1986, pp 28-29.
81

LEUN85 Leung, C., H., C., Choo, Q., H., A Knowledge-Base for Effective Software Specification and Maintenance, Proceedings of the 3rd International Workshop on Software Specification and Design, London, August 1985, pp 139-142, IEEE.
64

LEWI85 Lewis Johnson, W., Soloway, E., *PROUST: Knowledge-Based Program Understanding*, IEEE Transactions on Software Engineering, Vol. SE-11, No. 3, March 1985, pp 267-275.
63

LUB86 Lubars, M. D., Harandi, M. T., *Intelligent Support for Software Specification and Design*, IEEE Expert, Winter 1986, pp 33-41.
59

MACI87 MacIver, K., *Matters of Fact*, Datalink, September 14, 1987, pp 16-17.
140, 144

MADD83 Maddison, R. N. et al, *Information System Methodologies*, Wiley Heyden Ltd., UK, 1983.
71

MANN78 Manna, Z., Waldinger, R., *DEDALUS-The DEDuctive ALgorithm Ur-Synthesizer*, Proceedings of the National Computer Conference, Anaheim, USA, 1978, pp 683-690.
56

MART88 Martinez, D. R., Sobol, M. G., *Systems Analysis Techniques for the Implementation of Expert Systems*, Information and Software Technology, Vol. 30, No. 2, March 1988, pp 81-88.
180

MCD80 McDermott, J., *R1: An Expert in the Computer Systems Domain*, Proceedings of the 1st Annual National Conference of the American Association for Artificial Intelligence, Stanford, USA, 1980, pp 269-271.
44

MIN75 Minsky, M., *A Framework for Representing Knowledge*, in The Psychology of Computer Vision, Ed. Winston, P. H., McGraw-Hill, 1975, pp 211-277.
101

MITT86 Mittal, S., Dym, C. L., Morjaria, M., *PRIDE: An Expert System for the Design of Paper Handling Systems*, Computer, July 1986, pp 102-114.
65

MUM87 Mumford, E., *User Participation in a Changing Environment - Why We Need It*, Proceedings of Unicom Seminar on Participation in Systems Design, London Business School, April 1987, pp 3-16, Publ. Unicon Seminars Ltd., Middlesex, UK.
82, 154

NCC82 *Information Technology Strategy*, National Computing Centre Publications Manchester, 1983.
15

NEDO82 *Policy for the UK Information Technology Industry*, National Economic Development Office London, November 1982.
15

NOM87 Nomura, T., Lunn, S., *Integration of Knowledge-Based Systems with Data Processing*, Knowledge-Based Systems, Vol. 1, No. 1, December 1987, pp 24-31.
41

NORT86 Norton, M., *Behind the Scenes as the Database Becomes a Star*, Computing, December 4, 1986, pp 22-23.
140

OAK85 Oakley, B. W., *The Alvey Programme: Progress Report-1985*. Alvey Programme Annual Report 1985, pp 7-16, The Alvey Directorate, London, November 1985.
16

ODD86 Oddy, G., C., *The Knowledge Based Programmer's Assistant*, Proceedings of a IEE Colloquium on Knowledge-Based Techniques in Software Engineering, London, October 1986, IEE Digest No. 1986/99, pp 3/1 - 3/3.
63

OLS87 Olson, J. R., Reuter, H. H., *Extracting Expertise from Experts: Methods for Knowledge Acquisition*, Expert Systems, Vol. 4, No. 3, August 1987, pp 152-168.
44

OVUM86 *The Ovum Report - Commercial Expert Systems in Europe*, Ovum Ltd., London, UK, 1986.
44

OWEN85 Owen, K., *Can Advanced Research Help Today's DP Manager?*, Alvey News, April 1985, pp 13-15, Publ. The Alvey Directorate, London, UK.
69

PAO88 Paolo Puncello, P., Torrigiani, P., Pietri, F., Burlon, R., Cardile, B., Conti, M., *ASPIS: A Knowledge-Based CASE Environment*, IEEE Software, March 1988, pp 58-65.
193

PART86a Partridge, D., *Artificial Intelligence - Applications in the Future of Software Engineering*, Ellis Horwood (part of John Wiley & Sons), 1986.
150, 161

PART86b Partridge, D., *Engineering Artificial Intelligence Software*, Artificial Intelligence Review, Vol.1, No. 1, 1986, pp 27-41.
152, 160

PER86 Persch, G., *Automating the Transformational Development of Software*, Proceedings of the 3rd European Seminar and Tutorial on Industrial Software Technology from EWICS, Frieburg, FRG, June 1986.
60

PID85 Pidgeon, C. W., Freeman, P. A., *Development Concerns for a Software Design Quality Expert System*, Proceedings of the 22nd ACM/IEEE Design Automation Conference, Las Vegas, USA, June 1985, pp 562-568.
60

PRER85 Prerau, D., S., *Selection of an Appropriate Domain for an Expert System*, The AI Magazine, Vol. 6, No. 2, Summer 1985, pp 26-30.
41

RAMA84 Ramamoorthy, C. V., Prakash, A., Tsai, W., Usuda, Y., *Software Engineering: Problems and Perspectives*, Computer, October 1984, pp 191-209.
82

READ87 *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, UK, September 1987, Publ. Reading University.
44

RICH78 Rich, C., Shrobe, H. E., *Initial Report on a LISP Programmer's Apprentice*, IEEE Transactions on Software Engineering SE-4(6), pp 456-467, 1978.
54

RICH83 Rich, E., *Artificial Intelligence*, McGraw-Hill, New York, c1983.
101

RYAN88 Ryan, K., *Capturing and Classifying the Software Developer's Expertise*, Proceedings of an International Workshop on Knowledge-Based Systems in Software Engineering, University of Manchester Institute of Science and Technology (UMIST), Manchester, UK, March 1988, issued separately. Publ. by The Information Systems Research Group, Department of Computation, UMIST. Proceedings will also appear in Knowledge Based Systems (Butterworths Publications Ltd., Guildford, UK) in 1988.
146

RYCH85 Rychener, M. D., *Expert Systems for Engineering Design*, Expert Systems, Vol. 2, No. 1, January 1985, pp 30-44.
66

SCHW87 Schweickert, R., Burton, A. M., Taylor, N. K., Corlett, E. N., Shadbolt, N. R., Hedgecock, A. P., *Comparing Knowledge Elicitation Techniques: A Case Study*, Artificial Intelligence Review, Vol. 1, No. 4, 1987, pp 245-253.
44

SHARP88 Sharp, H., *The Role of Domain Knowledge in Software Design*, Technical Report 88/3, Faculty of Mathematics, The Open University, UK, 1988.
146

SHORT76 Shortcliffe, E. H., *Computer-Based Medical Consultations: MYCIN*, American Elsevier, New York, 1976.
38

SHPI86 Shpilberg, D., Graham, L., E., Schatz, H., *Expertax: An Expert System for Corporate Tax Planning*, Proceedings of the Second International Expert Systems Conference, London, September 1986, pp 99-123, Learned Information, Oxford, UK.
131

SMI84 Smith, Reid G., *On the Commercial Development of Expert Systems*, The AI Magazine, Fall 1984, pp 61-73.
41

SMI85 Smith, D. R., Kotik, G. B., Westfold, S. J., *Research on Knowledge-Based Software Environments at Kestrel Institute*, IEEE Transactions on Software Engineering, SE-11(11), November 1985, pp 1278-1295.
55

SOMM83 Sommerville, I., *Information Unlimited*, Addison-Wesley, USA, 1983.
46

SRIR86 Eds. Sriram, D., Adey, R., *Proceedings of the 1st International Conference on Applications of Artificial Intelligence in Engineering Problems*, Southampton University, UK, April 1986, Publ. Springer Verlag.
66

STE74 Stevens, W. P., Myers, G. J., Constantine, L. L., *Structured Design*, IBM System Journal, Vol. 13, No. 2, 1974, pp 115-139.
71

STUR87 Sturridge, H., *DB2 Passes the Quality Control Test at M & S*, Computing, October 1, 1987, pp 30-31.
140

SUM86 Sumner, M., Sitek, J., *Are Structured Methods for Systems Analysis and Design Being Used?*, Journal of Systems Management, June 1986, pp 18-23.
81

SYM88 Symonds, A. J., *Creating a Software-Engineering Knowledge Base*, IEEE Software, March 1988, pp 50 - 56.
193

TALB86 Talbot, D., *The Alvey Programme for Software Engineering*, in Software Engineering, The Decade of Change, Ed. Ince, D., Peter Peregrinus Ltd., 1986, pp 21-26.
67

VAN86 van de Brug, A., Bachant, J., McDermott, J., *The Taming of R1*, IEEE Expert, Fall 1986, pp 33-39.
44

VANA88 Van Assche, F., Layzell, P., Loucopoulos, P., Speltincx, G., *Information Systems Development: A Rule-Based Approach*, Proceedings of an International Workshop on Knowledge-Based Systems in Software Engineering, University of Manchester Institute of Science and Technology (UMIST), Manchester, UK, March 1988, pp D3-1 to D3-24. Publ. by The Information Systems Research Group, Department of Computation, UMIST. Proceedings will also appear in Knowledge Based Systems (Butterworths Publications Ltd., Guildford, UK) in 1988.
61

VANM81 van Melle, W., Shortcliffe, E. H., Buchanan, B. G., *EMYCIN: A Domain-Independent System that Aids in Constructing Knowledge-Based Consultation Programs*, Machine Intelligence, Infotech State of the Art Report, 9, no. 3, 1981.
38

WASS80 Wasserman, A. I., T*oward Integrated Software Development Environments*, Scientia, 115, 1980, pp 663-684.
66, 71, 73

WAT86 Waterman, D. A., *A Guide to Expert Systems*, Addison-Wesley, USA, 1986.
35, 39, 40, 41, 43, 44, 45, 48, 149, 150

WORD87 Worden, R., *Integrating KBS into Information Systems: The Challenge Ahead*, Proceedings of KBS in Government, Gatwick, UK, November 1987, Ed. Paul Duffin, pp 59-69, Online Publications, Pinner, UK.
39, 41, 149

YOU79 Yourdon, E., Constantine, L. L., *Structured Design*, Prentice-Hall, 1979.
71

ZACK87 Zack, B. A., *Selecting an Application for Knowledge-Based System Development*, Proceedings of the Third International Expert Systems Conference, London, 1987, pp 257-269, Publ. Learned Information (Europe) Ltd., Oxford, UK.
41