

# **INFORMATION SYSTEMS DEVELOPMENT**

**Paul Beynon-Davies**  
**THIRD EDITION**



## Macmillan Computer Science Series

- A. Abdellatif, J. Le Bihan, M. Limame, *Oracle - A User's Guide*  
Ian O. Angell, *High-resolution Computer Graphics Using C*  
Ian O. Angell and Gareth Griffith, *High-resolution Computer Graphics Using Pascal*  
C. Bamford and P. Curran, *Data Structures, Files and Databases, second edition*  
P. Beynon-Davies, *Database Systems*  
P. Beynon-Davies, *Information Systems Development, third edition*  
Linda E.M. Brackenbury, *Design of VLSI Systems – A Practical Introduction*  
Alan Bradley, *Peripherals for Computer Systems*  
P.C. Capon and P.J. Jinks, *Compiler Engineering Using Pascal*  
B.S. Chalk, *Computer Organisation and Architecture*  
Eric Davalo and Patrick Naim, *Neural Networks*  
Joyce Duncan, Lesley Rackley and Alexandria Walker, *SSADM in Practice – A Version 4 Text*  
D. England et al., *A Sun User's Guide, second edition*  
Jean Ettinger, *Programming in C++*  
J.S. Florentin, *Microprogrammed Systems Design*  
Michel Gauthier, *Ada - A Professional Course*  
M.G. Hartley, M. Healey and P.G. Depledge, *Mini and Microcomputer Systems*  
M.J. King and J.P. Pardoe, *Program Design Using JSP – A Practical Introduction, second edition*  
Bernard Leguy, *Ada - A Programmer's Introduction*  
M. Léonard, *Database Design Theory*  
David Lightfoot, *Formal Specification Using Z*  
A.M. Lister and R.D. Eager, *Fundamentals of Operating Systems, sixth edition*  
Tom Manns and Michael Coleman, *Software Quality Assurance, second edition*  
G.P. McKeown and V.J. Rayward-Smith, *Mathematical Foundations for Computing*  
B.A.E. Meekings, T.P. Kudrycki and M.D. Soren, *A book on C, third edition*  
R.J. Mitchell, *C++ Object-oriented Programming*  
R.J. Mitchell, *Microcomputer Systems Using the STE Bus*  
R.J. Mitchell, *Modula-2 Applied*  
J.P. Pardoe and M.J. King, *Object Oriented Programming Using C++*  
Pham Thu Quang and C. Chartier-Kastler, *MERISE in Practice*  
Ian Pratt, *Artificial Intelligence*  
F.D. Rolland, *Programming with VDM*  
Z.M. Sikora, *Oracle Database Principles*  
S. Skidmore, *Introducing Systems Analysis, second edition*  
S. Skidmore, *Introducing Systems Design, second edition*  
A.G. Sutcliffe, *Human-Computer Interface Design, second edition*  
C.J. Theaker and G.R. Brookes, *Concepts of Operating Systems*  
M. Thorin, *Real-time Transaction Processing*  
D.J. Tudor and I.J. Tudor, *Systems Analysis and Design – A Comparison of Structured Methods*  
A.J. Tyrrell, *Eiffel Object-Oriented Programming*

*Other titles*

Ian O. Angell and Dimitrios Tsoubelis, *Advanced Graphics on VGA and XGA Cards Using Borland C++*

N. Frude, *A Guide to SPSS/PC+, second edition*

Peter Grossman, *Discrete Mathematics for Computing*

H. Harper and A. Meadows, *GNVQ Advanced Information Technology*

P.D. Picton, *Introduction to Neural Networks*

Tony Royce, *COBOL - An Introduction*

Tony Royce, *Structured COBOL - An Introduction*

Tony Royce, *C Programming*

# **Information Systems Development**

## **An Introduction to Information Systems Engineering**

**Paul Beynon-Davies**  
*Department of Computer Studies*  
*University of Glamorgan*

Third Edition



© P. Beynon-Davies 1989, 1993, 1998

All rights reserved. No reproduction, copy or transmission of this publication may be made without written permission.

No paragraph of this publication may be reproduced, copied or transmitted save with written permission or in accordance with the provisions of the Copyright, Designs and Patents Act 1988, or under the terms of any licence permitting limited copying issued by the Copyright Licensing Agency, 90 Tottenham Court Road, London W1P 9HE.

Any person who does any unauthorised act in relation to this publication may be liable to criminal prosecution and civil claims for damages.

The author has asserted his rights to be identified as the author of this work in accordance with the Copyright, Designs and Patents Act 1988.

First edition 1989

Second edition 1993

Third edition 1998

Published by

MACMILLAN PRESS LTD

Houndsborough, Basingstoke, Hampshire RG21 6XS

and London

Companies and representatives

throughout the world

ISBN 978-0-333-74481-9 ISBN 978-1-349-14931-5 (eBook)

DOI 10.1007/978-1-349-14931-5

A catalogue record for this book is available from the British Library.

This book is printed on paper suitable for recycling and made from fully managed and sustained forest sources.

10 9 8 7 6 5 4 3 2 1  
07 06 05 04 03 02 01 00 99 98

*For my Wife Gillian and my Children:  
Rhydian, Ceri and Rhiannon*

The idea is like grass.  
It craves light,  
likes crowds,  
thrives on crossbreeding,  
grows better for being stepped on.

Ursula Le Guin  
*The Dispossessed*

# ***Contents***

<b>Preface</b>	ix
<b>Part One: Context</b>	1
1    Information	3
2    Information Systems	13
3    Information Systems and Organisations	21
4    Information Society and Economy	29
5    Information Technology	36
6    Information Systems Services Function	46
7    Information Systems Engineering	53
<b>Part Two: Tools</b>	67
8    Structured Programming Languages	69
9    Object-Oriented Programming Languages	76
10   Database Systems	83
11   Fourth Generation Environments	95
12   Computer Aided Information Systems Engineering	103
13   Multimedia and Hypermedia Information Systems	109
14   Knowledge Based Systems	116
<b>Part Three: Techniques</b>	124
Section One:       Data Analysis	125
15   Normalisation	127
16   Entity-Relationship Diagramming	141
Section Two:       Process Analysis	154
17   Data Flow Diagramming	155
18   Process Descriptions	169
19   Data Dictionaries	179
20   Entity Life Histories	188
Section Three:      Other Techniques	196
21   Structured Program Design	197
22   Object-Oriented Analysis and Design	212
23   User Interface Design	224
24   Prototyping	232
<b>Part Four: Methods</b>	242
Section One:       Business Analysis Methods	243
25   Business Process Re-engineering	244

26	Soft Systems Methodology	252
<b>Section Two: Development Methods</b>		258
27	Structured Methods	259
28	Participatory Design	273
29	Rapid Applications Development	282
30	Object-Oriented Methods	291
<b>Part Five: Management</b>		297
31	Project Management	298
32	Information Systems Management	306
33	Information Systems Planning	316
34	Information Systems Evaluation	329
35	Information Systems Outsourcing	337
36	Quality Assurance	343
<b>Part Six: Discipline</b>		351
37	Information Systems Failure	352
38	Information Systems and Their Interaction with Organisations	362
39	The Professionalisation of Information Systems Work	374
40	The Development of the Discipline of Information Systems	383
<b>Index</b>		392

# ***Preface to the Third Edition***

## **Changes**

The third edition of this work displays much of its ancestry in the first and second editions. From the first edition, it inherits its non-deterministic character in avoiding the imposition of any one particular framework on the use of tools, techniques and approaches described. Instead, it provides a series of relatively discrete notebooks on important topics in the area of information systems development. From the second edition it takes its objective of presenting a balanced account of both technical as well as organisational issues in the information systems area. This means continually emphasising the importance of aligning information systems with the needs of organisations.

The third edition represents a less radical change than that which occurred between the first and second editions. The basic approach of parcelling up the field into a number of separate parts, primarily for the purposes of presentation has been maintained. Having said this, we have restructured two of the original parts and renamed them Management and Discipline, because we believe that these labels correspond more readily to accepted groupings of topics discussed in this field.

At the more detailed level a number of other changes include:

1. Updating the material on tools, techniques and methods to take account of modern developments.
2. The removal of some chapters either because of the feeling we get that they are becoming less, rather than more relevant to information systems engineering (e.g. formal methods), or because we now feel that the topic areas are better discussed in other chapters (e.g. social science concepts).
3. A number of other chapters have been re-worked in order to more effectively set IS development within its organisational context. We also include a discussion of IS management and planning issues, particularly as they affect the IS development process.

## **Strategy**

The topic of Information Systems Development is a large one. It is a topic which is getting larger by the day as information systems impact more and more on our everyday and organisational lives.

Therefore, in this work we have parcelled up the field into several parts, primarily for the purposes of presentation.

1. *Context.* Here we discuss issues such as what is information, what is an information system, why we need information systems, how the concept of information is playing an ever more prevalent role in our lives, and how appropriate ways for developing information systems assume ever more importance.
2. *Tools.* Here we discuss some of the components with which, and from which, contemporary information technology systems are built.
3. *Techniques.* Here we describe some of the major techniques of analysis and design comprising the armoury of the information systems specialist.
4. *Methods.* Here we discuss some of the contemporary ideas which relate to a number of distinct approaches to building information systems.
5. *Management.* Here we discuss some of the current issues relating to the planning, management and support of IS projects. We also critically examine the role that quality and evaluation play within IS management.
6. *Discipline.* Here we conclude with an examination of four topics which help to define the discipline of information systems area. First, we raise the problem of the ubiquity of IS failure. Second, we re-iterate the importance of considering the fit between information systems and the organisations within which they work. Third, we describe the process of professionalisation which many within the information systems area are promoting. Fourth, we examine the issue of what constitutes the valid academic study of information systems development.

This work is subtitled *An Introduction to Information Systems Engineering*. My main aim in rewriting this work is to contribute to the development of the discipline of information systems in general and information systems engineering in particular. We use the term IS engineering here in the broad sense used by the British Computer Society (BCS). In one sense, IS engineering is a discipline which broadens the area of software engineering from ‘programming in the large’ to include issues of people and organisations. In another sense, IS engineering has a more specific focus than software engineering in concentrating on software written to support human activity (particularly decision-making) within organisations.

Casting the discipline as engineering, however, does not mean that I agree with the rather limited conception of engineering portrayed in many quarters of computing. Information systems are, by their very nature, open to interpretation from a number of different viewpoints. In this respect I am conducting a similar exercise to Ehn (1989) in trying to focus on a disciplinary base for the interdisciplinary subject matter of designing computer artefacts. It is hoped that some of this diversity is reflected in the current text.

As a final comment, it is hoped that further editions of this work will be produced to keep the text up-to-date with ongoing developments. Any suggestions concerning the current edition or material to be included in subsequent editions would be welcomed, and should be sent, addressed to the

author, care of the publisher. Also, a teaching pack to accompany this textbook, including sample solutions to the exercises, is available from the author.

### Acknowledgements

My thanks to staff and students of the Department of Computer Studies, the University of Glamorgan, for providing many useful comments on various iterations of the material presented in this book. Thanks especially to Malcolm Stewart at Macmillan for managing the production of all three editions of this work. May I wish him a long, happy and well-deserved retirement.

### Reference

- Ehn P. (1989). The Art and Science of Designing Computer Artefacts.  
*Scandinavian Journal of Information Systems*. 1(1). 21-42.

Paul Beynon-Davies

# ***Part One***

## ***Context***

### **Context**

General setting of experience. (*Oxford English Dictionary*)

Circumstances, factors, situation, milieu. (*Roget's Thesaurus*)

In this part, we discuss a number of issues that define the important context for information systems development work. Chapters 1 and 2 provide important definitions for the terms information and information systems. Chapter 3 discusses a number of different ways of defining the concept of organisation and also highlights the way in which different perspectives on what constitutes an organisation influences ways of conceiving of the utilisation of information technology. Chapter 4 addresses a number of issues surrounding the information society and chapter 5 introduces some of the primary component parts of information technology. Chapter 6 discusses that organisational function devoted to the planning, management and development of information systems. Chapter 7 concludes with a discussion of a number of different styles of information systems development.

# **1 Information**

## **1.1 Introduction**

The concept of information and its exploitation in information systems has been somewhat taken for granted in the contemporary practice of development. Information has been treated in many respects as a mystical fluid that emanates as if by magic from the development of computerised information systems.

As an attempt to combat this simplistic conception there has been a resurgence of interest in the philosophical and sociological underpinnings of information systems work. As such, a discipline of information systems is emerging which has boundaries with management, business studies, behavioural science, computing and many other areas. This chapter provides an introduction to this material. Many of the concepts discussed in this chapter will permeate the discussion of the chapters that follow.

## **1.2 What is Information?**

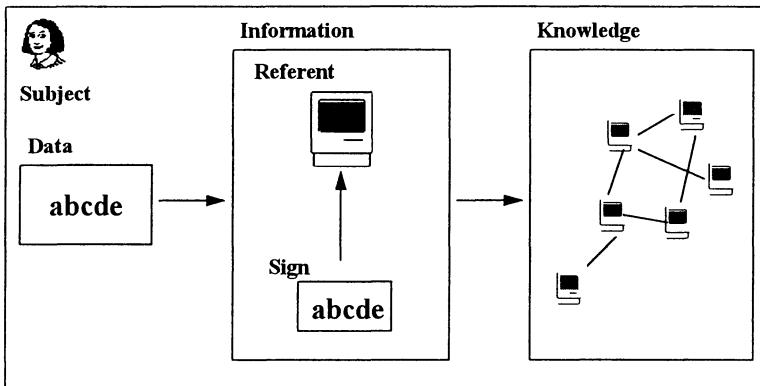
The concept of information is an extremely vague one open to many different interpretations (Stamper, 1985). One conception popular in the computing literature is that information results from the processing of data: the assembly, analysis or summarisation of data. This conception of information as analogous to chemical distillation is useful, but ignores the important 'place of human interpretation'.

In this section we shall define a workable definition of information based upon the distinction between data, information and knowledge. We shall then elaborate on this definition in further sections, in particular, adding people into the equation.

Tsitchizris and Lochovsky define information as being 'an increment of knowledge which can be inferred from data' (1982). Information therefore increases a person's knowledge of something. Note that this definition interrelates the concepts of data, information, knowledge and people (see figure 1.1):

1. Data is facts. A datum, a unit of data, is one or more symbols that are used to represent something.
2. Information is interpreted data. Information is data placed within a meaningful context.
3. Knowledge is derived from information by integrating information with existing knowledge.

4. Information is necessarily subjective. Information must always be set in the context of its recipient. The same data may be interpreted differently by different people, depending on their existing knowledge.



43	Personnel number No of products of a particular type sold	Attendance profile of department Total no of products sold
----	--	---

**Figure 1.1: Data, Information and Knowledge**

### *Example*

Consider the string of symbols 43. Taken together these symbols form a datum, but by themselves they are meaningless. To turn these symbols into information we have to supply a meaningful context. We have to interpret them. This might be that they constitute an employee number, a person's age, or the quantity of a product sold. Information of this sort will contribute to our knowledge of a particular domain. It might, for instance, add to our understanding of the total number of products of a particular type sold.

### *Example*

Another potent example of sign production is the cryptanalytic work conducted at Bletchley Park in the UK during the Second World War (Hodges, 1983). Here the data constituted encrypted radio signals from the German war machine. Decryption involved translating the signals into referents such as German U-boat positions and movements. The information gleaned was then incorporated into the total knowledge available to allied intelligence on military movements and strategy.

### 1.3 Semiotics

Liebenau and Backhouse (1990), following Stamper (1973, 1985), have discussed information in terms of semiotics. Semiotics or semiology is the study of signs.

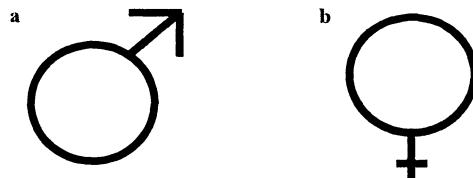
A sign is anything that is significant. In a sense, everything which humans do is significant to some degree. The world within which humans find themselves is resonant with sign-systems. A sign-system is any organised collection of signs. Everyday spoken language is probably the most readily accepted and complex example of a sign-system. Signs however exist in most other forms of human activity, since signs are critical to the process of human communication and understanding.

Note the link between the words sign and significant in English. These words clearly have the same root. The concept of the significance of signs cannot be divorced from people. Different people find different things significant.

A sign can be broken down into three component parts:

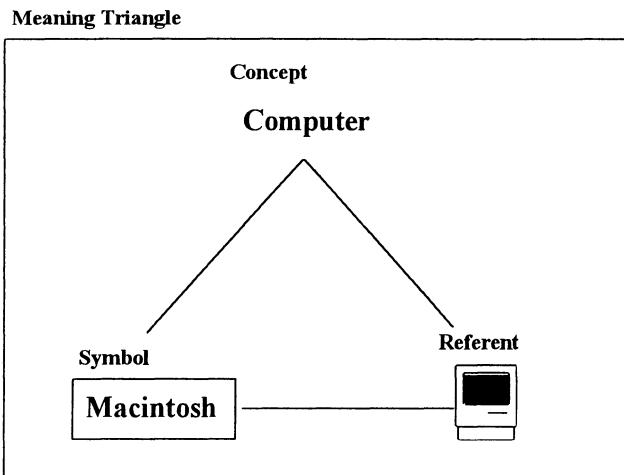
1. The *symbol*, sometimes referred to as the signifier. That which is signifying.
2. The *referent*, sometimes known as the signified. That which is being signified.
3. The *concept*. The idea of significance.

Take the example of the symbols indicated in figure 1.2. The symbol in figure 1.2a has as its referent the male population; the symbol in figure 1.2b has as its referent the female population. 1.2a has as its concept the idea of maleness; 1.2b the concept of femaleness.



**Figure 1.2: Two Symbols**

The three components of a sign are related together in a meaning triangle (figure 1.3) (Sowa, 1984). The left corner of the triangle is the symbol; the peak is the concept; and the right corner is the referent. In this figure the symbol is the word ‘Macintosh’; the referent is a particular range of electronic devices which are normally generalised in terms of the concept of a computer.



**Figure 1.3: Components of a Sign**

## 1.4 Social Systems

Signs are only significant in terms of some social system. A social system provides the context for the interpretation of signs. For instance, Dahlbohm and Mathiassen (1993) is a useful text which discusses how different perspectives on computing work (a social system) are crucial to understanding practice in this area. This accords with the call by Backhouse *et al.* (1991) for a focus for information systems research that draws on much of the work of Stamper. They have proposed an organising framework based on the disciplines of sociology and semiology:

‘Two well-established areas of study provide us with a firm foundation on which to build. These are sociology on the one hand and semiology on the other. The former already constitutes the main thrust of research into social organisation, institutional dynamics, group interaction, working conditions and social policy. The latter brings together the range of studies associated with contexts of language and communication, meanings, grammars, signs and codes’. (Backhouse, *et al.* 1991)

A number of concepts borrowed from social science are useful in understanding signs and their relationship to social systems: culture, norm, role, power and status. These concepts are important in understanding human behaviour in organisations and hence will recur in the discussion of the chapters that follow.

#### ***1.4.1 Culture***

A *culture* is the set of behaviours expected in a social group. A *social group* is any collection of people who regularly interact with one another. It is a truism that any long-standing social group develops its own expected set of behaviours. Such expectations are known as *norms* of human behaviour in the jargon of social science, to distinguish them from deterministic rules.

#### ***1.4.2 Roles***

People make behaviour. But people package behaviour in collections which social scientists call *social roles*. The analogy here is clearly with the acting profession. To paraphrase Shakespeare – ‘All the world is a stage, and we are merely players.’ In other words, we all take on a number of different roles throughout our lifetime, and for each role different expectations are involved.

#### ***1.4.3 Status***

A complementary concept to that of social role is that of social status. A person’s status implies some form of position in a social hierarchy. The formal hierarchy of an organisation may not be the most important one to know about. There are frequently one or more informal hierarchies that are more effective in organisational terms. Such hierarchies, for instance, frequently stimulate camouflage. Many people put up a ‘front’ in organisations to preserve their informal status. They may tell you that the organisation functions in such a way, when in fact it functions entirely differently.

#### ***1.4.4 Power and Authority***

Power is the ability of a person or social group to control the behaviour of some other person or social group. Authority is legitimised power in that those over whom it is exercised accept the exercise of power. Organisational politics is the embodiment of power play within some organisation.

In many organisations there will be a mismatch between naked power and authority. Manager A may be the person in whom formal power is invested. Manager B however has the recognised authority, and it is this manager who actually controls staff.

There is a tendency to conceptualise power as a commodity owned by a particular individual or group. Power and authority are however not static entities. Power and authority are relationships between individuals and groups. They are fluid entities that are continuously negotiated within organisations.

### **1.5 Signs and Social Systems**

All of the concepts discussed in relation to social systems above are clearly related to the concept of a sign:

1. If culture consists of shared understanding, such understanding must be communicated via a body of signs.
2. A person's role and indeed social status is frequently made apparent through signs. Corporate executives, for instance, will frequently signify their status in the organisation via a panoply of status 'symbols' such as expensive company cars, large offices and expensive furnishings.
3. Power has to be exercised through communication. People gain an idea of a person's authority through the significance of their ability to control the behaviour of others.

### **1.6 Data, Information and Signs**

The distinction between data and information can be viewed from the perspective of signs and sign-systems. In terms of the meaning triangle, data is symbols. For instance, the symbols M and F might be significant in some information systems context. To speak of information we must supply some concept and/or referent for the symbols. M might have as its referent the male population; F might have as its referent the female population. Taken together, the meaning of these symbols is supplied by the concept of human gender.

### **1.7 Levels of Signs**

Another way of considering signs and sign-systems is in terms of a number of levels of study (Stamper, 1973):

1. *Pragmatics*. The study of the general context and culture of communication. The shared assumptions that underlie human understanding. In order for communication to occur between human beings signs must exist in a context of shared understanding. There must be agreed expectations between the symbols and the referents or concepts that they signify. Pragmatics is the study of such mutual understanding. Much of pragmatics can be considered as the study of culture – the common expectations underlying human behaviour in a particular context.
2. *Semantics*. The study of the meaning of signs. The association between signs and behaviour. Semantics can be considered as the study of the link between symbols and their referents or concepts; particularly the way in which signs relate to human behaviour embodied in norms.

3. *Syntactics*. The logic and grammar of sign systems. Syntactics is devoted to the study of the physical form rather than the content of signs.
4. *Empirics*. The physical characteristics of the medium of communication. Empirics is devoted to the study of the medium of communication, e.g. sound, light, electronic transmission, etc.

Pragmatics and semantics study the purpose and content of communication. Syntactics and empirics study the forms and means of communication. Pragmatics and semantics clearly impinge upon other disciplines such as sociology and politics. Syntactics and empirics impinge upon the domain of psychology and indeed even electronics.

In information systems work the study of syntactics and empirics has traditionally dominated. The pragmatics and semantics levels have only recently begun to contribute to information systems development.

### ***Example***

Consider, for instance, a mail message transmitted on an electronic mail system. The mail message is something that exists within a social setting. It therefore exists within an environment of expectations, commitments and obligations. At the pragmatic level there must be some reason for sending the message which is presumably expressed in terms of the culture and context in which the information is used. At the semantic level the focus shifts to the subject matter of the message; what meaning is being transmitted by the message. At the syntactic level the language used to express the message is of concern. In so far as the communication takes place along electronic communication lines issues such as bandwidths and other properties of signal transmission will be the concern of the empirics level.

### ***Example***

Consider the icon in figure 1.4. At the level of empirics the medium of communication is clearly light and visual perception. Syntactics addresses the physical form of the sign, including in this case issues of colour and layout. Semantics concerns the meaning of the sign; in this case a prohibition not to smoke cigarettes. It involves the expectation that persons seeing this sign will modify their behaviour in response to its intended meaning. Pragmatics addresses the issues surrounding the general use of this sign in a human setting. In this case, it involves the general understandings concerning the link between cigarette smoking and ill health, as well as the debate about the effects of passive smoking.



**Figure 1.4: A Sign**

### **1.8 ‘Good’ Information**

There is a tendency to assume that all information is good information. This is particularly prevalent in the perspective that information supports decision-making; that information reduces the uncertainty in some domain, and that as a natural consequence the more information you have, the better the decisions you will make (chapter 3). This perspective suffers from a number of problems:

1. It ignores the fact that people can become overloaded with information. At some point, the more information a person has, the less effective he/she will be in making a decision.
2. Not all information reduces uncertainty, and more information certainly does not always reduce uncertainty. Some forms of information may actually increase uncertainty, in giving the decision-maker more options.

The important point is that we cannot divorce the concept of information from its use. ‘Good’ information can only be determined in the context in which it serves human action. Hence, the design and development of information systems must encompass the context of use.

#### ***Example***

The Audit Commission (1995) report on information needs in hospitals identifies a number of ways in which information is essential to supporting clinical work. Implicitly these form definitions of what constitutes good information in this domain.

##### ***Supporting clinical decisions***

1. assessing a patient’s medical history for relevant problems

2. aiding the management of a patients' visit or stay
3. assisting with tasks such as producing order forms, discharge letters etc.
4. sharing information about the care and progress of the patient with other health care professionals

*Monitoring clinical performance*

1. providing information for the audit of individual clinical cases and for quality assurance relating to services and outcomes
2. providing information for research studies into best care practices

*Evaluating clinical performance*

1. monitoring the quality of care against standards specified in contracts
2. monitoring costs and taking measures to reduce them where appropriate
3. ensuring that bills are raised for services provided
4. making statutory returns to government and international bodies

## 1.9 Conclusion

In this chapter we have sought to examine some of the accepted wisdom surrounding the term information. Data is facts. Information is interpreted data. Information is data placed within a meaningful context. Information is signs. Signs are inherently associated with human interpretation, understanding, and communication. Social systems and signs are inherently interlinked. In semiotic terms, information can be considered at a number of different levels: pragmatics, semantics, syntaxes and empirics. Not all information is necessarily good information.

## 1.10 References

- Audit Commission. (1995). *For Your Information*. HMSO.
- Backhouse J., Liebenau J. and Land F. (1991). 'On the Discipline of Information Systems'. *Journal of Information Systems*. 1(1). January. 19-27.
- Dahlbohm B. and Mathiassen L. (1993). *Computers in Context: the philosophy and practice of systems design*. NCC/Blackwell.
- Hodges A. (1983). *Alan Turing: the Enigma*. Hutchinson, London.
- Liebenau J. and Backhouse J. (1990). *Understanding Information: An Introduction*. Macmillan Press, Hounds-mills, Basingstoke.
- Sowa J.F. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley. Reading, Mass.
- Stamper R.K. (1973). *Information in Business and Administrative Systems*. Batsford, London.
- Stamper R.K. (1985). Information: Mystical Fluid or a Subject for Scientific

Enquiry? *The Computer Journal*. 28(3).  
Tsitchizris D.C. and Lochovsky F.H. (1982). *Data Models*. Prentice-Hall,  
Englewood Cliffs, NJ.

### 1.11 Exercises

1. An insurance society wishes to produce a list of all the claims made against policies. Analyse this report in terms of the distinction between data, information and knowledge.
2. Analyse the claims report in terms of Stamper's distinction between semantics, syntactics, pragmatics and empirics.
3. If all the products of information systems can be regarded as signs, discuss the problems involved in individuals and groups misinterpreting certain of these signs.
4. Discuss the effectiveness of the Highway Code as a sign system.

## ***2 Information Systems***

### **2.1 Introduction**

The term information system is used in a number of different contexts in the literature. This chapter attempts to build a workable definition of the term and in the process introduces a number of important distinctions which impinge on the material in further chapters.

### **2.2 Systems**

A system might be defined as a coherent set of interdependent components which exists for some purpose, has some stability, and can be usefully viewed as a whole. Systems are generally portrayed in terms of an input-process-output model existing within a given environment. The environment of a system might be defined as anything outside a system that has an effect on the way the system operates. The inputs to the system are the resources that it gains from its environment or other systems. The outputs from the system are that which it supplies back to its environment or other systems. The process of the system is the activity that transforms the system inputs into system outputs. Most organisations are open systems in that they are affected by their environment and other systems.

The idea of a system has been applied in many fields as diverse as physics, biology and electronics. The class of systems to which information systems are generally applied have been referred to as human activity systems (Checkland, 1980). Such systems have an additional component added to the input-process-output model described above: people. Human activity systems consist of people, conventions and artefacts designed to serve human needs.

Every human activity system will have one or more information systems. The purpose of these information systems is to help manage the human activity system.

### **2.3 A Simple Human Activity System**

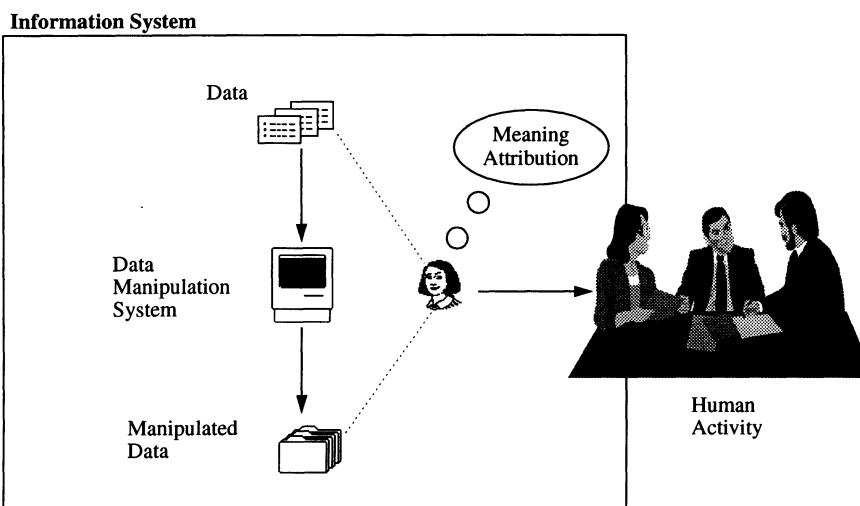
Take the example of a simple business that handles claims made against insurance companies as the result of motor accidents. The environment of this system comprises its relationship to its parent company, other insurance companies and the financial industry at large. The main inputs to the system are the claims made against insurers. The main outputs from the system are payments made to other insurers. The main activity of the system is the

resolution of claims. The primary people involved in the system are the claims clerks that take a particular claim from inception through to resolution. The system is also made up of a vast range of conventions governing the appropriate ways of processing claims. Most of these conventions govern the flow of information through the system: the information system.

## 2.4 Information Systems and Human Activity Systems

The primary purpose for creating an organised information system is to serve real-world action. The provision of information in organisations is always linked to action: to deciding to do things, accomplishing them, observing and recording the results and if necessary iteratively repeating this process. From the definition of information as data to which meaning has been attributed (chapter 1) and the objective of information systems as servers for action a number of consequences follow:

1. The boundary of information systems will always have to include the attribution of meaning, which is a uniquely human act. An information system will consist of both data manipulation and the attribution of meaning by humans (see figure 2.1).
2. The process of developing an information system requires explicit attention to the purposeful action that the information system is meant to serve. This involves understanding how the people in the organisation conceptualise their world: how they attribute meaning, and how this meaning drives their action.



**Figure 2.1: Information Systems and Human Activity Systems**

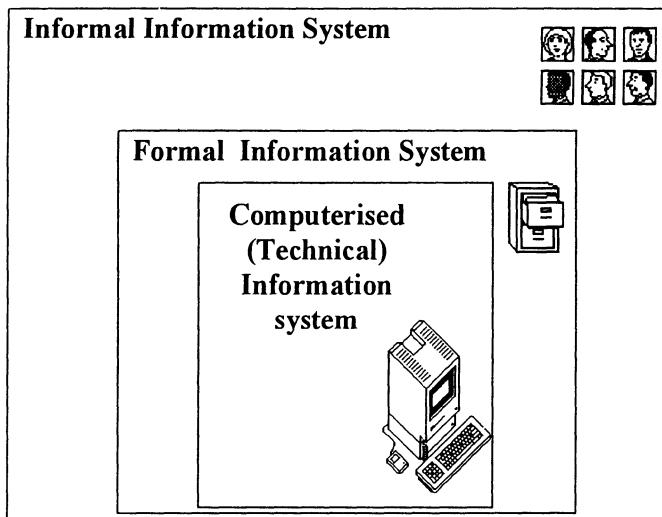
These tenets have important consequences in terms of how we approach the development of computerised information systems. Information systems cannot be constructed until some agreement has been reached about appropriate forms of understanding and action within organisations. We cannot separate information and information systems from human activity. In other words, any developer of computerised information systems has to address the important human dimension of organisations.

## 2.5 Formal, Informal and Technical Information Systems

Stamper makes the useful distinction between three levels of information system within organisations. Information systems may be formal, informal, or technical (see figure 2.2). Stamper (1973) cogently defines the differences between these three levels as follows:

‘To understand an organisation we must recognise three layers corresponding to the formal, informal and technical levels at which culture is transmitted and behaviour determined. At the formal level are the explicitly recognised precepts of behaviour which may be a part of the wider culture in which the organisation operates; on the other hand they may be expressed in the rules, regulations and official structure of authority. At the informal level, an organisation will gradually evolve complex patterns of behaviour which are never formulated, but which must be learnt by newcomers. The informal culture will be vital to the effectiveness of the organisation; in some respects it may aid, and in others impede, the attainment of organisational objectives.... At the technical level an organisation must be described in terms of its flows of messages about the transactions performed, plans made, problems investigated, and in terms of the data-processing activities necessary to accomplish organisational tasks.’

The important point about this definition is that computerisation normally addresses only a limited part of the formal information systems of organisation. Note, however, that this distinction between technical, formal and informal information systems is useful for conceptual purposes. It is a distinction that is frequently difficult to maintain in practice.



**Figure 2.2: Levels of Information System**

## 2.6 Formal Information Systems

Formal information systems are particularly prevalent in bureaucratic forms of organisation (Weber, 1947). A bureaucracy is a form of organisation in which human activity is highly regulated. Conventions are written down; control is exercised by formal hierarchies; communication is vertical in the sense of information travelling upwards from subordinates to superiors and decisions travelling in the opposite direction.

Many public-sector organisations are usually considered archetypal bureaucracies, although bureaucracies are also highly prevalent in the private sector as well.

## 2.7 Technical Information Systems

By a technical information system we normally mean the representation of some part of a formal information system using information technology. The term Information Technology (IT) is generally used to describe hardware and software for supporting information work, e.g. computers, communication networks, operating systems, database management systems etc. We shall use the term IT system as a synonym for the term technical information system.

One of the main points arising from figure 2.2 is that since technical information systems are conventionally built for supporting some of the formal information systems activity of a given organisation, a technical information system usually cannot be constructed until there is some agreement concerning

human activity and this activity has been formalised in the sense that it is specified and/or documented. The frequent difficulty in IS development is obtaining and maintaining such agreement.

## 2.8 Informal Information Systems

An informal information system can be considered as a corpus of expectations about how people should communicate in some setting. These expectations are by their very nature never written down or formalised. People learn the expected way of doing things in the process of being socialised into the organisation. However, such expectations are merely guides for action. People are able to adapt their behaviour in the light of changed circumstances.

The main importance of informal information systems is that they can be more robust than information systems established on formal and particularly technical grounds. An informal information system, because of its very nature, is likely to be better able to adapt to the changes in the external environment of the organisation.

### *Example*

Consider the case where in some organisation clerks are conventionally expected to issue purchase orders to established suppliers. However, this expectation is an informal one. No regulation has been established within the organisation to this effect. An established supplier for a particularly critical resource now goes out of business. If the expectation described above had been formalised then the clerks would probably have to go out to tender for a new supplier for the resource – a process which could take some considerable time. Because the expectation is an informal one clerks can immediately go out and look for a new supplier and adjust their decision-making on this basis.

## 2.9 Systems Analysis

The term systems analysis is normally used to describe the process of analysing aspects of formal information systems. Systems analysis is the precursor to systems design, which is the activity devoted to developing the plan for some information technology system.

A typical scenario is one in which the systems analyst is expected to go into an organisation and document all or part of the bureaucratic practices existing in that organisation. This typical conception of systems analysis we might call formal or ‘hard’ systems analysis.

## **2.10 Soft Systems Analysis**

In the same way that we distinguish between a formal and an informal information system we may likewise distinguish between formal and informal systems analysis. Formal or 'hard' systems analysis is the process of investigating and documenting the formal information systems of organisations. Informal or 'soft' systems analysis is the process of investigating and documenting the informal information systems of organisations.

Soft systems analysis equates quite closely with the idea of business analysis: of analysing the overall objectives and needs of an organisation and identifying the place of the organisation within its environment (Checkland and Scholes, 1990). The topic of business analysis is discussed in chapters 25 and 26. The importance of business analysis for IS work lies particularly in the identification of the appropriate place for information systems within organisations.

## **2.11 The Importance of Considering Informal Information Systems**

On the morning of 27th October 1986 the new computerised price and dealing information system of the London stock market crashed. Breakdowns and suspensions of trading occurred on the system for the following month. When an evaluation was conducted the source of the problem proved to be a massive underestimation of the volume and timing of system usage. On the first day, everybody logged onto the system out of curiosity. Thereafter, regular massive early loading of the system was commonplace as dealers struggled to get their share prices right as early in the day as possible (Willcocks and Mason, 1987).

The analysis of the formal system of stock market information was done well. The price and dealing information system (the IT system) was produced on time, to specification and within budget. However, the analysis of the informal system of share dealing appears to have been done badly. Analysts failed to get an appreciation of the importance of timely share information for brokers.

## **2.12 The Information Systems Specialist**

Institutions such as the British Computer Society have recently become interested in broadening the skills expected of information systems engineers. The idea of a hybrid manager – a person with both business and technical knowledge – has been much discussed (Palmer, 1990). In a sense, this initiative is merely echoing a sentiment first expressed by Stamper in the 1970s:

'The demands of society and the opportunities of technology are now changing so quickly that we must learn to construct organisations that are responsive to our needs. Organisations cannot be left to evolve; as far as it

is possible they must be designed. Many people are working on these problems: managers, administrators and staff specialists. In one way or another they are all trying to make organisations use information effectively. It is information that holds organisations together and drives them along. What we urgently need therefore, are information specialists who are as thoroughly acquainted with the information needs of organisations as they are with the capabilities of modern information technology.' (Stamper, 1973)

### 2.13 Goronwy Galvanising: A Case Study

To provide some context for many aspects of the discussion of information systems development, which follows, we begin portraying here the elements of a central case study. The aim is to illustrate how an information system that is to be developed for a small manufacturing company can be viewed from a number of different perspectives. In particular, the case study forms a central part of the techniques section where we illustrate the application of each technique to an aspect of the system. In this chapter we mainly consider the broad implications of the informal and formal information systems at the company. In chapter 2 we inspect the detail of the formal system specific to the proposed technical information system.

Goronwy Galvanising Ltd are a small company specialising in treating steel products such as lintels, crash barriers and palisades etc., produced by other manufacturers. Goronwy are a subsidiary of a large multi-national whose primary business is the production of various alloys. The multi-national maintains 10 plants on similar lines to Goronwy around the UK. Each plant is relatively autonomous in terms of managing its day-to-day business. Head office co-ordinates administrative activities such as accounting, finance and the dissemination of information.

Galvanising, in very simplistic terms, involves dipping steel products into baths of molten zinc to provide a rustproof coating. Untreated steel products are described as being 'black'. Treated steel products are referred to as 'white'. There is a slight gain in weight as a result of the galvanising process.

Goronwy mainly process steel for a major manufacturer known as Blackheads. More than 80% of their business is with Blackheads which places a regular set of orders with Goronwy. Other manufacturers, such as Pimples, order galvanised steel on an irregular basis.

The staff at Goronwy consists of a plant manager, a production controller, an office clerk, 3 shift foremen and 50 shop-floor workers. The plant remains open 24 hours per day, seven days a week. Most of the production workers including the foremen work a shift-pattern.

## 2.14 Conclusion

To summarise, a system might be defined as a coherent set of interdependent components which exists for some purpose, has some stability, and can be usefully viewed as a whole. Systems have inputs, outputs and processes. They work within a given environment. Human activity systems or 'soft' systems are systems with people. Every human activity system has one or more information systems. Information systems can be distinguished in terms of their level of formality: technical, formal, and informal. Conventional systems analysis is normally devoted analysing formal information systems. Soft systems analysis is devoted to analysing informal information systems.

## 2.15 References

- Checkland P. (1980). *Systems Thinking, Systems Practice*. John Wiley, Chichester.
- Checkland P. and Scholes J. (1990). *Soft Systems Methodology in Action*. John Wiley, Chichester.
- Wilcocks L. and Mason D. (1987). *Computerising Work: People, Systems Design and Workplace Relations*. Paradigm, London.
- Palmer C. (1990). Hybrids – a growing initiative. *The Computer Bulletin*. 2(6). August.
- Stamper R.K. (1973). *Information in Business and Administrative Systems*. Batsford, London.
- Weber M. (1947) *The Theory of Social and Economic Organisation*. Free Press.

## 2.16 Exercises

1. Characterise an information system known to you in terms of the features of a human activity system.
2. Describe an example of an informal, formal and technical information system in some organisation known to you.

# **3 *Information Systems and Organisations***

## **3.1 Introduction**

Information systems are used in the context of organisations. It has become very much of a truism to state that in modern Western economies the success of organisations is frequently very much dependent on the success of its information systems. In this chapter we discuss a number of models for organisations and how the concept of an information system and the development of such information systems fits within the framework of each model.

## **3.2 Organisation Theory**

Researchers have been debating the issue of what constitutes an organisation constitute for at least a hundred years. Much of this debate can be discussed in terms of a number of distinct perspectives on organisations. In this chapter we call each such perspective a model of organisations (Jirotka *et al.*, 1992). Each model determines its own particular slant on the use and utility of information systems.

### ***3.2.1 Organisations as Structures***

The most pervasive model of organisations sees them as structures that are divided into parts or functional departments. Each department is characterised by a pattern of precisely defined jobs. Jobs are organised in a clear hierarchical fashion with designated lines of authority from superior to subordinate. This hierarchical structure is so well formalised that it can be captured in an ‘organisational chart’ (Fayol, 1949).

This classical account of organisations is related to the ideas underlying the idea of ‘scientific management’ popularised by Frederick Taylor in the US in the early years of this century (Taylor, 1911). Scientific management focuses on three principles:

1. Managers should be given total responsibility for the organisation of work; workers should concentrate on manual tasks. A ‘thinking’ department of managers should be set up to be responsible for task planning and design.
2. All tasks should be examined and if necessary redesigned to improve efficiency. By studying the approach adopted, the tools utilised and the fatigue generated for any task, an optimum procedure for the task can be generated.

3. Methods should be adopted for the selection, training and monitoring of labour to ensure that work is done efficiently.

The underlying assumption of this approach is that the actions of people can be rationalised and closely defined. This model is clearly associated with that type of organisation known as a bureaucracy (chapter 2).

### ***3.2.2 Organisations as Networks***

As a reaction to Taylorism, attention shifted within organisation theory from a focus on tasks to a focus on ‘human’ issues. The famous Hawthorne studies of the 1920s and 1930s were initially concerned with discovering the effects of fatigue, accidents and labour turnover on rest pauses and the physical conditions of work (Mayo, 1933). However, as they progressed the studies became broader in outlook and investigated other aspects of the work situation such as workers’ attitudes and their social environment and the effect of such factors on worker productivity.

The significance of the Hawthorne studies is their interest in informal organisations based on friendship networks and spontaneous, unplanned interactions between members of such networks. This informal organisation is seen as working in parallel with, and sometimes opposed to, the formal organisation (chapter 2).

### ***3.2.3 Organisations as Environments***

Many organisational theorists have insisted that the environment of an organisation is essential to understanding how an organisation is defined both in terms of its relationships with the outside world and in terms of its internal structure. The organisation itself is seen as an open system made up of a series of interdependent subsystems with problematic interfaces. This means that organisations need sensitive management in order to balance internal needs with external environmental changes. It is management’s responsibility to obtain a good fit between the task of the organisation, the environment in which the task will be performed and the style of management.

One variant of this approach is to try to identify various relationships between successful organisational forms and different environments. Burns and Stalker (1961), for instance, suggest that mechanistic or bureaucratic forms of organisation are appropriate in stable environments; organic, or less formalised organisations are more appropriate in uncertain environments. Hence, in public sector environments where change is relatively slow one would expect mechanistic forms to thrive; in private sector environments that are more volatile one would expect more organic forms to exist.

### ***3.2.4 Organisations as Information Processors***

This view of organisations emphasises the importance of studying decision making, and more generally information processing, on the part of management.

Simon (1960) explores the parallels between human and organisational decision making. He maintains that organisations can never be truly rational, because members of organisations have limited information-processing abilities. Individuals have to take decisions on the basis of limited information about possibilities and consequences. They are also limited in the number of alternatives they can consider at any one time, and cannot accurately predict the outcomes of their actions. Individuals can therefore only achieve a bounded rationality in their decision-making. They *satisfice*, that is, they plan a course of action based upon good-enough decisions informed by rules-of-thumb and limited information. According to Simon, the concept of bounded rationality is institutionalised in the structure of decision-making in organisations and such decision-making is routinised and fragmented to make it more manageable. Hierarchies of control within organisations are there to simplify communication channels.

### ***3.2.5 Organisations as Constructions***

This perspective visualises organisations as social constructions. Organisations are seen as emerging from the day-to-day accomplishment of the tasks and responsibilities undertaken by participants (Silverman, 1982). Organisational theorists using this particular perspective are concerned with the relationship between work and an individual's self and identity. They are particularly interested in how participants account for their actions and present themselves and others. For instance, researchers have particularly elucidated the way in which members of distinct occupational groups and professions within health-care organisations use a particular ideology to establish a working consensus and co-ordinate their activities in terms of patient care.

### ***3.2.6 Organisations as Productions***

Several researchers have sought to detail the methods that members in settings use to account for their work (Garfinkel, 1967). Such researchers are particularly interested in how participants achieve their work and reason practically about their work. For instance, recent studies of air traffic controllers have revealed the important information embedded in the physical movement of artefacts in the control setting.

### **3.3 Models of the Organisation and Information Systems**

In this section we illustrate how each model of organisations critically affects the way in which we perceive the character and place of information systems within organisations. Note that the models are not independent, and hence certain models overlap in their conception of information systems.

#### ***3.3.1 Organisations as Structures***

The structural or systems model of organisations has been a particularly dominant model of organisations for information systems work. As we have seen in chapter 2 the idea of an information system and a human activity system have been inextricably interlinked by Checkland. Every organisation can be considered as a human activity system or as a series of human activity systems. This means that every organisation will have a number of information systems of some form.

An organisation can be considered on the macro level as a system. On the micro level, a business organisation can be considered to be made up of a number of subsystems or functional areas. For example:

1. *Marketing*. The area concerned with advertising, selling, pricing and distribution.
2. *Accounting and Finance*. The area concerned with generating and investing money and its accounting.
3. *Production*. The area concerned with purchasing raw materials, inventory control and manufacture.
4. *Research and Development*. The area concerned with developing new products or services and improving existing products or services.

The systems model of organisations has also particularly affected the way in which many information systems engineers (chapter 7) traditionally conceive of appropriate ways for constructing information technology systems. A long-standing problem in software engineering is to identify the functions that an information technology system should perform. One solution is to analyse organisational work in terms of tasks and specify the functions of an information system in terms of these tasks. Such task analysis may also later be used as a basis for deciding on an appropriate division of labour between computerised systems and the users of such systems.

#### ***3.3.2 Organisations as Networks***

This perspective emphasises the importance of informal organisation. It crucially determines the rationale for approaches such as Soft Systems analysis (chapter

26). The importance of human interaction to effective work in organisations has also particularly determined the socio-technical approach to developing information systems (chapter 28). This approach emphasises the importance of user participation and co-operation to the design, implementation and use of computerised information systems. As a guiding principle, social systems are designed in parallel and in correspondence with technical systems.

### ***3.3.3 Organisations as Environments***

Information is not only of internal relevance to organisations. In many domains the information flowing between organisations is a significant force in the economy. Consider the case of the Stock Market in the UK. The Stock Market can be conceived of as a vast information system concerned with the management of monetary relations between organisations. The very stuff of Stock Market activity is the management of exchange relations via information.

The importance of environment in the 'health' of organisations has become a particularly important emphasis in ideas underlying strategic information systems (chapter 33). Strategic information systems are those information systems that have an effective part to play in improving the interface between an organisation and its environment. For instance, many strategic information systems have been built which attempt to improve the relationship of customers with an organisation. For instance, American Hospital Supply placed information technology into its customers at no charge in order to capture the lucrative medical supplies market in the US.

The rationale for business process reengineering (chapter 25) is also frequently cast in terms of restructuring the way in which an organisation works, the purpose being to make the organisation more effective in terms of rapid changes in its environment (Hammer and Champy, 1993).

### ***3.3.4 Organisations as Information Processors***

In the systems approach each of the subsystems making up an organisation must be managed. The primary activity of management is decision-making. Business decision-making can be broken down into five areas: planning, organising, controlling, communicating, and staffing. All such activities are reliant on information.

Probably in this perspective above all others the importance of information as the life-blood of the organisation is pre-eminent. Information systems of whatever form are seen as being normally put in place to facilitate the process of decision-making within organisations. Decision-making in organisations is frequently divided into three levels: the strategic level, the administrative level and the operational level. Data flows through operational to the administrative to the strategic levels and feeds the decision-making process at each level.

Consider a food supermarket chain. At the operational level sales data is recorded at checkouts. This data allows administrative staff to record the amount of each type of product sold. In turn, a decision such as the amount of stock to reorder can be made. At the strategic level, the staff at headquarters can record nationally or perhaps internationally their sales performance in certain areas. This helps them to make decisions as to what they should be selling, where to site their stores, etc.

Information technology systems were first employed in the administrative level of organisations. In business organisations, the first functional area normally to employ computers was accounting and finance. Computers were used to record primarily administrative data related to the financial well being of organisations.

Information technology systems have now come to be employed not only in many other administrative areas but also in the operational and strategic levels of organisations. Point of sale and computer-aided manufacturing systems are two types of information system employed at the operational level. Many operational level systems are what we might call transaction-processing systems. Such systems are characterised by a heavy throughput of small changes (called transactions) made against a database. Management information systems (MIS) are probably the most prevalent form of information system employed at the strategic level. A classic MIS will be involved in producing summary information from mass transaction data. More recently the idea of a decision support system (DSS) or executive information system (EIS) has evolved. Such systems differ from an MIS in that they enable managers to more flexibly structure the information needed to make strategic decisions.

### ***3.3.5 Organisations as Constructions and Productions***

The model of an organisation as a social construction and as a production has only recently begun to inform the literature on information systems work. However, there are certainly difficulties inherent in the mismatch between the rich accounts of organisational life produced by researchers using these approaches and the pragmatic needs of information systems engineers. Nevertheless, such approaches have begun to inform the discipline of computer-supported co-operative work (Winograd and Flores, 1986). Here, the emphasis is on building computer systems that aid, support and enhance the process of human communication rather replacing it. It has been particularly discussed in the context of flatter, more democratic, forms of organisational structure. Many people have suggested that this looser form of organisational structure is more likely to prove successful in modern information economies because of its ability to adapt in times of rapid change.

### 3.4 Conclusion

People in modern western societies live out a substantial part of their lives in one form of organisation or another. The idea of an organisation can be considered from a number of different perspectives:

1. organisation as system
2. organisation as network
3. organisation as environment
4. organisation as information processor
5. organisation as construction
6. organisation as production

The first four perspectives have crucially informed the idea of information systems and their relevance to organisations. The last two approaches are beginning to have an influence, particularly in the context of more radical forms of organisational structures that we discuss in more detail in chapter 37.

### 3.5 References

- Burns T. and Stalker G.M. (1961). *The Management of Innovation*. Tavistock. London.
- Fayol H. (1941). *General and Industrial Management*. Pitman. London.
- Garfinkel H. (1967). *Studies in Ethnomethodology*. Polity, Cambridge.
- Hammer M. and Champy J. (1993). *Reengineering the Corporation: a manifesto for business revolution*. Nicholas Brearley, London.
- Jirotka M., Gilbert N. and Luff P. (1992). 'On the Social Organisation of Organisations'. *Computer Supported Co-operative Work*. 1(1). 95-118. Kluwer Academic.
- Mayo E. (1933). *The Human Problems of an Industrial Civilisation*. Macmillan, New York.
- Silverman D. (1982). *The Theory of Organisations*. Macmillan.
- Taylor F.W. (1911). *Principles of Scientific Management*. Harper and Row, New York.
- Winograd T. and Flores F. (1986). *Understanding Computers and Cognition: a new foundation for design*. Ablex, Norwood, NJ.

### 3.6 Exercises

1. Consider an organisation known to you. Use one or more of the perspectives above to characterise the organisation. For instance, would you class it as a bureaucratic or an organic organisation?

2. Which are the dominant perspectives employed by information systems people within the chosen organisation? In what way are information systems characterised in the organisation: e.g. as support systems, as decision-making tools, as co-ordination systems or as strategic weapons?

# **4 Information Society and Economy**

## **4.1 Introduction**

In the previous chapter we discussed how the shape of an organisation is affected by its environment. In the normal sense of the term used within the Information Systems area, environment means the competitive marketplace. In this chapter we broaden the meaning of this term to include the way in which information and information systems are affecting societies and economies.

## **4.2 The Information Society**

In the 1970s Daniel Bell (1970), a US sociologist, wrote an influential book entitled *The Coming of Post-Industrial Society*. In this work, Bell made a series of predictions about the state of Western societies. His major premise is now generally accepted that Western societies are becoming information societies. He maintained that whereas the revolution of the latter part of the nineteenth century was an industrial revolution, the revolution of the latter part of the twentieth century is an information revolution.

However, there is some debate about whether information technology will bring about the same level of changes in society as characterised by the industrial revolution. One school of thought believes that we are clearly moving out of the stage of an industrial society and into the stage of a post-industrial society. Members of this school point to a number of indicators present in contemporary Western societies that support this thesis: the change from a goods-producing society to a service society; that information is becoming a major commodity for organisations; that there has been an information explosion.

The second school of thought believes that the magnitude and speed of changes in society have been heavily exaggerated. They maintain that we are seeing just one more stage in the development of industry comparable to developments that have gone before. Although the growth in the service sector in Western economies is significant, it has not removed the need for goods-producing industries. Gershuny (1978), for instance, analysed employment and consumption patterns in the UK during the 1970s and found evidence that the gradual emergence of a service society was a myth. He finds that a fall in service consumption is balanced by a rise in the consumption of durable goods, which in effect allows consumers to produce services for themselves.

### **4.3 Information as a Commodity**

Much discussion has been made of the commoditisation of information. However, Bell (1979) believes that information is not a commodity in the sense of industrial commodities. Industrial commodities are produced in discrete, identifiable units, and are exchanged and sold to be consumed. Classic industrial commodities are loaves of bread and automobiles. Such commodities are characterised by the fact that one buys the product from a seller and thereby take physical possession of it. The exchange is also governed by legal rules of contract.

Bell bundles together questions of information with those pertaining to knowledge. The problem with information or knowledge is that even when it is sold, it remains with the producer. Bell sees it as a collective commodity in that when it is created it is by its very nature available to all. For information to be a commodity, information must have value. Not all information may be valuable or useful. What determines the value or utility of information is clearly dependent on the particular perspective of some individual or group. The idea of determining the use-value of information is one approach to attempt to commoditise information.

The use-value of some types of information as a commodity for social control has caused concern amongst civil liberties groups in many countries. Modern information technology permits organisations to collate together information from various sources to form profiles of individuals or groups.

### **4.4 Transactional Information**

Information technology systems have altered the nature of society's records. Burnham (1983) particularly illustrates the way in which transactional information can be used to record the daily lives of almost every person in the United States. This information can be combined with more traditional kinds of information such as people's age and place of birth to permit organisations to make decisions about the planning of their work.

#### *Case*

The American Telephone and Telegraphic Company (AT&T) has built a large network of linked databases recording information on the telephone calls that people make in the United States.

Access to this database gave a senate committee of the US Congress additional ammunition in its investigation of the relationship between President Carter's brother Billy and the government of Libya. Almost every page of the final report to the Senate Committee contains footnotes to the precise time and day of calls made by Billy Carter and his associates.

AT&T itself used the database to keep track of the companies that decided to hook into the long-distance facilities offered by one of its competitors. It also appears to have used the database as a means of determining the customer profile of its competitors.

#### *Case*

One of the major UK supermarket chains maintains a large database of retail transactional information produced from its point-of-sale machines in each supermarket. This information is used not only to automatically determine stock reordering but also to determine the major purchasing strategy for the chain as well as the siting of new stores.

#### **4.5 Data Protection**

The rise of transactional information has brought to the fore the issue of ensuring the privacy of data held about an individual. In the UK, for instance, the Data Protection Act 1984 lays down a number of principles that maintain that personal data must be:

1. obtained and processed, fairly and lawfully
2. held for the lawful purposes described in the data user's register entry
3. used for those purposes and disclosed only to those people, described in the register entry
4. adequate, relevant and not excessive for the purposes for which it is held
5. accurate, and where necessary, kept up to date
6. held for no longer than is necessary for the registered purposes
7. accessible to the individual concerned who, where appropriate, has the right to have the information corrected or erased
8. surrounded by proper security

All organisations that maintain personal data must register the data held with the data protection register and are obliged to ensure that their use of such data conforms to the principles above.

Many other European countries such as those in Scandinavia also have data protection legislation in place. In 1998 the UK government intends to implement new legislation to bring the act in line with the data protection directive of the European Union. The new law will include an extension to a number of data subject's rights such as the processing of special categories of data will be subject to more restrictive rules. An example here is data revealing racial or ethnic origins. Also, some coverage of manual records will be included in the new act (Hinde, 1998).

## **4.6 The Information Economy**

In chapter 2 we discussed how information systems support human activity systems. The economy of a country can be considered as a massive human activity system. Information has become an important resource in the way in which modern economies work. For some organisations, information is an important commodity in itself.

### ***4.6.1 The Information Business***

The trend of information becoming a major commodity for organisations is evident, for instance, in the work of the advertising industry. Advertisements for particular products will be placed within television programmes that are watched by a significant proportion of people who are likely to buy such products. The advertising company makes decisions about which adverts to place where on the basis of information from market researchers who conduct detailed surveys of viewing behaviour. In this context, information has become a currency that determines the cost of particular advertisements. Adverts placed within programmes watched by peak viewing populations clearly become more costly to place than those embedded within less watched programmes.

The news agency Reuters is another example of an organisation where information is its life-blood. Reuters prepares and sells news reports to broadcasting companies, newspapers and other news outlets around the world. In this sense, information is Reuters' product or commodity.

### ***4.6.2 Virtual Organisations***

Some people have argued that many modern-day organisations are being completely changed by the 'information revolution', almost to the point where they cease to look like a traditional industrial or commercial organisation. They become virtual or network organisations.

Some characteristics of this form of organisation are:

1. Information is a major commodity as well as a resource for the organisation.
2. Parts of the organisation may be geographically dispersed, but they are held together by communication networks.
3. Links with suppliers and customers are conducted electronically. Hence the activity of the organisation can be sited in locations geographically remote from its customers or suppliers.
4. Information technology is central to the activity of the organisation.

***Example***

The business of a high-street bank is information. It is an organisation whose purpose is the management of information for its customers – recording money transactions. The rapid rise of automatic teller machines or ATMs in the UK is one indicator of the way in which high-street banks are heavily utilising information technology to reduce overheads. Many banks are looking to replace the traditional branch network with a series of services provided centrally, currently over the phone, but in future via interface units based in the home.

***Example***

The UK bank First Direct, a subsidiary of the Midland bank, is an example of this change in banking organisation. First Direct has no branches. All business with its customers is conducted via the telephone, by fax or via ATMs. This allows the company to offer a 24 hour, 365 days a year service to its customers. Many other financial institutions in the UK are now beginning to emulate these activities.

#### ***4.6.3 Information Highways***

Much discussion has recently centred around the construction of national information infrastructures both in Europe and in the United States. Such communication infrastructures are deliberately compared to a transport network. The vision of the information highway is that it would comprise a standard communications network by which a vast range of services could be offered to consumers. Commercial organisations such as high street retailers could offer purchasing of goods from the home. Public sector organisations might offer services such as electronic tax forms, job vacancies, remote consultation by GPs, or access to members of parliament.

Many have even portrayed information highways as the backbone of economies in the 21st century. The argument is that information highways will affect the traditional consumer-producer relationship between organisations themselves, as well as between customers and organisations. For instance, the ordering of materials and resources for production could be done via the highway rather than by post or by telephone.

#### ***4.6.4 Inter-organisational Information Systems***

One feature of the growing importance of information and information systems to modern economies is the growth in inter-organisational information systems. An inter-organisational information system (IOS) is an automated information system shared by two or more organisations (Barrette and Konsynki, 1982). An

IOS consists of a computer and communications infrastructure that permits the sharing of an application, such as reservations or ordering supplies. An IOS can be a particularly effective way for organisations to share the costs of developing and administering an information system. The Internet (chapter 13) is being seen by many as a particularly potent force in encouraging the growth of IOS.

The organisations involved in an IOS may be either participants or facilitators. Whereas participants may be competitors, or organisations in a buyer-seller relationship, a facilitator is an organisation that develops and operates an IOS for other organisations' use.

### *Examples*

1. American Airlines' Sabre reservation system was initially developed by AA for its own use. It is now operated by AA for other airlines' use.
2. An automobile manufacturer in the USA maintains computer-to-computer communication to its primary suppliers to implement just-in-time inventory systems.
3. A number of large hotel chains in the USA have built links between their reservation systems and car rental systems.

## **4.7 Conclusion**

Some have proposed that whereas the revolution of the latter part of the nineteenth century was an industrial revolution, the revolution of the latter part of the twentieth century is an information revolution. However, there are a number of difficulties with treating information as a classic economic good. Nevertheless, transactional information is particularly important to modern economies. There are also a number of problems, particularly in the area of personal liberty and the right to privacy, associated with the rise of transactional information. Modern information technology may cause many organisations to become virtual in the sense that they restructure around the application of information technology. The construction of information highways may radically change the nature of global economies.

## **4.8 References**

- Barrette S. and Konsynski B.R. (1982). 'Inter-Organisational Information Sharing Systems'. *MIS Quarterly*. Fall.
- Bell D. (1979). 'The Social Framework of the Information Society'. In Dertouzos M. and Moses J (Ed). *The Information Society*. MIT Press. Cambridge, Mass.
- Burnham D. (1983). *The Rise of the Computer State*. Random House, New York.

- Gershuny J. (1978). *After Industrial Society: the emerging self-service economy.* Macmillan, London.
- Hinde A. (1998). Time to get Personal. *Computer Bulletin*. January.

#### 4.9 Exercises

1. Find an example of a network organisation other than the examples given.
2. Consider how some organisation known to you might be affected by the existence of an information highway.
3. Find and describe some example of an inter-organisational information system not described in this chapter.
4. To what extent do you think that a country such as the UK can be termed an information society?

# **5 Information Technology**

## **5.1 Introduction**

In chapter 2 we described three levels at which information systems may be addressed. In most modern-day organisations, the lowest level of information system, that of a technical information system or an information technology system, has become increasingly important. In this chapter we discuss at a reasonably high level the component parts of such systems.

## **5.2 Hardware**

This book is primarily devoted to the process of developing information systems from software, data and people components. The other component in the equation, namely hardware or electronic computing machinery, is discussed in numerous other texts (Chalk, 1996). In this section we discuss the minimum of concepts necessary to appreciate the makeup of the hardware and software underlying information technology systems.

### **5.2.1 Turing Machines**

A modern-day computer is a physical implementation of an abstract machine known as a Turing machine, so called because it was proposed by Alan Turing in the 1930s (Hodges, 1983). Turing conjectured that a Turing machine possesses the power to solve any problem that is solvable by computational means. Hence, Turing machines are generally described as being ‘universal’ machines.

A Turing machine is made up of two parts: a control mechanism and an input medium (Brookshead, 1989). The control mechanism can be in one of a finite number of states at any time. One of these states is deemed to be the initial state and another the halt state of the machine.

The input medium was originally characterised as an infinitely long piece of paper tape. The machine is equipped with a tape head that is used to both read and write symbols on the paper tape. A machine can move the tape both backwards and forwards through the tape head.

The individual actions that can be performed by a Turing machine consist of write actions and move actions. A write operation consists of replacing a symbol on the tape with another symbol and then shifting to a new state. A move operation consists of moving the tape head one cell to the right or one cell to the left and then shifting to a new state. Which action will be performed at a

particular time depends on the current symbol in the cell visible to the tape head as well as the current state of the machine's control mechanism.

Turing's original 'machine' was a human acting with pencil and paper. It was only with the advent of electronic equipment that the idea of Turing machines could be practically realised. However, the important point is that the theoretical computational abilities of the system remain the same, regardless of the technology. In fact, Turing's model is more general than today's electronic computers, since a Turing machine is never restricted by lack of storage space (it uses a tape of infinite length), whereas an actual machine must employ finite storage, however great.

The other important characteristic of a Turing machine is that it demonstrated that any computational process can be 'programmed'. That is, a set of instructions can be given to such a machine to determine its behaviour.

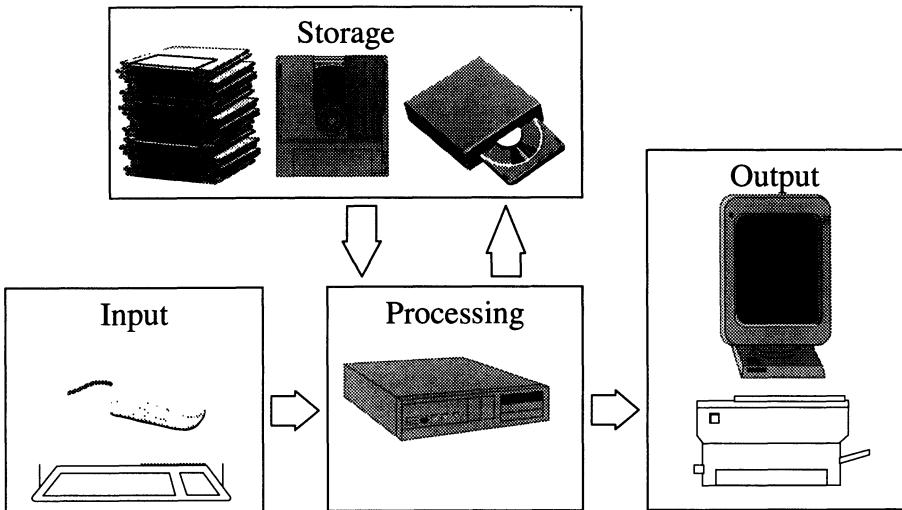
### **5.2.2 The Components of Hardware**

Figure 5.1 illustrates the four primary components of computing hardware:

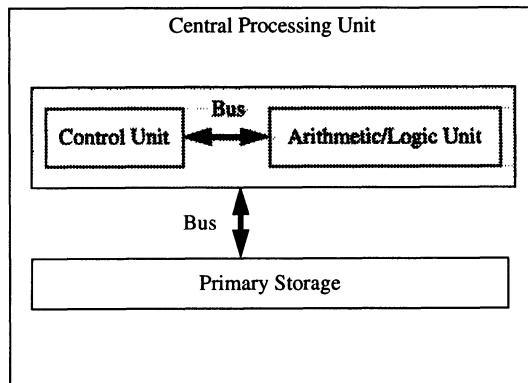
1. Input. Via devices such as keyboards, mice, joysticks etc.
2. Processing. The central processing unit (CPU).
3. Storage. Devices such as hard disk drives, floppy disk drives, CD-ROM drives.
4. Output. Usually via monitors or printers.

The workhorse of a computing machine is the CPU. The central processing unit can be subdivided into the following components (figure 5.2):

1. The Control Unit directs and co-ordinates the rest of the system in carrying out program instructions.
2. The Logic Unit calculates and compares data, based on instructions from the controller.
3. The Primary Storage Unit holds data for processing, instructions for processing and processed data waiting to be output.
4. Communication between these three components is effected by physical connections known as buses.



**Figure 5.1: Components of a Computer System**



**Figure 5.2: The Central Processing Unit**

To this we might add that modern-day computers are seldom purely standalone devices. Most computers, particularly in business organisations, are connected to other machines via short-haul or long haul communication lines. A collection of machines connected by short-haul communication lines is known as a local area network. A collection joined by long-haul communication lines is known as a wide area network. Computer networks are a major component in the prevalent strategy of distributed computing, i.e. siting computer power where it is needed. Hence output devices such as printers or storage devices such as high capacity disk drives can be shared across the network. This situation is illustrated in figure 5.3.

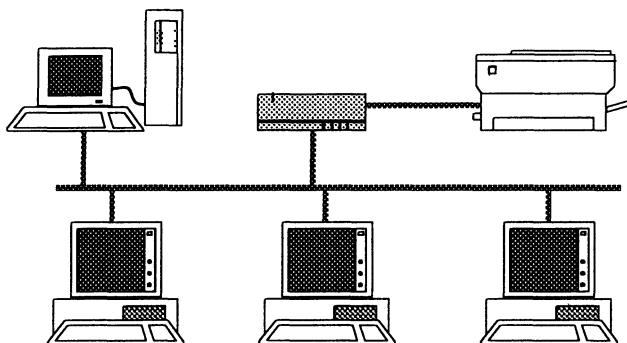


Figure 5.3: A Computer Network

### 5.3 Software

Software is computer programs. A computer program is a series of instructions for hardware. Programs transform the universal machine embodied in hardware into a machine specialised for some task.

It is useful to delineate a number of different levels of software, of which the most important are:

1. *Operating systems*. These are programs that supervise the running of all other programs on some hardware. The operating system undertakes tasks such as scheduling the running of programs, controlling input and output to programs, and managing files on secondary storage.
2. *Programming languages*. These are formal languages for writing instructions. There are various levels of programming languages as discussed in chapters 8 and 9. Programming languages are part of a class of information system development tools that we consider in part 2.
3. *Applications*. A software application or application system is normally used as another term for an information technology system. An application system is a system, normally written using some language or tool-set, and designed to perform a particular set of tasks for some organisation.

We can also distinguish between software that has been produced for the mass market, so-called *shrink-wrapped software*, and software that has been produced specifically for the purposes of some organisation, so-called *bespoke software*. Falling between these two categories is the class of software known as *packaged software*. This is software which is written to handle some generic organisational function such as order processing, but which can be tailored to the specific needs of some organisation.

## 5.4 Layers of an Information Technology System

It is useful to consider an information technology system as being made up of a number of subsystems or layers:

1. *Interface subsystem*. This subsystem is responsible for managing all interaction with the end-user.
2. *Rules subsystem*. This subsystem manages the application logic in terms of a defined model of business rules.
3. *Transaction subsystem*. This subsystem acts as the link between the data subsystem and the rules and interface subsystems. Querying, insertion and update activity is triggered at the interface, validated by the rules subsystem and packaged as units (transactions) that will initiate actions (responses or changes) in the data subsystem.
4. *Data subsystem*. This subsystem is responsible for managing the underlying data need by an application.

In the contemporary IT infrastructure each of these parts of an application may be distributed on different machines, perhaps at different sites. This means that each part usually needs to be stitched together in terms of some communications backbone. For consistency, we refer to this facility here as the communications subsystem (see figure 5.4).

Some people have also argued that a typical application is also like an iceberg. The user interface only forms the tip of the iceberg (some 10% of the application). The major element of the application is that part of the iceberg that is hidden from view, the 90% below the water.

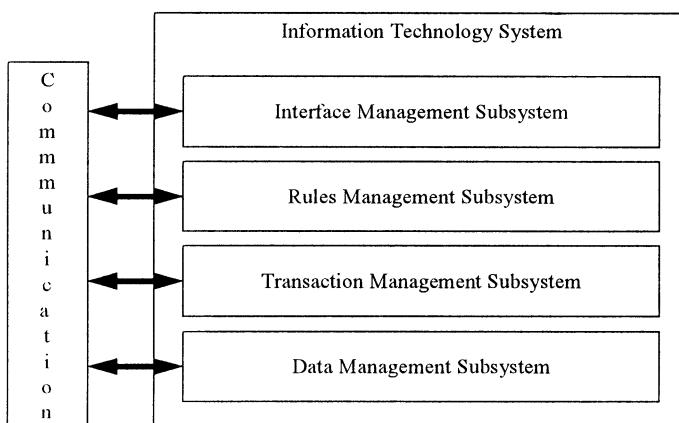


Figure 5.4: Layers of an Information Technology System

## 5.5 Distributing Application Layers

Historically speaking, the component layers of an information technology system have been sited at different parts of a technological infrastructure.

1. *Batch systems.* In the first phase of information technology systems, each of the component layers of an information technology system was located on one machine. Large mainframe systems ran the data management, transaction management, rules management and much of the interface management functions of an application. Connection to such systems was via so-called ‘dumb’ terminals. They were referred to as dumb because they contained very little inherent functionality and primarily enabled operators to control systems via character-based interfaces (chapter 23). Most of the processing on such systems was conducted in batch mode, i.e. the processing of vast amounts of transactions in sequence with very little direct user input. Specialist data entry staff conducted data entry. Retrieval of information was available through paper reports produced from the system.
2. *On-line systems.* Technological developments enabled the user interface layer and some of the rules management layer to be located on so-called ‘intelligent’ terminals. They were referred to as ‘intelligent’ because they were able to remove some of the processing from the centralised mainframe or minicomputer. This advance enabled the development of on-line systems. Such systems enabled users in the workplace to enter data directly into the information system and in certain respects to query the data in the system. The rise of on-line systems enabled the development of management information system (MIS) and decision support system (DSS) applications.
3. *Client-Server systems.* With the rise of personal computers much more of the functionality of the information technology system began to be placed on desktop machines. The sophistication of graphical user interfaces (chapter 23) meant that more power needed to be available on the desktop to run such interfaces effectively. With the rise of software for the desktop, slices from the total functionality of an information technology system could be built using application development tools available for the desktop.

## 5.6 Constructing the Application

Historically, the four parts of a conventional IT application were constructed using one tool, the high-level or third generation programming language (3GL). A language such as COBOL was used to declare appropriate file structures (data subsystem), encode the necessary operations on files (transaction subsystem), validate data processed (rules subsystem) and manage the terminal screen for data entry and retrieval (interface subsystem). However, over the last couple of decades there has been a tendency to use a different, specialised tool for the

construction of one or more of these layers. For instance:

1. Graphical User Interface tools have developed as a means of constructing sophisticated user interfaces (chapter 23).
2. Fourth generation languages have developed as a means of coding business rules and application logic (chapter 11).
3. Transaction processing systems have developed to enable high throughput of transactions (chapter 10).
4. Database Management Systems have developed as sophisticated tools for managing multi-user access to data (chapter 10).
5. Communications are enabled by a vast range of software supporting local area and wide area communications.

All of these developments mean that the contemporary information technology system is a hybrid. In other words, it is built using a range of distinct technologies.

## 5.7 Specifying the Application

Our model of information technology systems is useful in identifying a number of different streams of information systems analysis and design.

Any technical information system can be viewed in two distinct but complementary modes: the dynamic view and the static view.

1. The dynamic view emphasises the importance of process; how data flows through an information system and is transformed by the various processes making up the information system. This embodies what we described above as the transaction subsystem and the rules management subsystem.
2. The static view emphasises data; how data is or should be organised in order to support organisational processes. This embodies what we have described above as the data management subsystem.

We document the dynamic view of an information system in a process model. We document the static view of an information system in a data model. Both the data model and the process model are needed to form a complete information systems model. The activity of producing a data model is known as data analysis (part 3, section 1). The activity of producing a process model is known as process analysis (part 3, section 2).

To the process model and data model we should also add the user interface model. A great deal of literature has now developed surrounding the analysis and design of user interfaces. We discuss this material in chapter 23.

## 5.8 Goronwy Galvanising

In this section we discuss further the current system at Goronwy Galvanising.

Black products are delivered to Goronwy on large trailers in bundles referred to as batches. Each trailer may be loaded with a number of different types of steel product. Each batch is labelled with a unique job number. Each trailer is given its own advice note detailing all the associated jobs on the trailer.

Goronwy mainly process lintels for a major steel manufacturer, Blackheads. The advice note supplied with lintels is identified by an advice number specific to this manufacturer. Each job is identified on the advice note by a job number generated through Blackheads' check digit routine. Figure 5.5 represents a typical advice note received from Blackheads detailing the black material on a trailer.

Blackheads Steel Products		Despatch Advice			
Advice No.	A3137	Date	11/1/1998		
Customer Name & Address	Goronwy Galvanising	Instructions	Galvanise and return		
Order Number	Description	Product Code	Item Length	Number	Weight (Tonnes)
13/1193G	Lintels		1500	20	
44/2404G			1500	20	
70/2517P			1350	20	
23/2474P			1200	16	
Hauliers Name	International 5	Received in good order			

Figure 5.5: Delivery Advice

Smaller manufacturers such as Pimples supply an advice note in which jobs are identified by a concatenation of an advice number and line number on which the job appears. Each job, whether it be for Blackheads or Pimples, is also described on the advice note in terms of a product code, description, item length, order quantity and batch weight. Each advice is dated.

On arrival at the plant the material is unpacked and checked for discrepancies between the material indicated on the advice note and the material actually on the trailer. Two major types of discrepancy are important. A count discrepancy occurs when the number of steel items actually found is less than or greater than the amount indicated. A non-conforming black discrepancy arises when some of the material is unsuitable for galvanising. For instance, a steel lintel may be bent. Such discrepancies are written as annotations to the advice note.

When all material has been checked, the advice note is given to the production controller. He and the office clerk transcribe details, including any discrepancies, from the advice note onto job sheets. An example job sheet is illustrated in figure 5.6. The job sheet is passed down to the shop floor where the shift foreman uses it to record details of processing.

Job Sheet						
Order No.	Description	Product Code	Item Length	Order Qty	Batch Weight	
13/1193G	Lintels	UL15	1500	20		
Count Discrepancy	Non-Conforming Black	Non-Conforming White	Non-Conforming No Charge			
Galvanised	Despatch No.	Despatch Date	Qty Returned	Weight Returned		
Y						

**Figure 5.6: Job Sheet**

Most jobs will pass through the system smoothly. The steel items will be placed on racks, dipped in the zinc bath and left to cool. The site foreman will then check the condition of the job. If all items have galvanised properly he will tick the job completed box on the job sheet and pass it back to the production controller.

Occasionally, some of the items will not have galvanised properly. Such items are then classed as non-conforming white, and indicated as such on the job sheet. These items will normally be regalvanised at some later date.

When Goronwy have treated a series of jobs the production controller will issue a despatch note and send it down to the shop floor. Workers will then stack the white material on trailers ready to be returned to the associated manufacturers. Each trailer will have an associated advice note detailing the material on the trailer. Partial despatches may be made from one job. This means that the trailer of white material need not correspond to the trailer of black material originally supplied to Goronwy. The despatch advice is given a unique advice number and is dated. Each despatch details the job number, product code, description, item length, batch weight, returned quantity and returned weight. Figure 5.7 represents a despatch advice prepared for Goronwy for a collection of white material.

<b>Goronwy Galvanising</b>		<b>Despatch Advice</b>					
Advice No.	101	Date	12/12/1997				
Customer Name & Address	Blackheads						
Order Number	Description	Product Code	Item Length	Order Qty	Batch Weight	Returned Qty	Batch Weight
13/1193G	Lintels	UL15	1500	20	150	20	150
44/2404G		UL15	1500	20	150	20	150
70/2517P		UL135	1350	20	135	20	135
23/2474P		UL12	1200	16	100	14	82
Driver	Received by:						

Figure 5.7: Despatch Advice

## 5.9 References

- Brookshear J.G. (1989). *Theory of Computation: Formal Languages, Automata and Complexity*. Benjamin-Cummings, Redwood City.
- Hodges A. (1983). *Alan Turing: The Enigma*. Hutchinson, London.
- Chalk B.S. (1996). *Computer Organisation and Architecture*. Macmillan Press, Basingstoke, Hants.

## 5.10 Exercises

1. Why are computers referred to as ‘universal’ machines?
2. Try decoupling the data, transaction, user interface, and rule subsystems of some information technology system with which you are familiar.
3. What advantages do you think arise from separating out the four components of an information technology system?
4. Try to find one example of an operating system, programming language and piece of applications software in an organisation known to you.
5. Try to find one example of shrink-wrapped, bespoke and packaged software in an organisation known to you.

# **6 Information Systems Services Function**

## **6.1 Introduction**

Most medium to large scale organisations have people specifically employed in IS work. In this chapter we consider a number of important issues in relation to how this function is or should be organised. The name of such a function will vary amongst organisations. In some organisations it may be called the IT or IS department, perhaps even the DP (data processing) department. In this chapter we refer to it generically as the IS services function, to emphasise that in most organisations IS is a critical supporting or strategic service supplied to the organisation.

## **6.2 Activities of the IS Service**

The IS services function, depending on the size of the organisation itself, may engage in a vast range of different activities. Here we list some of the core activities that the IS service may be engaged in:

1. *IS planning.* Planning the new IS portfolio. By portfolio we mean the range of information systems needed by the organisation (chapter 33).
2. *IS management.* Managing the existing IS portfolio (chapter 32). That is controlling the process of supporting and maintaining existing IS. This process will also frequently include issues of evaluating the investment in IS (chapter 34).
3. *Project management.* The management activities associated with individual information systems projects (chapter 31).
4. *Conducting IS development.* The activity of constructing and delivering new application systems: analysing, designing, producing, testing and implementing systems (chapter 7).
5. *Conducting IS maintenance.* This area corrects bugs in existing information systems and tailors/tunes existing IS to match with changes in the working environment of the organisation.
6. *Conducting IS operations/support/administration.* This area operates large, multi-user systems and/or supports the use of IS throughout the organisation. One crucial area of contemporary support is associated with the administration of corporate databases (chapter 10).
7. *Conducting new technology appraisal.* Because of the importance of IS to many organisations, a group within the IS services function may be tasked with investigating new technological possibilities.
8. *Human resources management.* Recruiting and organising IS personnel and

- maintaining staff development programmes to ensure the professional service of staff (chapter 39).
9. *Quality assurance.* Ensuring the quality of the IS resource and the artefacts that it produces and maintains (chapter 36).
  10. *IS evaluation.* Monitoring the success of IS projects and the IS service itself (chapter 34).

This is clearly not an exhaustive list but it indicates some of the dominant areas for consideration.

### 6.3 The Organisation of the IS Service

In calling this service a function we deliberately side step the issue of how this service is organised. In some organisations, particularly small organisations, there may be no actual section or department specialising in IS. However, in most modern-day medium to large-scale organisations some specialist structure is put in place for this service. Clearly there are a number of different ways in which such a service can be organised. In this section we consider the issue of organisation under four major headings: Division of labour; Location of IS service; Financial management; Relationship with user community

#### 6.3.1 *Division of Labour*

By division of labour we refer to the way in which various jobs or roles are delineated and structured within the IS service. Traditionally, the division of labour of the IS service has been one of quite rigid job specifications based around a hierarchical form of order. Over the last twenty years or so a number of pressures for change have caused the gradual fragmentation of this structure and newer structures for the IS service have emerged.

##### 6.3.1.1 *The Traditional IS Department*

Because of the exigencies of building and running information systems on large, centralised mainframes, the traditional information systems services department was structured in a hierarchy of clearly delineated job functions:

1. At the bottom of the hierarchy lie the operating staff. These staff are tasked with maintaining the operation of the centralised mainframe and the systems which run on it.
2. Next in the hierarchy come the programmers. These may be organised into groups such as maintenance programmers and development programmers. Development programmers construct new applications. Maintenance programmers repair and extend existing applications.

3. Systems analysts are the next rung in the hierarchy. These are persons primarily involved in the analysis and design of information systems. It is these persons who make contact with the business users.
4. Many organisations segmented staff further in terms of project teams of analysts, programmers and sometimes operators. The topic of project team organisation is discussed in more detail in chapter 31. Each such project team was normally headed by a project manager.
5. The IS department was headed by one person probably called something like the data processing (DP) manager. In a large organisation there were frequently a number of middle-level managers such as operations managers, development managers and maintenance managers, each co-ordinating a particular aspect of information systems work.

#### *6.3.1.2 Pressures for Change*

During the 1970s and 1980s a number of forms of pressure forced some radical changes on many information systems departments. These changes have accelerated during the 1990s.

1. *The end-user movement.* The increasing infiltration of desktop computers into organisations has meant that computing power can be sited wherever it is required. Desktop packages such as word-processing software and spreadsheets have been specifically written for the end-user. End-users have consequently gained much more experience of computing and have become much more confident in expressing their requirements, sometimes even building applications themselves.
2. *The integration movement.* The developing use of databases and in particular the database approach has meant that organisations have to plan for and manage data at the corporate level. Organisations see key added value from integrating information systems across sectors.
3. *The distribution movement.* Processing power and storage capacity no longer need to be centralised in organisations. Processing and storage can be distributed around the organisation on diverse sets of platforms and diverse software can be enabled to co-operate across local and wide area networks.

#### *6.3.1.3 The Information Centre*

In response to these pressures and others, the IS services function has recast itself in various different forms. One such form is the information centre: a corpus of IS expertise whose role is to service other departments which are heavily involved in handling a large proportion of their IS/IT themselves. This is in marked contrast to the traditional IS department described above which had a monopoly over organisational computing.

With the rise of different organisational structures for the IS service such as

the information centre, a greater diversification of IS staff has ensued. We now have job-titles such as Hybrid Managers, Analysts/Programmers, Database Administrators, Data Analysts, Business Analysts, and Systems Integrators, to name but a few. One particularly notable trend has been the growth in what might be called support staff. That is, those staff not directly tasked with developing any new software directly but tasked with installing and integrating existing software and helping end-users in the use of such software.

### 6.3.2 *Location of IS Service*

The rise of new organisational forms, such as the information centre, is partly a result of an increased number of options now open to the IS service in terms of its location. Below we list some of the dominant forms available:

1. *Centralised*. This is the traditional model in which IS provides one single service with single access provision. Hence the IS service is located in one large facility with all other organisational units relying on this facility for all forms of IS provision.
2. *Decentralised*. Here the IS service is provided by a number of smaller units each providing single access provision. Under this model the IS service still forms a logical whole, but the various functions that it provides, such as IS planning, management, development and maintenance, may be segmented off into separate organisational units.
3. *Distributed*. In this model the IS service is made up of a set of connected IS functions each providing a multiple of services. In particularly large scale organisations such as multinationals it may prove impossible to provide any one function on a centralised basis. Hence, for instance, each country might have its own IS services function, providing all of the activities described in section 6.2 to their national units.
4. *Devolved*. This model adheres to a distributed framework but each IS unit is not independent. Instead, the IS service is made up of a matrix of units each sited close to the point of need and falling under direct business unit control.
5. *Outsourced*. Frequently because of a perceived dissatisfaction with many of the above forms of organisation for the IS service on the part of the wider business, many organisations have recently pursued an outsourcing strategy. Here, the IS service provision is sited either in whole or in part with external contractors. This issue is discussed in more detail in chapter 35.

No single form of location for the IS service is a clear winner. Each of these forms of location has its own advantages and disadvantages. To illustrate this we consider the advantages and disadvantages associated with the centralised versus decentralised dichotomy.

*Advantages of centralised IS service*

1. Greater control can be exercised over the operation of IS resources
2. Recruiting and maintaining IS skills becomes easier in a large IS service
3. Economies of scale can be achieved in the procurement of hardware and software
4. A centralised IS is better able to consider issues of integration and the development of large infrastructure projects
5. Greater standardisation and compatibility of systems can be achieved
6. Duplication of effort is more easily avoided
7. Speedier and more consistent strategic decision-making is possible

*Disadvantages of a centralised IS service*

1. The IS service is more likely to be divorced from the ‘coal-face’ of the business
2. Some diseconomies may be possible, e.g. high backup costs
3. Dissatisfaction is more likely concerning the level of personal attention given to business departments, particularly in terms of rapid response to needs
4. A large, centralised IS resource may be less able to adapt quickly to changes in the business environment and technology

### ***6.3.3 Financial Management***

As stated above, information systems is normally a service function within organisations. Hence, some way normally has to be found of funding such a service. A number of the possible options for schemes of financial management are listed below:

1. *Free service.* Internal IS service is farmed out free of charge to other departments. Some arrangement whereby each department has its budget top-sliced to finance IS is familiar
2. *Profit centre.* IS bills other departments for its service
3. *Separate company.* The IS service is converted into a separate company which contracts its services to the parent and potentially to external customers
4. *Outsourced.* The IS service is given over either in whole or part to a different, specialist, IS company

Although IS is now seen to be central to most organisations, the IS service has traditionally experienced great difficulty in quantifying its benefits to the organisation. This issue is discussed in more detail in chapter 34.

## 6.4 Relationship with User Community

There has been much discussion of the ‘wall’ that exists between IS and the rest of an organisation. This we might even characterise as being the traditional approach and the purpose of IS under this model as being ‘do it to them’. In more recent times, the IS department has attempted to strike a more co-operative stance with its user community. Here we might distinguish a data management approach that defines the purpose of IS being ‘do it for them’ from the project-based approach, where the purpose is ‘do it with them’. Also, we have the idea of information centres in which the purpose of IS is ‘help them to do it for themselves’. Finally, there are externally focused departments where the purpose may be seen as maintaining the organisation’s information warehouse.

## 6.5 Conclusion

In this chapter we have considered a number of issues related to the organisation of the information systems services function. It must be recognised that the organisation of the IS service has an overarching effect on the way in which IS is planned for, managed and operates. For instance, if the IS service is distributed this has a crucial bearing on the way in which cross-organisational systems may be developed. Here, we have considered organisation in terms of its division of labour, location, financial management, and relationship with user community. A number of pressures for change has meant that different forms for the organisation of IS have arisen in recent times. Part of the reason for this is that the IS service has experienced a problem in justifying its existence.

Each of the different possibilities for the organisation of the IS service discussed in this chapter can be seen to be a historical reflection of features of the environment of commercial computing. In his global, historical study of the IS industry, Friedman (1989) has proposed that three phases characterise the history of commercial computing, each phase dominated by a different set of constraints:

1. *Hardware constraints* (up to mid 1960s). The phase in which hardware costs and limitations of capacity and reliability dominated.
2. *Software constraints* (mid 1960s to mid 1980s). The phase dominated by the productivity of systems developers and difficulties of delivering systems on time and within budget.
3. *User relations constraints* (mid 1980s to early 1990s). The phase dominated by system quality problems, arising from an inadequate perception of user demands and an inadequate servicing of needs.

Friedman also argues that since the early 1990s commercial computing has been entering a fourth phase dominated by organisational and environmental

constraints. The particular focus of this phase is the search for ways in which IS can improve the position of the organisation in its environment.

## **6.6 Reference**

Friedman A.L. with Cornford D.S. (1989). *Computer Systems Development: history, organisation and implementation*. John Wiley, Chichester.

## **6.7 Exercise**

1. In terms of some organisation known to you find out whether it has an IS services function. In what way is the IS service organised? Utilise the distinctions made in this chapter to help describe the organisation of this function.

# **7 *Information Systems Engineering***

## **7.1 Introduction**

In chapter 1 and chapter 2 we introduced the context for information systems development in terms of preliminary definitions for the terms information and information system. In chapter 3 we discussed some of the consequences of information and information systems for organisations and in chapter 6 we described some of the features of the information service within organisations.

A key conclusion is that the development of information systems based in information technology is now a significant part of the activity undertaken by many organisations. The main objective of this chapter is to describe the key context for, and features of information systems development (ISD), particularly the recent attempts to cast it as an engineering discipline. We return to the issue of whether information systems development is a true engineering discipline in chapter 39.

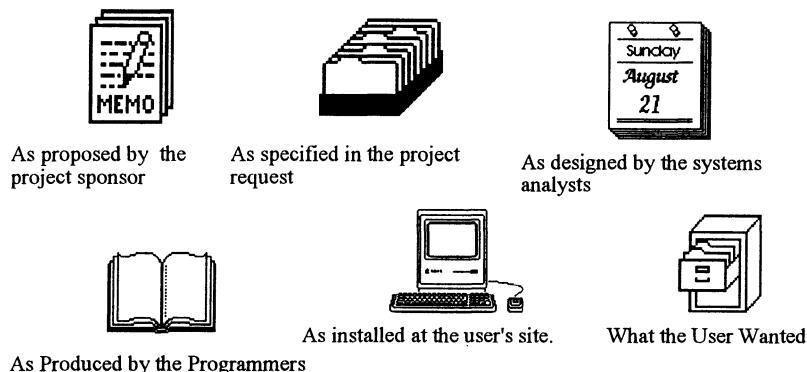
## **7.2 The Software Problem**

Over the last 20 to 30 years the performance of computer hardware has increased by an order of 100. In the same period, software performance has increased only by an order of 10. This is usually described as the software problem, the applications backlog, the software bottleneck or the hardware-software gap (Boehm, 1981).

The software bottleneck is really the high-level representation of a series of smaller problems, such as:

1. Users cannot obtain applications when they want them. There is often a delay of years.
2. It is difficult, if not impossible, to obtain changes to systems in a reasonable amount of time.
3. Systems have errors in them, or often do not work.
4. Systems delivered do not match user requirements.
5. Systems cost much more to develop and maintain than anticipated.

Figure 7.1 illustrates the software problem in more humorous form.



**Figure 7.1: The Software Problem**

### **7.2.1 Traditional vs Contemporary Development**

Another way of describing the essence of the problems and proposed solutions to the information systems development process is in terms of a contrast between what might be called traditional and contemporary approaches to information systems development. The traditional approach to ISD (normally portrayed as a historical phenomenon) is normally described in the following terms:

1. Applications were developed in a piecemeal manner.
2. Applications primarily addressed the operational levels of business.
3. Analysis was driven by the conventional activities of business.
4. Documentation was in terms of flow-charts and narrative.
5. No formal guidance was given on what to do.
6. There was a concentration on description of a solution in computing rather than business terms.

In contrast, contemporary information systems development is normally portrayed in the following terms:

1. Emphasis on integrated applications.
2. Applications at the tactical and strategic levels of business.
3. Emphasis on structure.
4. Emphasis on both data and process concerns.
5. Vast array of graphically based techniques available.
6. Emphasis on a step-by-step approach to development.
7. Emphasis on documenting the problem domain in terms of logical and conceptual models.

In this sense, the traditional approach is normally used as a base line from which to compare other, more systematic approaches to systems development.

### 7.3 Information Systems Development as Engineering

Kuhn (1970) defines a paradigm as being a set of universally recognised scientific achievements that for a time provide model problems and solutions to a community of practitioners. Although Kuhn uses this construct to explain the social nature of revolutions in the mature natural sciences, the concept is useful for our purposes in describing the structure of modern-day information systems development.

In our terms, a paradigm might be defined as a sometimes loosely organised framework of concepts which guides activity. Kuhn says that ‘men whose research is based on shared paradigms are committed to the same rules and standards for scientific practice’. In information systems development we normally mean by a paradigm a suggested approach to building information systems. The approach might incorporate a particular philosophy, use particular tools, techniques and methods and even have a suggested area of application.

One thematic solution much promoted to the software problem has been the attempt to establish information systems development as an engineering discipline. The British Computer Society, for instance, has long seen itself as the professional body for information systems engineering in the UK.

In this section we consider the remit of the information systems engineer. In particular we describe information systems engineering as a discipline which may take its influence from three development paradigms that have been promoted over the last few decades: software engineering, information engineering and knowledge engineering.

#### 7.3.1 Software Engineering

Many solutions to the software problem have been proposed. One of the most influential has been to try to cast software development as an engineering exercise. Boehm (1976) defines software engineering in the following terms:

Software engineering is the practical application of scientific knowledge to the design and construction of computer programs and the associated documentation required to develop, operate and maintain them.

This rather general definition captures the all-encompassing nature of the term software engineering. This definition is however somewhat vague. It contains at least one term *scientific knowledge* that is subject to a number of different interpretations. Hence, a more pragmatic definition of software engineering is given below:

Software engineering is the systematic application of an appropriate set of techniques to the whole process of software development.

This definition concisely states the three most important principles of software engineering:

1. A set of techniques is used to increase quality and productivity.
2. The techniques are applied in a disciplined, not a haphazard way.
3. The techniques are applied to the whole process of software development.

One of the important themes of software engineering is the emphasis on a clear structure for software development. This is usually contrasted with the traditional, *ad hoc* approach to software development (see section 7.2.1).

### **7.3.2 Information Engineering**

Originating in the work of Finklestein and Martin (Martin, 1984), information engineering has been an effective complement to software engineering. Information engineering is defined by Martin in the following terms:

The term software engineering refers to the set of disciplines used for specifying, designing and programming computer software. The term information engineering refers to the set of interrelated disciplines that are needed to build a computerised enterprise based on data systems. The primary focus of information engineering is on the data that are stored and maintained by computers and the information that is distilled from these data.

Martin discusses the way in which information engineering builds itself on a number of premises:

1. That data lie at the centre of modern data processing.
2. That the types of, or structure of, data used in an organisation do not change very much.
3. That given a collection of data, we can find a way to represent it logically.
4. That data are relatively stable, but processes that use this data change rapidly.
5. That because data remain relatively stable, whereas processes are subject to rapid change, data-oriented techniques succeed if correctly applied where process-oriented techniques have failed.

Figure 7.2 illustrates the basic idea underlying these premises. The skyscraper blocks represent individual applications or processes (the life-blood of software engineering) which are subject to rapid change. All such processes are built on the same bedrock, information engineered foundations which, by their very nature, must remain relatively stable.

The term information engineering is also used to indicate a particular commercial information systems development methodology (see chapter 27).

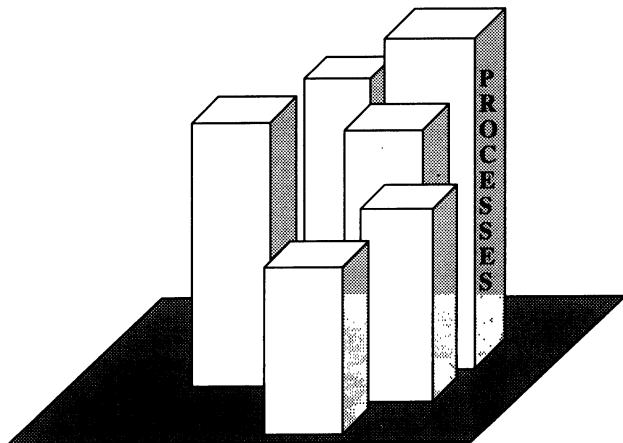


Figure 7.2: Information Engineering

### 7.3.3 Knowledge Engineering

Knowledge engineering is the discipline devoted to the effective construction of knowledgebase systems, i.e. systems that represent knowledge. During the late 1980s a number of people began calling for information systems specialists to be not simply software engineers or information engineers; they also needed to be knowledge engineers. The fundamental problem of information systems development was reconceived as how to represent some subset of organisational knowledge in a computational medium. This is the issue of knowledge representation, a well-discussed topic in Artificial Intelligence. It is for this reason that a chapter on knowledgebase systems is included in the tools section. The information systems developer needs better tools for knowledge representation, and knowledgebase systems are one such tool.

More recently the issue of knowledge workers and knowledge management has become a significant area of concern for some information systems specialists. Knowledge workers are those people within organisations who work with and generate knowledge. Knowledge management is concerned with the problems of representing and controlling organisational knowledge. Both of these areas emphasise the important role of technology in supporting knowledge work.

## 7.4 The Nature of Information Systems Engineering

The chief role of software engineering has been to propose appropriate methods for the development of software. The role of information engineering has been concerned with the problems of managing data and the problems of integrating information systems with business activities. The role of knowledge engineering has been to find ways of capturing and representing organisational knowledge.

However, a number of important areas lie outside the current focus of all three of these paradigms, but which clearly ought to lie within the focus of information systems engineering. For this reason we present in this section a brief initial definition of information systems engineering, together with an overview of the primary components of this discipline. In chapter 38 and 39 we examine this definition in greater detail.

### 7.4.1 *Definition*

Information systems engineering/development is the science and art of designing and making, with economy and elegance, computer-based information systems that support the activity of particular organisations.

Information systems development is clearly a process with a defined output: an information technology system. However, information is a far more complex entity than the material with which other engineering disciplines such as structural engineering have to work. This is because it is impossible to divorce information from the process of human interpretation (chapter 1). Information systems are primarily embedded within human activity systems (chapter 2) and hence issues concerning the design of work are equally as relevant, as are the issues of the design of the technology (chapters 25, 26 and 28).

### 7.4.2 *Components*

If we conceive of information systems engineering as a process then two sets of activities are important to this process. The first set of activities are those internal to any particular development project. In very general terms, these concern the linear activities of systems analysis, systems design, systems construction, systems delivery and systems maintenance:

1. *Systems analysis*. This represents the activity of determining the requirements for some information technology system.
2. *Systems design*. The process of designing how the information technology system will work.
3. *Systems construction*. The process of building an information technology system.

4. *Systems delivery*. The process of implementing a system within its organisational setting.
5. *Systems maintenance*. The process of correcting errors in the IT system and tuning the system over time to meet new organisational requirements.

The second set of activities, what we might call secondary activities, are external to any one IS development project. They normally bridge across the IS development portfolio of a particular organisation.

1. *Business analysis* approaches help in determining the key objectives of some organisation or part of an organisation, and relating IS to these objectives (chapters 25 and 26).
2. *IS planning*. The objectives determined by business analysis should be a necessary input to IS planning. IS planning is the process of defining an information systems architecture for some organisation, and developing two types of plan for the IS services function: strategic and operational (chapter 33).
3. *IS management* is the process of putting plans into practice, of evaluating progress against plans, and of adapting plans in the face of contingencies (chapter 32).
4. *Project planning and management* (chapter 31). These activities move the focus of planning and management activity down to the level of given development projects.

In terms of both the internal and external elements of the ISD process we may distinguish between three layers of what we might call supporting ‘technology’. The term technology is used here in its broadest sense to refer to any form of device that aids the work of some person or group of persons.

1. *Methods*. These constitute frameworks which suggest, sometimes in great detail, the tasks to be undertaken in a given development process. Hence, for instance, SSADM (chapter 27) constitutes a method for the analysis and design of information systems while PRINCE (see chapter 28) comprises a method for project management.
2. *Techniques*. These form the component parts of methods in that they constitute particular ways of undertaking given parts of process. Hence, for example, entity-relationship diagramming (chapter 16) is an established analysis technique while PERT is an established project management technique.
3. *Tools*. By tools we primarily mean here available hardware, software and communication facilities for engaging in some part of the internal project development process or its set of associated external activities. Hence, for instance, Microsoft Access is an established tool for systems construction while Microsoft Project is an established tool to aid project managers.

As well as internal and external activities relevant to the process of information systems development we should also consider a number of much higher level processes occurring at the level of economies and societies which have an effect upon ISD. Within the domain of this work we consider three such environmental processes:

1. *Organisational fit.* Here we need to consider issues relating to ways in which we can assess the utility of particular information systems to particular organisations.
2. *Professionalism.* Issues relating to the development of a profession of information systems engineering.
3. *Theory.* In what ways can we study the process of ISD and how may such study help to improve the process of IS development?

## 7.5 The Life-cycle of Information Systems Development

One important element that serves to define the ISD process is the idea of a life-cycle. This constitutes an ideal description of the way in which development work is organised. There are a number of competing models of the information systems development life-cycle. In this section we first consider a model which had some influence in the 1970s and 1980s and has formed the bedrock for many of the so-called structured methods (chapter 27). We then compare this model with a number of alternatives.

### 7.5.1 The Waterfall Model

The most developed contemporary model for the information systems development life-cycle is undoubtedly based around the ‘waterfall’ approach. This approach portrays the development process as being made up of a series of well-defined stages, with well-defined inputs to each stage, and well-defined outputs from each stage. It is described as a waterfall because activity is meant to flow down in a sequence from initiation of a project to its completion. This approach is illustrated in figure 7.3.

In large organisations, there are usually a large number of requests for new application systems. Such requests may be initiated by a formal IS planning process. More usually they will be produced in an *ad hoc* fashion by functional units within the organisation. To handle such a diversity of applications formally, most such enterprises engage in some form of project selection process. The purpose of such a selection process is to identify the most suitable applications for development in terms of organisational objectives (chapter 34).

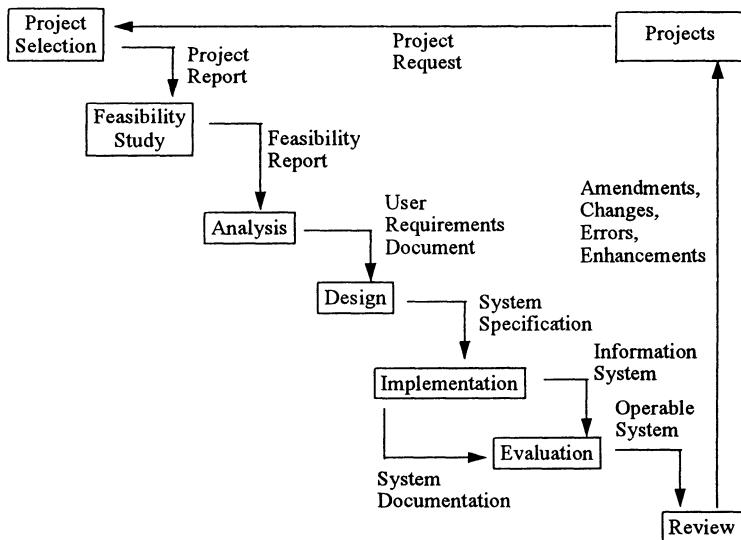


Figure 7.3: The Waterfall Model of Development

The primary mechanism of the project selection process is the project selection committee. This committee does not examine each project in detail. This is the responsibility of the feasibility study. Here, a systems analyst, or team of systems analysts, identifies the initial framework for the application, and investigates whether it is feasible to tackle the project given the available organisational resources. The end-result of the feasibility study is the feasibility report. This report is presented to relevant users who offer their comments and opinions. This user input may then be fed back into the feasibility study that produces a revised report.

In a sense, the project selection process and the feasibility study act as two ‘filters’ at the beginning of the development life-cycle. Project selection is a coarse-grained filter. Its objective is to reject those projects that are patently unsuitable in terms of the mission of an organisation. The feasibility study is a fine-grained filter. Its objective is to reject projects on more detailed grounds.

A typical feasibility report will contain a list of problems in the current environment, a list of requirements for the proposed system, an overview analysis of the major activities and data of the proposed systems, a list of possible technical options for implementation, and a summary of the costs and benefits of the system.

Once the users are satisfied with the feasibility report, it is fed into the analysis phase. The end-result of this phase is the user specification or user requirements document. Elements of this requirements specification are continually reviewed with end-users until they are satisfied that the requirements adequately describe their needs. Most contemporary requirements documents

exploit a series of graphic notations for expressing processing and data details. A ‘tool-kit’ of such techniques is presented in later chapters of this work.

The requirements document is a logical or implementation-independent view of the proposed information system. This must be turned into a physical or implementation-dependent view of the system through the process of design. The end-result of the design phase, the systems specification, is again reviewed with users until all interested parties are happy with progress.

The completed design document is used to direct the production of the application system. Programmers are given specifications for the various modules of the system, and proceed to program them in the language and on the hardware chosen for the implementation. Another important product of the implementation phase is documentation. This must describe the working system both for technical staff and for the end-user.

The system, once written and tested, must be subject to a critical evaluation phase. This primarily means comparing the system produced with elements of the original requirements document to check that the system has achieved its objectives.

Once the system is in operation, it is usually subject to a whole series of user reviews. Such reviews are likely to generate a number of further project requests – amendments to the existing system, suggestions for new systems etc. – which feedback into the project selection process.

### *Criticisms of the Waterfall Model*

Approaches based upon the waterfall model have been criticised on a number of grounds.

1. In a strict waterfall approach it is difficult to move back and change elements of a previous stage.
2. One of the most popular criticisms is that the waterfall approach is too monolithic and consequently cannot cope with rapid organisational change (chapter 29).
3. Typically, users are involved with development only in the early stages of a waterfall-based development life-cycle. This can lead to problems, particularly at the delivery stage where users may resist the implementation of an information system (chapter 28).

#### **7.5.2 Alternative Models of the Development Life-cycle**

The linear or ‘waterfall’ model of information systems development as described above is the one that has achieved a certain pre-eminence in recent times. There are however a number of other contrasting models that have been proposed, particularly based around more iterative notions of the ISD life-cycle.

### Rapid Prototyping

A more incremental method of systems development, usually called prototyping, or sometimes rapid prototyping, has come back into favour. Figure 7.4a illustrates the essential elements of a pure prototyping approach. Here, the emphasis is on rapidly developing elements of a working system and continuously feeding back the results of user evaluation. The prototype that emerges from this exercise is regarded as the working system. This topic is discussed in more detail in chapter 24.

### Structured Prototyping

A more cautious vision of prototyping is illustrated in figure 7.4b. Here prototyping is seen primarily as a requirements elicitation tool. A working version of the technical information system is not expected as the result of the prototyping exercise. Instead, the prototype is documented in a requirements specification before undertaking design.

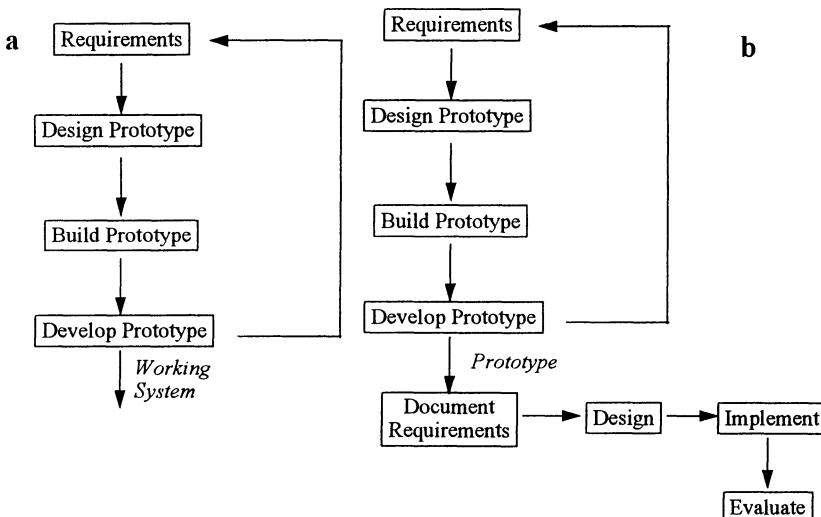
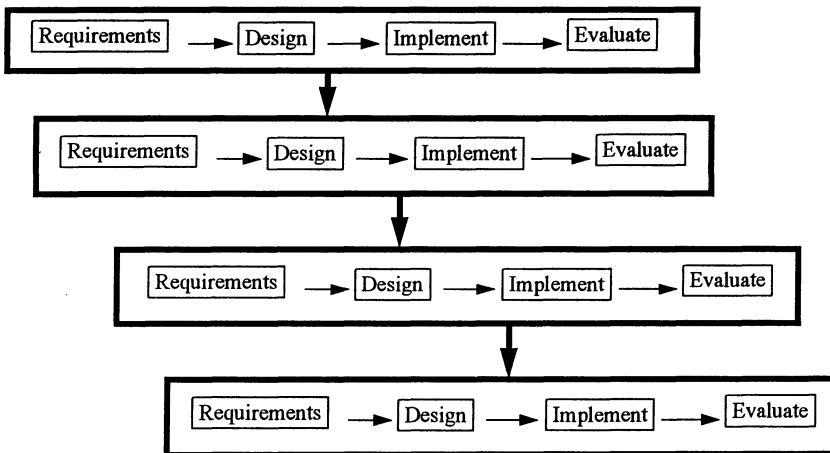


Figure 7.4: Rapid and Structured Prototyping

### ***Evolutionary Development***

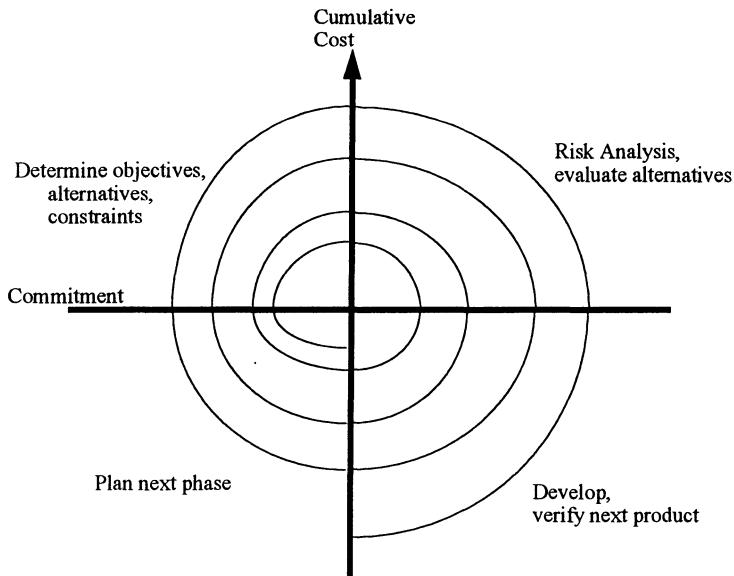
In contrast to the monolithic nature of the waterfall model, this approach calls for a series of development efforts, each of which leads to a delivered product. Each delivered product provides some needed operational capability while also generating further requirements (Crinnion 1991). Figure 7.5 illustrates this idea.



**Figure 7.5: Evolutionary Development**

### ***Spiral Model***

Boehm (1987) has proposed an approach that falls between a strict waterfall life-cycle and those based on iterative models. It is known as the spiralist model and is illustrated in figure 7.6. The radial co-ordinate in figure 7.6 represents the total cost incurred on a project to date. Each loop of the spiral, from the X axis clockwise through 360 degrees represents one phase of development. A phase may be waterfall-based, prototyping-based, or evolutionary-based. The decision as to which model to use, or whether to terminate the project, is made at each crossing of the X axis.



**Figure 7.6: Spiral Model of Development**

## 7.6 Conclusion

Most approaches to information systems development have arisen as attempted solutions to problems with software development. Such approaches tend to locate themselves around what they see to be an engineering approach to the ISD problem. One approach, software engineering, takes a very process-oriented view on software development, traditionally focusing on the issue of programming. Another approach, information engineering, focuses primarily on the issues of data management, and hence not surprisingly has been particularly focused on database technology. Knowledge engineering focuses on knowledge issues and is particularly an attempt to apply an artificial intelligence perspective to systems development. Knowledge-based systems have therefore been a primary area of interest.

The chapter went on to define information systems engineering in terms of a process model consisting of three levels: internal activities, external activities and environmental activities. This layered model serves to broaden the definition of ISE/ISD, particularly to address organisational concerns. Hence a discussion of ISD must focus not only on issues of tools, techniques and methods but also on issues of management and planning.

We concluded the chapter with a discussion of a number of models of the life-cycle of ISD. One model of the life-cycle has traditionally been dominant – the waterfall model – but has undergone some criticism in recent times. For this reason a number of alternative development models have been proposed.

## 7.7 References

- Beynon-Davies P. (1992). *Knowledge Engineering for Information Systems*. McGraw-Hill, London.
- Boehm B.W. (1976). 'Software Engineering'. *IEEE Transactions on Computers*. 25. 1226-1241.
- Boehm B.W. (1981). *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ.
- Boehm B.W. (1987). The Spiral Model of Software Development and Enhancement. In R.H. Thayer (Ed.) *Software Engineering Project Management*. 128-142. IEEE Press, California.
- Crinnion J. (1991). *Evolutionary Systems Development*. Pitman, London.
- Kuhn T.S. (1970). *The Structure of Scientific Revolutions*. (2nd Ed.). University of Chicago Press, London.
- Martin J. (1984). *An Information Systems Manifesto*. Prentice Hall, Englewood Cliffs, NJ.

## 7.8 Exercises

1. In terms of some organisation known to you document what approach to information systems development is taken. What elements of information systems development is covered by engineers in the organisation? For instance, is business analysis conducted, is project management undertaken etc.?
2. Software engineers sometimes define their discipline as being 'programming in the large'. What do you think is meant by this definition, and in what way does it overlap with our definition of information systems engineering supplied in this chapter?

## ***Part Two***

## ***Tools***

### **Tool**

Thing designed to help or enable the hand to apply force

(*Oxford English Dictionary*)

Implement, instrument, apparatus, appliance

(*Roget's Thesaurus*)

This part discusses a common set of tools used to build contemporary IT systems. Traditionally, the most ubiquitous of such tools was the third generation programming language. More recently, the trend has become one of utilising either object-oriented and/or fourth generation languages for the construction of processing logic. Part of this processing logic may be specified using knowledge-based technology. At the back-end of the IT system there will usually be some database system in place under the management of the database management system. At the front-end, multimedia and hypermedia technology will be used to build aspects of the user interface.

# **8 Structured Programming Languages**

## **8.1 Introduction**

Programming languages are used to describe algorithms, i.e. sequences of steps that lead to the solution of problems. Programming languages are broadly classified into two groups: low-level languages and high-level languages. Low-level languages are close to machine languages. They demonstrate a strong correspondence between the operations implemented by the language and the operations implemented by the underlying hardware. High-level languages in contrast are closer to human languages. Each statement in a high-level language will be equivalent to many statements of a low-level language. The key advantage offered by high-level languages is therefore abstraction. As the level of abstraction increases the programmer needs to be less and less concerned about the hardware on which a program runs and more and more concerned with the problems of the application. Hence, the trend has been to build more and more abstraction into programming languages.

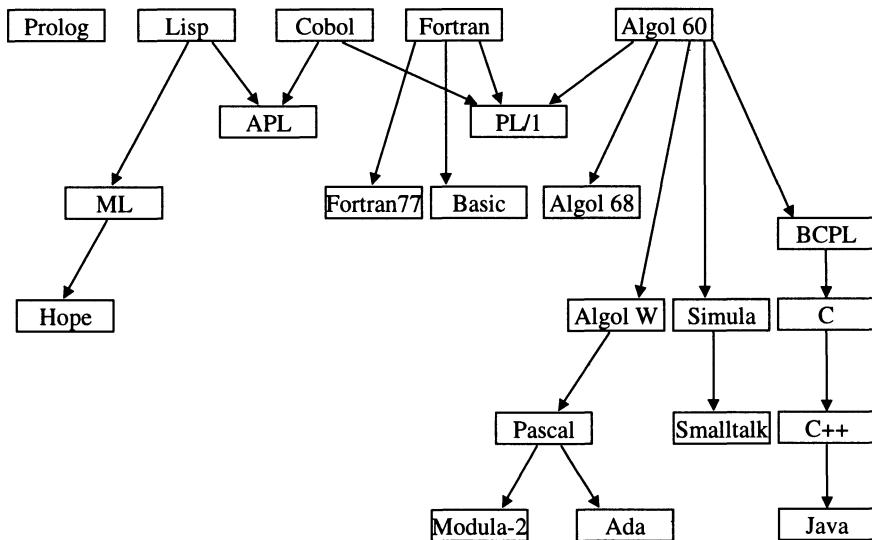
In this chapter we concentrate primarily on high-level languages, illustrating concepts using the language Pascal. In chapter 9 we look at a particular set of languages known as object-oriented languages, and in chapter 11 we look at fourth generation languages.

## **8.2 Generations of Programming Languages**

There is some consensus that there have been at least three generations of programming languages:

1. *Machine code*. This is the earliest form of programming language, only one step removed from the binary code used by the machine to perform instructions.
2. *Assembly language*. This was the first attempt to abstract out detail of the machine and provide the programmer with a more powerful set of symbolic instructions with which to write programs.
3. *High-level languages* (also known as third generation languages). These are meant to be general-purpose programming languages further removed from machine implementation. Such languages can be divided into further groupings such as imperative languages (FORTRAN, COBOL, C), functional languages (LISP), logic programming languages (PROLOG), and object-oriented languages (Smalltalk, C++). Imperative languages are by far the most widely used for information systems development, although object-oriented languages in particular are beginning to have some influence in the

development domain. Figure 8.1 illustrates a family tree of high-level languages.



**Figure 8.1: A Family Tree of High-Level Languages**

Some people talk of fourth generation languages and some even of fifth generation languages. Fourth generation languages and environments are the topic of chapter 8. The term fifth generation language is generally used in the context of knowledge-based systems work (chapter 14).

### 8.3 Structured Languages

In 1965, a Dutch academic named Edsger Dijkstra presented a paper in which he described how programming could be reduced to a few basic rules. One of these rules called for the elimination of GOTOs from programming languages. The other rules detailed how programs could be built from three basic constructs: sequence, selection and iteration. This, in essence, stimulated the structured programming movement.

During the 1970s and early 1980s a series of so-called structured languages emerged. The intention was to provide a more rigorous tool-set with which to build information systems. The earliest of such languages were a group of languages known as the ALGOL family. Kernighan and Plauger (1976) brought out an influential, minimalist language at Bell Laboratories known as C. This is currently one of the most popular of the modern-day programming languages.

## 8.4 The Major Constructs of Structured Programming Languages

In this section we discuss the basic building blocks of most structured programming languages. We illustrate the concepts using a simple dialect of the language Pascal, a language much used in educational circles and a derivative of ALGOL.

Most conventional programming languages are built out of two constructs: statements and control structures.

### 8.4.1 Statements

Statements are the basic instructions of high-level languages. Statements are command lines built from a mixture of keywords, variables and constants. Examples of valid Pascal statements are:

```
balance := balance + credit;  
READ (credit);  
WRITELN ('Balance is', balance);
```

The first statement assigns the summation of the values held in the two placeholders or variables, balance and credit, to the placeholder on the left of the ‘:=’ sign. The second statement reads a value from a file into a variable. The third statement writes a string to the terminal screen.

### 8.4.2 Control Structures

Control structures are used to control the flow of execution of statements. Control structures come in three major forms: sequences; conditions; and loops.

#### 8.4.2.1 Sequences

Sequences are logical sets of statements. In Pascal it is usual to encase sequences in the keywords BEGIN and END, called a block, e.g.

```
BEGIN  
    balance := balance + credit;  
    WRITELN ('Balance is', balance);  
END;  
  
BEGIN  
    balance := balance - debit;  
    WRITELN ('Balance is', balance);  
END;
```

Note how each statement is terminated with a semicolon. Keywords are written here in uppercase merely to distinguish them from application-specific words, such as variable names.

#### *8.4.2.2 Conditions*

Conditions allow selection among alternatives. There are three forms of conditions:

##### 1. Single-branched

```
IF credit > 0 THEN balance := balance + credit;
```

##### 2. Double-branched

```
IF transactionType = 'C' THEN
    balance := balance + credit
ELSE
    balance := balance - debit;
```

##### 3. Multiple-branched

```
CASE transactionType OF
    'C': balance := balance + credit;
    'D': balance := balance - debit;
    'P': WRITELN ('Balance is', balance);
END;
```

The multiple-branched condition can be emulated by a series of deeply nested IF THEN ELSE statements. The CASE construct is however generally held to be more readable in these situations.

#### *8.4.2.3 Loops*

Loops are the mechanisms of iteration. Iterative statements come in three forms:

##### 1. *Countable loop*. This form of loop is performed a specified number of times.

```
FOR count := 1 to 10 DO
BEGIN
    READ (credit);
    balance := balance + credit;
    count := count + 1
END;
```

2. *While loop.* A test is performed at the start of each iteration. If the condition specified is true the loop continues. If the condition is false, the loop terminates.

```
count := 1
WHILE count < 11 DO
  BEGIN
    READ (credit);
    balance := balance + credit;
    count := count + 1
  END;
```

3. *Repeat loop.* This works in a similar manner to the while loop, except that the test is performed at the end of each iteration rather than at the start.

```
count := 1
REPEAT
  READ (credit);
  balance := balance + credit;
  count := count + 1
UNTIL count = 10;
```

#### 8.4.2.4 Other Component Parts

There are many other component parts to modern programming languages. We discuss three important parts below:

1. *Declarations.* Pascal programs are made up of two parts: a heading and a body. The heading provides the program name and lists the program's interfaces with its environment. The body consists of a declaration part and a statement part. All variables used in a Pascal program must be declared to be of a given data type, e.g.

```
PROGRAM BankAccount (input, output);

VAR
balance, credit, debit: REAL;
age: INTEGER;
transactionType: CHAR;
```

The parameters of the program declared between parentheses indicate that input should be taken from the keyboard and output should be made to the screen. Variables can also be clustered in structures such as records, e.g.

```
TYPE
  Account = RECORD
    AccountId: INTEGER;
    Balance: REAL;
  end;
```

2. *Input/output.* The default input/output is reading via the keyboard and writing to the screen. To read from an external file, the filename has to be specified in the parameters of the program. The simple program below reads customer identifiers from a text file named customers and displays them on the screen.

```
PROGRAM CustomersDisplay (Customers, OUTPUT);
```

```
VAR
  Customers: TEXT;
  CID: PACKED ARRAY [1..10] OF CHAR;
  Count: INTEGER;
```

```
BEGIN
```

```
  RESET (Customers);
```

```
  FOR count := 1 to 100 DO
    READLN (Customers, CID);
    WRITELN ('Customer:' :CID)
  END;
```

```
END.
```

The RESET command initialises the file pointer to the first line of the Customers file. READLN reads a customer identifier into a string of 10 characters long, and moves to the next line in the file.

3. *Procedures.* As we shall see in chapter 21, one of the main tenets of structured development is that any system should be built out of a series of highly cohesive but loosely coupled modules. Modules are declared in Pascal as procedures. A procedure is made up of three parts: a procedure heading which specifies a name for the procedure and parameters to be passed to the procedure; a series of local variable declarations; and a procedure body. The simple program below contains an embedded procedure. It prints out a percentage of the amount owed on a chargecard.

```
PROGRAM ChargeCard (OUTPUT);
```

```
PROCEDURE PrintOwed(Balance, Percentage:INTEGER);
```

```
VAR  
    Payment: INTEGER;  
BEGIN  
    Payment := Balance * Percentage DIV 100;  
    WRITELN ('Pay £', Payment)  
END;  
  
END.
```

## 8.5 Conclusion

The high-level programming language is still the medium in which most commercial information systems are constructed. COBOL, originally an unstructured language, still maintains its place as a language for commercial development. However, the lessons of good structure have influenced much of the style of modern-day COBOL programming.

Having said this, much modern development occurs in structured languages. By far the most prevalent is the language C. Because of its portability, many other tools such as DBMS and 4GLs are written in this language. In the next chapter we consider a number of concepts embodied in a popular superset of C known as C++.

## 8.6 Reference

Kernighan B.W. and Plauger P.J. (1976). *Software Tools*. Addison-Wesley, Reading, Mass.

## 8.7 Exercises

1. What does describing a language as imperative mean?
2. The GOTO statement is sometimes referred to as an unconditional jump. What do you think is meant by this description?
3. Discuss some of the reasons why the GOTO statement is generally considered a dangerous programming construct.
4. Why is it useful to have both a 'do while' and 'repeat until' statement in modern programming languages?
5. Discuss some of the advantages of being able to build systems out of a series of related program modules.
6. Many unstructured variants of COBOL are still very much used for systems development. Why do you think this is?

# **9 Object-Oriented Programming Languages**

## **9.1 Introduction**

The fundamental difference between conventional and object-oriented programming (OOP) relates to the way each approach treats data and process. In conventional programming (chapter 8) data and process are separate things. To create an information system we define our data structures and then we define routines to operate upon them.

In OOP, process and data are intertwined. When we build an information system using OOP we define both data and processes to be the characteristic of objects. In this approach, information systems are seen as being composed of a large set of interacting objects.

This chapter presents an overview of OOP. We first discuss some of the major concepts of OOP. Then we discuss the development of object-oriented programming languages, and we illustrate OOP with an extension to Pascal. We conclude with a brief discussion of the modern phenomenon known as Java.

## **9.2 Objects, Methods, Messages, and Encapsulation**

An object is some entity that contains data and the associated set of routines that operate on data. The data associated with an object are said to be the attributes or instance variables of an object. The routines associated with an object are referred to as an object's methods. To make an object perform one of its methods we have to send an object a message. OOP is fundamentally message passing.

For example, we might create an object that represents a rectangle. Its attributes constitute the four corners of the rectangle. Its methods constitute routines for drawing, erasing and moving rectangles. To actually draw a rectangle, you send the object a draw message. This binding together of attributes and methods with a common interface is referred to as encapsulation.

Contrast this approach with the way we would handle rectangles using a conventional programming approach. First we would probably define a data structure that represents the four corners of the rectangle. Then we would build routines to draw, to erase and to move a rectangle. Each of these routines will take a rectangle data structure as an argument.

### 9.3 Classes

Every object belongs to an object class. The class defines the implementation of a particular kind of object. We normally say that every object is a member or instance of a class. Hence, every instance of a rectangle is a member of the class Rectangle. We might define the class Rectangle as below:

<b>Class</b>	Rectangle
<b>Instance Variables</b>	top, left, bottom, right
<b>Methods</b>	draw line from point to point erase lines offset points
<b>Messages</b>	Draw, Erase, Move

### 9.4 Inheritance and Polymorphism

We can define a class to be a specialisation of an existing class. The new class is called the subclass, and the existing class is called the superclass. A class without a superclass is said to be the root class of the hierarchy.

A class hierarchy is used to support the concept of inheritance. A subclass is said to inherit all the instance variables and methods of its superclass. Subclasses can also define additional instance variables and methods or override the methods defined in the superclass. Overriding a method means that the subclass responds to the same message as the superclass, but uses its own routine to respond to the message.

For example, suppose we create a class to represent employees of a company. We define the class as below:

<b>Class</b>	Employee
<b>Superclass</b>	none
<b>Instance Variables</b>	name, dateOfBirth
<b>Methods</b>	return name return (today – dateOfBirth) return 0
<b>Messages</b>	GetName, GetAge, GetPay

We can now define two subclasses: HourlyEmployee and WeeklyEmployee. Both subclasses inherit all the instance variables and methods of the Employee class. Each subclass however defines a new instance variable to store the appropriate salary, and each subclass overrides the GetPay method to return the appropriate weekly salary. The definitions are given below:

<b>Class</b>	HourlyEmployee
<b>Superclass</b>	Employee
<b>Instance Variables</b>	hourlySalary
<b>Methods</b>	return (hourlySalary * 40)
<b>Messages</b>	GetPay

<b>Class</b>	WeeklyEmployee
<b>Superclass</b>	Employee
<b>Instance Variables</b>	yearlySalary
<b>Methods</b>	return (yearlySalary / 52)
<b>Messages</b>	GetPay

Sending the same message, GetPay, to the two classes HourlyEmployee and WeeklyEmployee will therefore get the same response, but will have been implemented in different ways. This ability to send the same message to objects of different classes is known as polymorphism.

## 9.5 Object-oriented Programming Languages

A vast range of contemporary programming languages claim to be object-oriented. Not all such languages however support the concepts described in sections 9.1 to 9.4. Wegner makes a useful distinction between what he calls object-based, class-based and true object-oriented languages (Wegner, 1991). Object-based languages are those languages that support the concept of an object in the sense that data and processes are encapsulated together, but do not support the concept of classes or inheritance. An example of a class-based language is Ada. Class-based languages offer support for objects and classes. An example is the language CLU. True object-oriented languages offer support for objects, classes, inheritance and the related concepts of encapsulation and polymorphism. The foremost example is the language SmallTalk.

## 9.6 Object Pascal

Many examples of conventional programming languages now offer object-oriented extensions. The advantage of this approach is that the programmer can gradually migrate from conventional to object-oriented programming. One of the most popular of such languages is C++, an object-oriented superset of C (Jordan, 1990). In the last couple of years much interest has been shown in the programming language known as Java. Java is a variant of C++, which runs on a virtual machine. This means that it can run theoretically on any machine and can be transmitted using the common communication protocol TCP/IP – Transmission Control Protocol/Internet Protocol. This makes it especially suitable for writing applications that run over the internet (chapter 13). Many

application developers are consequently re-writing their software or parts of their software in Java to take advantage of the portability that this affords.

To maintain consistency, in this section we consider how objects might be implemented in extensions to the language Pascal (chapter 8). We will assume that a class declaration is similar to a record declaration. We give the class a name, specify its superclass, and declare its instance variables and methods as below:

```
TYPE
  <className> = OBJECT (<superclassName>)
    <instanceVariableDeclaration>;
    <methodDeclaration>;
END;
```

Suppose, for instance, we wish to create an object representing companies with which our business corresponds. We might declare it as below:

```
TYPE
  Company = OBJECT
    name: STRING;
    address: STRING;
    VATNo: INTEGER;
    CONSTRUCTOR Init(initName, initAddress: STRING,
      initVat: INTEGER);
    FUNCTION GetName: STRING;
    PROCEDURE PrintLabel;
  END;
```

Note that this represents the root class of our hierarchy, since no superclass is specified. Note also that methods are defined as standard Pascal procedures and functions. Each procedure or function must then be prefixed with the class name to which it applies. For example:

```
PROCEDURE Company.PrintLabel;
BEGIN
  WRITELN (name);
  WRITELN (address)
END;
FUNCTION Company.GetName: STRING;
BEGIN
  GetName := name
END;
```

Two special types of procedure are also required: constructor and destructor. Constructor creates new instances of objects, destructor removes instances of objects. For example:

```
CONSTRUCTOR Company.Init(initName, initAddress: STRING);
BEGIN
    name := initName;
    address := initAddress;
    vatNo := initVat
END;
```

Two subclasses of the class Company can now be declared:

```
TYPE
    Supplier = OBJECT(Company)
        reliabilityRating: INTEGER;
        PROCEDURE MakeSupply(RM: RawMaterial)
    END;

    TYPE
        Customer = OBJECT(Company)
            creditRating: INTEGER;
            PROCEDURE MakeOrder(P: Product)
        END;
```

Supplier and Customer will now inherit the instance variables and methods of the superclass Company. Each subclass also has its own instance variable and method defined for it. MakeSupply is specific to Supplier, MakeOrder is specific to Customer. If a subclass wanted to override any of the methods of a superclass then we would have to include the keyword 'override' as an additional clause in a method declaration. Suppose, for instance, we wished to produce a special form of printed label for customers. We would then override the PrintLabel method as below:

```
TYPE
    Customer = OBJECT(Company)
        creditRating: INTEGER;
        PROCEDURE MakeOrder(P: Product);
        PROCEDURE PrintLabel OVERRIDE
    END;
```

Having declared the class hierarchy, processing involves sending messages (calls) to the methods of classes. Suppose, for instance, we wish to process a new customer. We might do this as follows:

```
VAR smiths: Customer;  
  
smiths.Init('Smiths and Sons Ltd.', 'The Grove, Pontypridd', 1234);  
  
smiths.PrintLabel;
```

## 9.7 Advantages of Object-oriented Programming

A number of advantages are claimed for OOP:

1. It is claimed that encapsulation aids maintenance in the sense that changing the implementation of a given object is relatively straightforward.
2. The emphasis of OO is on software reuse. The idea is to allow the system builder to create a framework for the new system out of readily available classes.
3. OO approaches have been proposed as a unifying paradigm for systems analysis, design and implementation. As we shall see in chapter 22, OO concepts such as encapsulation, inheritance, etc. are equally relevant to the analysis and design of systems as they are to programming.

## 9.8 Disadvantages of Object-oriented Programming

1. Reuse is only feasible if the programmer knows of the existence of existing classes. The overhead of searching and overriding existing classes may be as great as that involved in writing new code.
2. Although a number of proposals are being made, no consensus has been reached yet on appropriate ways of building OO systems.
3. OO seems particularly suited for problems that involve some aspect of real-world simulation. The case for exploiting pure OO in the area of traditional ISD still has to be made.

## 9.9 Conclusion

Object-oriented concepts are having an influence not only on programming languages but also on other varieties of software such as operating systems and DBMS. In chapter 14 we look at some related ideas in the area of knowledge based systems. The idea of producing systems from hierarchically organised collections of classes has even influenced the areas of systems analysis and design. We discuss these issues in chapters 22 and 30.

### 9.10 References

- Jordan D. (1990). 'Implementation Benefits of C++ Language Mechanisms'.  
*CACM*. 33(9). 61-64
- Wegner P. (1991). 'Learning the Language'. *Byte*. March. 245-253.

### 9.11 Exercises

1. Identify some of the possible objects in the simple claims insurance system described in section 2.3.
2. Identify candidate methods for these objects and potential messages that we might send to objects.
3. Define at least one of these objects using the syntax of Object Pascal.
4. How much hype do you think there is presently in object-orientation?
5. Discuss the corresponding need for an object-oriented approach to analysis and design.
6. A class browser, a tool for locating appropriate object classes, is a much-discussed tool amongst the object-oriented programming fraternity. List some other tools you feel might be important for the object-oriented programmer.
7. The database fraternity (chapter 10) have spent much effort separating programs from data. Object-oriented programming reintegrates these two components. Is this a step back, or a step forward?
8. Discuss the claim that object-oriented programming is in some way more 'natural' than conventional structured programming.
9. How might structured and object-oriented programming be integrated? Is there any benefit in such integration?
10. What advantages arise from polymorphism?
11. Identify some possible objects for the Goronwy Galvanising system.

# **10 Database Systems**

## **10.1 Introduction**

When organisations first began to use the computer they naturally adopted a piecemeal approach to information systems development. One manual system at a time was analysed, redesigned and transferred onto the computer with little thought to its position within the organisation as a whole. This piecemeal approach was necessitated by the difficulties experienced in using a new and more powerful organisational tool.

This piecemeal approach by definition produces a number of separate information systems, each with its own program suite, its own files, and its own inputs and outputs. As a result of this:

1. The systems, being self-contained, do not represent the way in which the organisation works, i.e. as a complex set of interacting and interdependent systems.
2. Systems built in this manner often communicate outside the computer. Reports are produced by one system, which then have to be transcribed into a form suitable for another system. This proliferates inputs and outputs, and creates delays.
3. Information obtained from a series of separate files is less valuable to the organisation because it does not provide the complete picture. For example, a sales manager reviewing outstanding orders may not get all the information he needs from the sales system. He may need to collate information about stocks from another file used by the company's stock control system.
4. Data may be duplicated in the numerous files used by different information systems in the organisation. Hence, personnel may maintain data similar to that held by payroll. This creates unnecessary maintenance overheads and increases the risk of inconsistency.

## **10.2 Databases and Database Management Systems**

Because of the many problems inherent in the piecemeal approach, it is nowadays considered desirable to maintain a single centralised pool of organisational data, rather than a series of separate files. Such a pool of data is known as a database.

It is also considered desirable to integrate the systems that use this data around a piece of software that manages all interactions with the database. Such a piece of software is known as a database management system or DBMS.

### **10.3 What is a Database?**

A database in manual terms is analogous to a filing cabinet, or more accurately to a series of filing cabinets. A database is an organised repository for data. The overall purpose of such a system is to maintain data for some set of enterprise objectives. Normally, such objectives fall within the domain of business administration. Most database systems are built to retain the data required for the running of the day-to-day activities of a business.

Organisation usually implies some logical division, usually hierarchical. Hence we speak of a database as being a collection of files. For instance, a collection of files containing information on jobs at Goronwy would normally constitute a database. Each file in a database is in turn also a structured collection of data. In manual terms, such files would be folders hung in a filing cabinet. Each file in the cabinet consists of a series of records. These might be cards of information, for example, on each job due to be processed. Each record is divided up into a series of areas known as fields such as jobNo, batchWeight, etc. Within each field a specific value is written.

### **10.4 Properties of a Database**

The question of organisation is therefore of fundamental importance to a database system. In database terms organisation further implies a series of properties: data sharing, data integration, data integrity, data security, data abstraction, data independence.

#### ***10.4.1 Data Sharing***

Database systems were originally developed for use within multi-user environments. In such environments, data held in a database is not usually there solely for the use of one person. A database is normally accessible by more than one person perhaps at the same time. Hence a personnel database might be accessible by members of not only the personnel department but also the payroll department.

#### ***10.4.2 Data Integration***

Shared data brings numerous advantages to the organisation. Such advantages, however, only result if the database is treated responsibly. One major responsibility of database usage is to ensure that the data is integrated. This implies that a database should be a collection of data that has no unnecessarily duplicated or redundant data. In the past, separate personnel and payroll systems maintained similar data on company employees such as names, addresses, dates of birth, etc.

The aim of a database system is to store one logical item of data in one place only.

#### ***10.4.3 Data Integrity***

Another responsibility arising as a consequence of shared data is that a database should display integrity. In other words, that the database accurately reflects the universe of discourse that it is attempting to model. This means that if relationships exist in the real world between objects represented by data in our database then changes made to one partner in such a relationship should be accurately reflected in changes made to other partners in that relationship. Hence, if changes are made to the information stored on a particular company department, for instance, then relevant changes should be made to the information stored on employees of that department.

#### ***10.4.4 Data Security***

One of the major ways of ensuring the integrity of a database is by restricting access; in other words, securing the database. The main way this is done in contemporary database systems is by defining in some detail a set of authorised users of the whole, or more usually parts of the database. A secure system would be one where the payroll department has access to information for the production of salaries but is prohibited from changing the pay points of company employees. This activity is the sole responsibility of the personnel department.

#### ***10.4.5 Data Abstraction***

A database can be viewed as a model of reality. The information stored in a database is usually an attempt to represent the properties of some objects in the real world. Hence, for instance, a personnel database is meant to record relevant details of people. We say relevant, because no database can store all the properties of real-world objects. A database is therefore an abstraction of the real world.

#### ***10.4.6 Data Independence***

One immediate consequence of abstraction is the idea of buffering data from the processes that use such data. The ideal is to achieve a situation where data organisation is transparent to the users or application programs that feed off data. If, for instance, a change is made to some part of the underlying database no application programs using affected data should need to be changed. Also, if

a change is made to some part of an application system then this should not affect the structure of the underlying data used by the application.

### **10.5 What is a Database Management System?**

A database management system (DBMS) is an organised set of facilities for accessing and maintaining one or more databases. A DBMS is a shell which surrounds a database and through which all interactions take place. The interactions catered for by most existing DBMS fall into 4 main groups (Sikora, 1997):

1. *Structural maintenance.* Adding new data structures to the database; removing data structures from the database; modifying the format of existing data structures;
2. *Transaction processing.* Managing the following logical units of work: inserting new data into existing data structures; updating data in existing data structures; deleting data from existing data structures.
3. *Information retrieval.* Extracting data from existing data structures for use by end-users; extracting data for use by application programs.
4. *Database administration.* Creating and monitoring users of the database; restricting access to files in the database; monitoring the performance of databases.

### **10.6 Data Models**

Any database adheres to a particular data model. A data model is an architecture for data. It describes the general structure of how data is organised. A data model is generally held to be made up of three components (Tsitchizris and Lochovsky, 1982):

1. A set of data structures.
2. A set of data operators.
3. A set of inherent integrity rules.

These three components are frequently referred to as data definition, data manipulation and data integrity respectively.

### **10.7 The Relational Data Model**

In this section we provide a brief overview of the relational data model, one of the most popular of the contemporary data models. For a more detailed discussion of this topic the reader is referred to Beynon-Davies (1996).

We begin our discussion with a definition of the component parts of the relational data model: data definition, data manipulation, and data integrity. We then give a brief overview of what is the standard core of any relational DBMS – the database sub-language SQL.

### 10.7.1 Data Definition

A database is effectively a set of data structures for organising and storing data. In any data model we must have a set of principles for exploiting such data structures for business applications. Data definition is the process of exploiting the inherent data structures of a data model for a particular business application. The relational data model has only one data structure – the disciplined table or relation.

#### Policies

Policy No.	Holder No.	Start Date	Renewal Date	Premium	Policy Type
5432	2001	01/02/1985	01/02/1998	45.35	Standard Life
5242	2001	01/03/1980	01/03/1998	300.26	Standard Buildings
5348	2001	01/03/1980	01/03/1998	98.78	Standard Contents
4289	2189	01/01/1976	01/01/1998	20.20	Standard Life
4346	2189	01/01/1976	01/01/1998	156.56	Standard Buildings
4389	1136	31/02/1970	31/02/1998	11.63	Standard Life

#### Holders

Holder No.	Holder Name	Holder Address	Holder Tel. No.
2001	J.F.Davies	2, The Drive, Gabalfa	(01222) 289436
2189	T.Evans	15, Orchard Ave, Penylan	(01222) 543483
1136	L.Jones	1, The Parade, Canton	(01222) 389389

The two tables above, policies and holders, are relations because they adhere to a certain restricted set of rules:

1. Every relation in a database must have a distinct name.
2. Every column in a relation must have a distinct name within the relation.
3. All entries in a column must be of the same kind.
4. The ordering of columns in a relation is not significant.
5. Each row in a relation must be distinct. In other words, duplicate rows are not allowed in a relation.

6. The ordering of rows is not significant. There should be no implied order in the storage of rows in a relation.
7. Each cell or column/row intersection in a relation should contain only an atomic value. Multi-values are not allowed in a relation.

To enforce the property that duplicate rows are forbidden each relation must have a so-called primary key. A primary key is one or more columns of a table whose values are used to uniquely identify each of the rows in a table. PolicyNo and holderNo are the primary keys of the tables above. A primary key will be chosen from the set of candidate keys. A candidate key is any column or group of columns that could act in the capacity of a primary key.

The primary unit of data in the relational data model is the data item, for example, a part number, a policy number or a person's date of birth. Such data items are said to be non-decomposable or atomic. A set of such data items of the same type is said to be a domain. For example, the domain of policy numbers is the set of all possible policy numbers. Domains are therefore pools of values from which actual values appearing in the columns of a table are drawn.

Foreign keys are the 'glue' of relational systems. They are the means of interconnecting the information stored in a series of disparate tables. A foreign key is a column or group of columns of some table that draws its values from the same domain as the primary key of some other table in the database. In our insurance database example holderNo is a foreign key in the policies table. This column draws its values from the same domain as the holderNo column – the primary key – of the holders table. This means that when we know the holderNo of some policy we can cross-refer to the holders table to see, for instance, where that customer lives.

#### ***10.7.2 Data Manipulation***

Data manipulation has four aspects:

1. How we input data into a relation.
2. How we remove data from a relation.
3. How we amend data in a relation.
4. How we retrieve data from a relation.

When Codd first proposed the relational data model by far the most attention was devoted to the final aspect of data manipulation – information retrieval. In his early papers, Codd proposed a collection of operators for manipulating relations (Codd, 1970). He called the entire collection of such operators the relational algebra.

The relational algebra is a set of some eight operators. Each operator takes one or more relations as input and produces one relation as output. The three main operators of the algebra are select, project and join. Using these three

operators most of the manipulation required of relational systems can be accomplished. The additional operators – product, union, intersection, difference and division – are modelled on the traditional operators of set theory.

1. *Select/Restrict.* The select or restrict operator of the relational algebra takes a single relation as input and produces a single relation as output. Select is a ‘horizontal slicer’. It extracts rows from the input relation matching a given condition and passes them to the output relation.
2. *Project.* The project operator takes a single relation as input and produces a single relation as output. Project is a ‘vertical slicer’.
3. *Join.* The join operator takes two relations as input and produces one relation as output. A number of distinct types of join have been identified. Probably the most commonly used is the natural join. The natural join operator combines two tables together but only for records matching a given condition. It also projects out one of the join columns.

#### 10.7.3 Data Integrity

When we say that a person has integrity, we normally mean we can trust what that person says. We assume, for instance, a close correspondence between what that person says and what he or she does.

When we say a database has integrity we mean much the same thing. We have some trust in what the database tells us. There is a close correspondence between the facts stored in the database and the real world that it models. Hence, in terms of our insurance database we believe that the fact, *policy 5432 is a life insurance policy*, is an accurate reflection of the workings of our enterprise.

Two types of integrity are important to the relational model: entity and referential integrity.

Entity integrity concerns primary keys. Entity integrity is an integrity rule which states that every table must have a primary key and that the column or columns chosen to be the primary key should be unique and not null.

Referential integrity concerns foreign keys. The referential integrity rule states that any foreign key value can be in one of two states. The usual state of affairs is that the foreign key value refers to a primary key value of some table in the database. Occasionally, and this will depend on the rules of the organisation, a foreign key value can be null. In this case we are explicitly saying that either there is no relationship between the objects represented in the database or that this relationship is unknown.

### 10.8 The Database Sub-language SQL

One of the major formalisms, which define the present generation of relational

database management systems, is Structured Query Language or SQL for short. SQL was originally designed as a query language based on the relational algebra. SQL however is a lot more than simply a query language, it is a database sub-language. This database sub-language is becoming the standard interface to relational and non-relational DBMS.

SQL comes in 3 major parts:

1. A data definition language (DDL) with integrity enhancement.
2. A data manipulation language (DML).
3. A data control language (DCL).

#### ***10.8.1 Data Definition Language***

Suppose that we have a company requirement to produce a database of information, like the tables given above. The structure for each of the tables in the database can be set up using the create table command. For example:

```
CREATE TABLE holders  
(holderNo CHAR(4),  
holderName CHAR(20),  
address CHAR(30),  
telNo CHAR(12))
```

The create table statement allows us to specify a name for a table, and the names, data-types and lengths of each of the attributes in the table. We may also add a number of clauses to a create table statement to specify appropriate primary and foreign keys for each table.

```
CREATE TABLE policies  
(policyNo CHAR(4),  
holderNo CHAR(4),  
startDate DATE,  
renewalDate DATE, policyType CHAR(20))  
PRIMARY KEY (policyNo)  
FOREIGN KEY (holderNo) REFERENCES holders)
```

When a table is created information is written to a number of system tables. This is a meta-database that stores information about the structure of tables at the base level. Information about a table can be removed from the system tables by using the DROP command:

```
DROP TABLE policies
```

Only a certain amount of amendment activity is allowed on table structures in terms of the SQL standard. We can add an extra attribute to a table:

```
ALTER TABLE holders  
ADD (dateOfBirth, DATE)
```

We may also modify the size of an existing attribute:

```
ALTER TABLE holders  
MODIFY (address CHAR(40))
```

#### **10.8.2 Data Manipulation Language**

Having created a structure for the tables in our database, we may enter data into such tables using the INSERT command:

```
INSERT INTO holders (holderNo, holderName, address, telNo)  
VALUES  
(‘1111’, ‘P.Beynon-Davies’, ’12, Redrose Hill, Splott’, ‘(0222) 434567’)
```

If the list of values is in the same sequence as the sequence of attributes in the table, the sequence of attribute names can be omitted:

```
INSERT INTO holders  
VALUES  
(‘1111’, ‘P.Beynon-Davies’, ’12, Redrose Hill, Splott’, ‘(0222) 434567’)
```

We also maintain the ongoing data in the database through use of the update and delete commands:

```
UPDATE policies  
SET policyType = ‘SLife’  
WHERE policyType = ‘Standard Life’  
  
DELETE FROM policies  
WHERE holderNo = ‘5432’
```

Although SQL has a data definition and file maintenance subset, the language was designed primarily as a means for extracting data from a database. Such extraction is accomplished through use of the select command: a combination of the restrict, project, and join operators of the relational algebra. Simple retrieval is accomplished by a combination of the select, from and where clauses:

```
SELECT policyNo, holderNo
FROM policies
WHERE policyType = 'Standard Life'
```

The list of attribute names can be substituted with the wildcard character ‘\*’, in which case all the attributes in the table are listed:

```
SELECT *
FROM policies
WHERE policyType = 'Standard Life'
```

To produce a sorted list as output we add the `order by` clause to the select statement:

```
SELECT policyNo, holderNo
FROM policies
WHERE premium > 75.00
ORDER BY premium
```

To undertake aggregate work, such as computing the average premium of each insurance policy, we use the `GROUP BY` clause:

```
SELECT AVG(premium)
FROM policies
WHERE startDate > '01-JAN-80'
GROUP BY premium
```

SQL performs relational joins by indicating common attributes in the where clause of a select statement. For instance, the select statement below extracts data from the policies and holders tables of relevance to life insurance salesmen and orders it by the `startDate`.

```
SELECT policies.holderNo, holderName, policyNo, startDate
FROM policies, holders
WHERE policies.holderNo = holders.holderNo
AND policyType = 'Standard Life'
ORDER BY startDate
```

#### ***10.8.3 Data Control Language***

The primary mechanism for enforcing control issues in SQL is through the concept of a view. Views are virtual tables which act as ‘windows’ on the database of real tables. The view below establishes a virtual table using the query

above. Salesmen granted access only to this view would be unable to see information of relevance to other insurance areas.

```
CREATE VIEW life
AS SELECT policies.holderNo, holderName, policyNo, startDate
FROM policies, holders
WHERE policies.holderNo = holders.holderNo
AND policyType = 'Standard Life'
ORDER BY startDate
```

The view becomes a table definition in the system catalog and remains unaffected by updating of the underlying policies and holders table.

Access can be restricted on tables and views to particular users via the grant and revoke facilities of SQL. Grant allows users read and file maintenance privileges on tables or views. Revoke takes such privileges away.

```
GRANT INSERT,UPDATE
ON life TO pbd
```

```
REVOKE SELECT,INSERT
ON life FROM pbd
```

## 10.9 Conclusion

Databases and DBMS are important for modern information systems development because they encourage the development of an integrated policy for organisational data. Such a policy more clearly reflects, or models, the organisation as being a set of interdependent subsystems.

Also because of its logical simplicity, the relational database approach has stimulated a whole range of database design techniques. Perhaps the two most prominent are Codd's notion of normalisation (see chapter 15), and Chen's notion of entity-relationship diagramming (see chapter 16).

## 10.10 References

- Beynon-Davies P. (1996). *Database Systems*. Macmillan Press, Hounds mills, Basingstoke.
- Codd E.F. (1970). 'A Relational Model for Large Shared Data Banks'. *CACM*. 13 (1). 377-387.
- Sikora Z.M. (1997). *Oracle Database Principles*. Macmillan Press, Hounds mills, Basingstoke.
- Tsitchizris D. and Lochovsky F. (1982). *Data Models*. Prentice Hall, Englewood Cliffs, NJ.

**10.11 Exercises**

1. Define a table to hold insurance brokers' information using the create table command of SQL. The table should contain a broker number, name, address and telephone number.
2. To link brokers to policies we need an extra field in the policies table. What is this field and how do we conduct this modification using SQL?
3. Assuming answers to questions 1 and 2 write an insert statement in SQL to put some data into the brokers' table. Also write an update statement to add appropriate information to the policies table.
4. Write a select statement that will list all the policies relevant to the broker you have created.
5. Package the select statement as a view.
6. Grant retrieval access only on this view to a user named Jones.
7. Start considering the data that we would need to hold for Goronwy Galvanising.
8. What reports would be needed by the production controller at Goronwy?

# **11 Fourth Generation Environments**

## **11.1 Introduction**

As with many terms in computing the concept of fourth generation languages (4GLs) and environments is an extremely hazy one. One way to distinguish between third generation and fourth generation tools is to express the ideal of a non-procedural development environment. It must be remembered however that few if any contemporary products satisfy this ideal.

Third generation languages such as COBOL and C are inherently procedural in nature. That is, the programmer is required to specify not only what is needed but also how the computer is expected to solve the problem in a detailed, step-by-step manner. In contrast, fourth generation languages are ideally designed to be non-procedural. The programmer need only specify what is required. The how is left to the 4GL compiler or interpreter to sort out.

Another way of expressing this is to say that any algorithm is made up of two components: logic and control (Kowalski, 1979). In a third generation language, logic and control are necessarily intertwined. The programmer expresses both the solution to the problem and such things as sequencing and iteration in one medium. In the ideal 4GL some indication of the logic of the process is given: the general structure of the data needed to support the process; the general form of the output expected from the process, and the form of the interface needed between the user and the process. No indication need be given of the detailed control needed to accomplish the processing.

## **11.2 Components of a Fourth Generation Environment**

Given the above definition, few if any existing self-styled 4GLs are truly 4GLs. The contemporary form of 4GL is usually a higher-level programming language in the sense that it has in-built functions for information systems, particularly screen and database handling. In this sense, they are more like 3½ GLs.

The ideal of non-procedurality achieves more ready recognition in the idea of a fourth generation environment (4GE). A fourth generation environment can be conveniently used as an umbrella-term for a collection of distinct products, many of which can be classified as CAISE (see chapter 12):

1. *Physical data dictionary*. A place for storing the specifications both of the data used, and of the databases where this data is physically located.
2. *Screen painter*. Used to outline the format of any data entry and retrieval screens needed by an application. Such screens are normally driven from default output produced from the data dictionary.

3. *Report generator*. Used primarily to specify the format of printed reports, normally by reference to the database structures stored in the data dictionary.
4. *Query language*. A means of flexibly specifying non-procedural queries on a database.
5. *Dialogue specifier*. Used to indicate the nature of program flow, where this is not apparent from the non-procedural information provided.
6. *Code generator*. A means of generating 3GL code from data dictionary definitions.

Many environments include a high-level procedural language and other facilities such as spreadsheet and graphics tools within their environments. In section 11.4 we shall look at the offering available from the Oracle Corporation as an instance of a fourth generation environment (Sikora, 1997).

### **11.3 Application Generators**

The term application generator is frequently used as a synonym for fourth generation environment. An application generator is designed to create one or more of the standard pieces of an information system such as menus, data entry screens, enquiry programs, reports or batch programs. They work typically by displaying a series of parameters that can be modified by the developer. These customised parameters are then either interpreted by a shell when the application is run, or compiled into some form of source code, probably COBOL or C.

In the case of an update program, for example, the developer might specify which files are to be updated, the fields from the file that are to be displayed and modified, and the restrictions to be placed on the information to be entered. The application would then run without needing further input from the developer.

This generator approach works well when the data and processes needed by an application are not too complex. It is insufficient however for the more complex applications required by most information systems departments. One approach to solving this problem is to offer more screens in the generator such that the developer can specify the logic of the application in greater detail. Clearly however, this approach is limited by the fact that the parameter screens can become so complex as to be unusable. The answer adopted by most products is therefore to generalise as many of the user requirements as possible using the non-procedural approach, but then offer ‘hooks’ where the developer can attach modules of 3GL or procedural 4GL code.

### **11.4 The Oracle Kernel and Toolkit**

Modern day relational DBMS generally come in two parts, which we shall refer to as the kernel and the toolkit. The kernel comprises the core DBMS functions

generally implemented in some dialect of SQL. Around this standard interface most vendors offer a range of additional software tools for producing information systems. Thus the concept of DBMS discussed in chapter 10 is extending ever further outwards to encompass more and more areas of application building.

The kernel of the fourth generation environment comprises a relational database engine. Around this kernel lies a standard interface known as SQL\*Plus. SQL\*Plus allows the user to create and modify tables, enter and modify data, and set up *ad hoc* queries. SQL\*Plus is SQL with additional features for editing, running command files and controlling the format of output.

Around this kernel and interacting with this interface a range of products are offered:

1. SQL\*DBA provides the DBA with functions necessary to create, maintain, start, stop, and recover a database.
2. PRO\*SQL provides the means of integrating SQL with standard programming languages such as C, FORTRAN, COBOL.
3. Oracle Forms is a forms-based application development tool. It is primarily used to build data entry or retrieval screens that can call database objects such as PL/SQL programs and database triggers.
4. Oracle Reports is a package for complex report generation.
5. Oracle Graphics can be used to construct graphic representations (e.g., bar, line and pie charts) automatically for data that is the result of an enquiry on the database.
6. Oracle textretrieval is a tool that provides a means of managing unstructured data such as text and images.

Versions 7 and 8 of Oracle also encompass a procedural 4GL known as PL/SQL and a set of CASE tools based around entity-relationship diagramming.

## 11.5 Embedded SQL

SQL comes in two forms. It can either be used interpretively or it can be embedded within a host language such as COBOL, FORTRAN, PL/I, C, or Pascal. It is the former usage that we discussed in chapter 6. The latter usage is primarily for application developers. It is designed to simplify database input and output from application programs via a standard interface. In this section we introduce the concept of SQL as an embedded database language by discussing one simple example in a pseudo-host language modelled on Pascal.

```
PROGRAM emp_names;
```

```
VAR empno: integer;
```

```

BEGIN
  EXEC SQL DECLARE employees CURSOR FOR
    SELECT empno
      FROM employees;
  PRINT 'Employee Numbers';
  PRINT '_____';
  EXEC SQL OPEN employees;
  EXEC SQL FETCH employees INTO :empno;
  WHILE SQLCODE = 0 DO
    PRINT empno
    EXEC SQL FETCH employees INTO :empno;
  EXEC SQL CLOSE employees;
END;

```

The program above prints the list of employee numbers in a table named employees. Embedded data language statements are preceded by the keywords EXEC SQL. The DECLARE statement defines a pointer to the employees table known as a cursor. When the open statement is activated, the select statement defined at the declare stage is executed. This creates a copy of all the employee numbers in the base table and places it in a workspace area. To retrieve an actual record from this workspace we use the fetch command. The fetch statement takes an employee number and places it in a program variable empno, indicated by the colon. Sqlcode is a system variable. It returns 0 if the fetch executed successfully, non-zero otherwise. Hence we use it here to act as a terminating condition to traverse the employee numbers in the workspace.

The concept of a cursor is extremely important in that it acts as the cement between the inherently non-procedural nature of SQL and the inherently procedural nature of a third generation language such as Pascal. There is in fact what is usually referred to as an impedance mismatch between SQL and 3GLs. SQL, being relational, works at the file level. It takes tables as input and produces tables as output. 3GLs however work at the record level. They only process one record at a time. We therefore have to have some mechanism for translating from a file-based to a record-based approach. This is provided by the cursor.

A program written in this hybrid manner is normally submitted to a pre-compiler. The *exec sql* keywords are important in enabling the pre-compiler to identify SQL commands from Pascal commands. The pre-compiler will take the SQL commands and create a database access module. It will also take the Pascal commands and produce a source program suitably modified with database access calls. This modified source is then compiled in the normal manner.

## 11.6 PL/SQL

PL/SQL is Oracle's version of a procedural 4GL (the PL stands for procedural language) with embedded SQL. PL/SQL was first introduced with version 6 of Oracle. It was developed with the aim of permitting the construction of complex database operations completely within the database server. This means that the server does not need to pass back control after every SQL operation.

PL/SQL functions can be implemented in non-procedural development tools such as Oracle forms, Oracle reports and Oracle graphics. PL/SQL functions can also be packaged as stored programs on the server. PL/SQL is also used in the definition of database triggers.

PL/SQL is a block-oriented language. Every PL/SQL block consists of the following three components:

1. *Declarative part.* Contains the names and data types of variables and constants as well as cursor declarations.
2. *Executable part.* Contains SQL DML statements embedded in control constructs such as IF-THEN-ELSE. May also contain commands for transaction management.
3. *Exception handling part.* An exception is an error condition defined either by the user or by the system. Such exceptions can be trapped by a series of declared 'exception handlers'.

Below we give an example of a simple PL/SQL program:

```
DECLARE
deptCodeConst lecturers.deptCode%TYPE := 'CS';
totalRecs := number(3);

BEGIN
SELECT count(*) INTO totalRecs
FROM lecturers
WHERE deptCode = deptCodeConst;

IF totalRecs > 0 THEN
update lecturers
set deptCode = null
WHERE deptCode = deptCodeConst;
ENDIF;

COMMIT;
```

**EXCEPTION**

```

WHEN no_data_found THEN
INSERT INTO error_log1 VALUES(sysdate, deptCodeConst);
WHEN others
err_code := sqlcode;
err_text := sqlerrm;
INSERT INTO error_log1 VALUES(sysdate, err_code, err_text);

END;

```

The PL/SQL block above first counts the number of computer studies lecturers. If there are lecturers for this department it then proceeds to set their department code to null.

## **11.7 Implementing Integrity**

Oracle 7 offers a number of facilities for embedding integrity within a relational database. These facilities can be broadly distinguished in terms of whether they constitute procedural or declarative mechanisms.

### ***11.7.1 Declarative Mechanisms***

The declarative mechanisms constitute variants of SQL syntax for primary key, foreign key and other constraints. Each such constraint can be given a name and specified either at CREATE TABLE time or separately using an ALTER TABLE command. The examples below illustrate the process of declaring constraints in Oracle 7:

```

CREATE TABLE Modules
(moduleName CHAR(15),
level NUMBER(1) CONSTRAINT m_level_1 DEFAULT(1) NOT NULL,
courseCode CHAR(3),
staffNo NUMBER(5),
CONSTRAINT m_pk PRIMARY KEY (moduleName),
CONSTRAINT m_fk FOREIGN KEY (staffNo) REFERENCES Lecturers)

```

```

ALTER TABLE Modules ADD CONSTRAINT m_level_2 CHECK (level
between 1 and 3)

```

The first example will create the modules table with appropriate primary and foreign keys. Note that the primary and foreign key clauses have been given a

name via a CONSTRAINT command. This is useful in that any infringement of such constraints will then throw up the name of the constraint in any error message. The second example illustrates how a constraint may be added to an existing table declaration. Here a CHECK clause is being used to ensure that the level attribute lies within a given range of values.

### 11.7.2 Procedural Mechanisms

Integrity is maintained procedurally in Oracle primarily via database triggers. A trigger is a procedure written in PL/SQL which is directly assigned to a database table and which is automatically activated by specific actions on that table. For example, suppose we have an enrolments table which records which students have enrolled for which modules. Inserting an enrolment record into this table may activate a trigger that automatically updates the current roll of a particular module.

An Oracle trigger is characterised in terms of triggering statements (insert, update, delete), the trigger timing (before, after) and the trigger type (statement or row trigger). The trigger timing specifies whether the trigger should be executed before or after the triggering statement. The trigger type determines whether a trigger should be fired once per statement regardless of the number of rows affected or each time that row is modified.

Triggers are created using the CREATE TRIGGER command. A database trigger can contain any number of PL/SQL statements, SQL DML operations, or calls to other PL/SQL programs. Triggers can also make use of special IF constructs that allow structuring of PL/SQL code to be able to handle multiple events within one trigger. The example below illustrates the general structure of a trigger definition:

```
CREATE OR REPLACE TRIGGER lecturer_delete_trig
AFTER DELETE
ON Lecturers
BEGIN
    IF TO_CHAR(sysdate, 'HH24') > 13 THEN
        RAISE_APPLICATION_ERROR(-2010, 'Not allowed to
delete a lecturer now');
    END IF;
END;
```

The example above is a statement level trigger. It executes immediately a deletion against the Lecturers table. The trigger checks to see if the deletion has occurred after 13:00. If it does it raises an error.

## 11.8 Oracle 8

At the time of writing the Oracle Corporation has just released version 8 of their DBMS. This version includes a number of performance and functional enhancements in areas such as those described above. In particular, Oracle 8 is focused on enhancements in features that permit scalability for high-volume transaction processing. It also contains a number of extensions to the relational model in the area of object-oriented facilities for application development (Bobrowski, 1998).

## 11.9 References

- Bobrowski S. (1998) *Oracle 8 Architecture*. Oracle Press. Osborne/McGraw-Hill, Berkeley.
- Cobb R.H. (1985). 'In Praise of 4GLs'. *Datamation*. July 15th. 90-96.
- Kowalski R. (1979). 'Algorithm = Logic + Control'. *CACM*. 22. 424-436.
- Sikora Z.M. (1997). *Oracle Database Principles*. Macmillan Press, Hounds Mills, Basingstoke.

## 11.10 Exercises

1. Discuss the distinctions between a third generation language and a fourth generation language; also, a fourth generation language and a fourth generation environment.
2. Why do you think a data dictionary is central to a fourth generation environment?
3. In what ways do you think 4GEs may improve programmer productivity and the quality of information systems?
4. Discuss whether you think embedded SQL is an ideal mechanism for interfacing 3GLs with databases.
5. In what way do you think object-orientation is likely to affect the future of 4GE?
6. How is the concept of fourth generation related to CAISE?
7. In what way do you think database systems have influenced the development of fourth generation languages?
8. Modern fourth generation languages are really only three and a half generation languages. Discuss.
9. The IT director of Goronwy's holding company is considering purchasing a 4GE. Write a formal memo to him explaining the advantages and disadvantages of using a 4GE.

# ***12 Computer Aided Information Systems Engineering (CAISE)***

## **12.1 Introduction**

In this chapter we shall discuss the growing number of automated tools for information systems development. Many people place such tools within the context of CASE – Computer Aided Software Engineering. The author prefers the term CAISE – Computer Aided Information Systems Engineering – in that CASE may be regarded as a subset of CAISE. Many CAISE tools are involved in the direct construction of software and are hence logically software engineering tools. Many others are not directly involved in the production of software but in other artefacts associated with the IS development process. Most database design tools, for example, have as their remit the production of structures for storing and manipulating data. Also, project management tools enable the production of documents associated with the management of IS projects rather than any application-specific detail.

The large number and diversity of CAISE tools makes it impossible to do justice to a representative sample. We have therefore chosen to describe facilities of one particular popular and inexpensive product, namely PC Select.

## **12.2 A Model of the Development Process**

CAISE is a logical consequence of a recursive or incestuous view of information systems development. It has stimulated the view that information systems development, considered as an information system in itself, should be subject to and benefit from the same sorts of automation that characterise everyday information systems.

CAISE is therefore based upon a particular model of the information systems development process. In this model, the development process is seen as a set of activities operating on objects to produce other objects. The objects manipulated by such activities will frequently be documents, diagrams, file-structures or even programs. Similarly, the activities involved may be relatively formal (e.g. compile a program) or informal (e.g. obtain user's requirements). It is not surprising therefore that the waterfall model of information systems development and the associated adoption of structured techniques (chapter 7) is particularly suited to the application of CAISE (Parkinson, 1991).

In recent times many existing CAISE tools have adapted themselves to handling object-oriented methods. Many such tools have been marketed specifically at the area of object-oriented analysis and design (chapter 30).

### **12.3 Back-end, Front-end and Integrated CAISE Tools**

A distinction is normally made between back-end CAISE tools and front-end CAISE tools.

Front-end CAISE tools are generally directed at the analysis and design stages of information systems development. In terms of database development, for instance, front-end tools include such products as E-R diagramming editors (chapter 16), determinacy diagramming editors (chapter 15) and logical data dictionaries (chapter 19).

Back-end CAISE tools are directed at the implementation, testing and maintenance stages of information systems development. Here tools such as physical data dictionaries, performance monitors and other aids for physical database design are relevant.

Many vendors have now attempted to integrate their front-end and back-end tools. The intention is, for instance, to offer assistance at all the stages of information development, and to expect the outputs from each stage to feed as inputs to subsequent stages. This we might describe as integrated CAISE.

### **12.4 I-CAISE, C-CAISE and Meta-CAISE**

One of the major decisions for potential users of CAISE is whether to use component CAISE (C-CAISE) or integrated CAISE (I-CAISE). Component CAISE relies on CAISE software from different suppliers being able to work together to cover different parts of the project life-cycle. I-CAISE, such as the Information Engineering Facility from James Martin Associates, is a suite of CAISE tools from one supplier designed to work together. The heavy investment in Integrated Project Support Environments (IPSE) in the late 1980s are the precursors of I-CAISE (McDermid, 1985).

The main advantage of I-CAISE is that it imposes a rigid discipline. Developers have to learn only one tool-set. The main problems with I-CAISE are that many products are not as integrated as one would expect, and that opting for I-CAISE means being locked into one supplier. Hence, the supplier becomes a strategic partner in development plans.

C-CAISE offers a more flexible option in that organisation can pick what best suits their organisational approach. The main problem with C-CAISE is that the different products making up C-CAISE may use incompatible data structures, hence making communication difficult.

The common threads running through many of the techniques of modern systems development has meant that a certain degree of abstraction is possible in CAISE. The foremost example of such abstraction is a group of products collectively known as Meta-CAISE tools. Using such tools, an organisation can build its own CAISE toolkit perhaps to suit its own in-house methodology (chapter 27).

## 12.5 Facilities of CAISE

The objective of an integrated CAISE environment is to improve the effectiveness of an organisation in developing and maintaining information systems. In this sense it will provide facilities for development objects to be created, manipulated and communicated between members of a project team. Facilities for the management of projects (chapter 31) should also be a part, as should facilities for managing inter-project relationships. This can be summarised by saying that CAISE can provide a gearing effect at three different levels:

1. Individual
2. Project
3. Corporate

### *12.5.1 Individual Gearing*

The individual gearing that can be provided by CAISE is concerned with enabling a person to carry out his or her activities more effectively. Examples of the types of facilities provided by CAISE to facilitate individual gearing are:

1. For a programmer, better compilers and higher level languages such as 4GLs (chapter 11).
2. For an analyst better documentation facilities, particularly diagramming editors.
3. For a project manager, control tools such as PERT planners (chapter 31).

### *12.5.2 Project Gearing*

CAISE support at the project level is concerned with providing means for co-ordinating the activities of a set of individuals. Support at this level includes:

1. Integration of tools to ensure smooth information flow between team members.
2. Support for standards.
3. Maintenance of automatic version control and other aspects of configuration management.

### *12.5.3 Corporate Gearing*

In any business reliant on information systems, there will be a number of discrete but interrelated projects. The effectiveness of the company will depend upon its ability to control the interrelationships between projects. Such

relationships might be technical (e.g. one project relies on the framework of another), to do with resources (e.g. one project may rely on another project finishing to release key skills) or concerned with business effectiveness. Examples of facilities that relate to gearing at the corporate level include:

1. Tools for strategic information systems planning (chapter 33).
2. Dissemination of working practices, methods, and tools.
3. Construction and utilisation of a corporate 'experience-base' (e.g. the availability of component libraries).

## **12.6 The Advantages of CAISE**

Many of the advantages of CAISE arise from automation. Let us examine, for instance, the process of constructing data models in terms of entity-relationship diagrams. It is relatively straightforward to produce small entity models on paper. As soon as we start to scale up the exercise to realistically large levels however many problems emerge. Some paper-based data models end up being displayed at one end of the office and terminate two hundred yards away at the other side of the building! The number of entities, relationships and attributes in such data models makes a paper-based storage mechanism impracticable.

Many of these problems of scale are well known from software engineering. Much work has been invested in building integrated project support environments (IPSE) designed to support large teams of software developers working on long-term projects (McDermid, 1985). Some of the key problems which CAISE tools for database work address are:

1. In terms of a large data model we have to ensure that we maintain the integrity of the data model in much the same way as we have to maintain the integrity of an everyday database. CAISE tools will contain automatic checking and error-finding facilities.
2. We have to ensure that all members of the development team are using the same notational devices. A CAISE tool helps to enforce standards.
3. We have to ensure that concurrent access to the model is controlled and that the data model remains an accurate reflection of design decisions. We also have to ensure that a history of design decisions is available for inspection. A version control facility is therefore essential for most CAISE tools.

## **12.7 Problems with CAISE**

One of the fundamental problems of CAISE is that it has frequently been seen as a panacea for all the ills of systems development. This it is clearly not. Spurr (1989) more accurately portrays CAISE as a culture shock for many organisations.

One of the main lessons that has been learnt by organisations applying CAISE technology is that CAISE demands large investments of time and effort, particularly in the training of staff. Most CAISE tools are inherently linked with given methodologies (chapters 27 and 30). Hence, to apply CAISE products effectively one must also adopt the tenets of the methodology the product supports. The consequence of this is that organisations frequently have to invest in changes of methods as well as changes in technology. This forces project management techniques to be reviewed.

## 12.8 Select

Select is an integrated and relatively inexpensive CAISE tool. It offers a number of specific versions tailored for specific methodologies. For example, it has a version for SSADM – Select/SSADM (chapter 27) – and Select/OMT (chapter 30). The facilities offered by this package fall mainly into the front-end category, although there are some facilities for things such as schema generation that fall into the back-end category. Select/SSADM particularly allows people to construct the standard documentation such as Logical Data Structures (effectively E-R diagrams), data flow diagrams and entity-life histories associated with the SSADM methodology. Without the support of a CAISE tool such as Select/SSADM, the production of SSADM deliverables such as LDSs, DFDs and ELHs would be a laborious and error-prone task. The use of Select/SSADM ensures that the SSADM diagrammatic conventions and rules are adhered to. This form of individual gearing means, for instance, that it is impossible, using Select/SSADM, to draw a data flow arrow connecting two data stores in a DFD, or to use incorrect conventions for representing the optionality of relationships in a logical data structure.

Select allows people to set up ‘projects’ which may contain all the documentation associated with a particular development project. A number of people can work concurrently within the domain of one particular project. This enables a degree of project gearing as discussed in section 12.5.2.

One way in which Select/SSADM can be used as a back-end facility is by automatically generating a file of SQL DDL statements (chapter 10) from a logical data structure. To do this an LDS must first be extended by defining the primary/foreign keys associated with each entity and data-typing the other attributes of each entity. Having done this, the user can submit the design to a process that generates a file of SQL Create table and Create index statements.

## 12.9 Conclusion

CAISE by its very nature is an attempt to automate the process of information systems development. Some people see its eventual aim as being the provision of

a sufficient range of facilities to make information systems development a factory production process. In this sense, the aim is to provide an information system for the production of other information systems.

There is however a major problem with this aim. As of now, there is no consensus as to what constitutes a suitable information system for the production of information technology systems. The rise of standard structured methodologies such as SSADM and object-oriented methodologies such as OMT has undoubtedly stimulated a certain degree of commonality among the key CAISE vendors. In general, however, the diversity of CAISE still reflects the diversity of current methods and techniques for information systems development. This is particularly true in the developing area of object-oriented methods.

### **12.10 References**

- Codd E.F. (1990). *The Relational Data Model for Database Management. Version 2*. Addison-Wesley, Reading, Mass.
- McDermid J. (1985). (Ed.). *Integrated Project Support Environments*. Peter Peregrinus, London.
- Parkinson J. (1991). *Making CASE Work*. NCC Blackwell, Oxford.
- Spurr K. (1989). 1(5). 'CASE: A Culture Shock'. *The Computer Bulletin*. June. 9.

### **12.11 Exercises**

1. What is CAISE?
2. Describe the distinction between front-end and back-end CAISE tools.
3. Discuss the distinction between Component-CAISE, Integrated-CAISE and Meta-CAISE.
4. What are the advantages of CAISE?
5. You are given the remit to assess the potential of CAISE tools for the IT needs of your organisation. Produce a short report stating the desirability and feasibility of applying CAISE.
6. CAISE is a logical development of the structured methods movement. Discuss.
7. Discuss whether the idea of an information systems factory is feasible.
8. Is it reasonable to expect a coherent end-product from a set of component CAISE tools?
9. How might be CAISE be applicable to the Goronwy Galvanising project?

# ***13 Multimedia and Hypermedia Systems***

## **13.1 Introduction**

Hypermedia is the approach to building information systems made up of nodes of various media (such as text, audio data, video data, etc.) connected together by a collection of associative links. A subset of this discipline known as hypertext concentrates on the construction of loosely connected textual systems. The term multimedia is frequently used as a synonym for hypermedia, although strictly multimedia refers to any system that handles multiple media. Hence, multimedia may be seen as superset of hypermedia.

This chapter discusses the general concepts underlying hypermedia systems and particularly identifies the place of hypermedia in current and future information systems. The most important current relevance of hypermedia is in the context of the phenomenon known as the Internet.

## **13..2 History**

Vannevar Bush (1945) envisaged hypermedia systems in the 1940s. Bush discussed the concept of a memex (memory extender), a device capable of storing and retrieving information on the basis of content.

In 1968, Douglas Englebart demonstrated the Augment system. Augment is an on-line working environment designed to augment the human intellect. It could be used to store memos, research notes and documentation. It also had facilities for computer conferencing and collaborative working.

Ted Nelson extended Bush's original concepts in the Xanadu project. Xanadu is designed to be an ever-expanding environment that can be used to create and interconnect documents containing text, video, audio and graphics. Nelson actually coined the term hypertext and defined it as being 'non-linear reading or writing'.

A number of prominent hypermedia prototypes were developed in the 1980s. Amongst these were NoteCards, InterMedia and Zog.

However, probably the single most important factor that has contributed to the rise of interest in hypermedia is HyperCard, a software tool packaged with the Apple Macintosh. In recent times hypermedia systems have formed the bedrock of activity on the Internet.

### 13.3 Hypertext

Hypertext is normally discussed in terms of three models of organisation for text: linear text; hierarchical text and network text:

1. *Linear text.* Linear text is exemplified in the format of a conventional novel. A novel is linear text in the sense that the reader is expected to start at the beginning and progress to the conclusion via sequential reading of chapters.
2. *Hierarchical text.* Many textbooks or reports are organised hierarchically in terms of chapters, sections, sub-sections, etc. The reader can now access the material at a number of different points in the hierarchy.
3. *Network text.* The dictionary or encyclopaedia exemplifies network text. In a dictionary each entry has an independent existence but is linked to a number of other entries via references.

Although examples of linear text and hierarchical text are discussed in the literature on hypertext, it is the network model that most hypertext systems attempt to emulate. A hypertext system might be regarded as an on-line implementation of network text (Conklin, 1987).

Rada (1991) also makes the distinction between small-volume hypertext, large-volume hypertext, collaborative hypertext and intelligent hypertext. Most existing commercial systems have been small-volume, standalone systems. Multi-user, large-volume hypertexts with intelligent access are still very much a research endeavour.

Contemporary hypertext systems would not be possible without the development of direct manipulation, WYSIWYG (What-You-See-Is-What-You-Get) (see chapter 23) interfaces with high volume storage devices such as CD-ROM.

The present application of hypertext systems lie in the area of on-line dictionaries, encyclopaedias and medical textbooks. The commercial world is already seeing something of an explosion of products in this, ranging from interactive catalogues, technical documentation, help systems and computer-aided instruction.

### 13.4 Hypermedia

Hypermedia expands the concept of hypertext with various other media:

1. Data – as held in databases and spreadsheets
2. Graphics – icons, photographs, images, diagrams
3. Sound – commentary, sound-effects
4. Video – film, animation

Some companies such as Walt Disney have already used systems in their theme parks designed to affect our kinaesthetic, tactile and olfactory senses. The term virtual reality in particular is becoming associated with systems that offer a multi-sensory interface to computer systems and it is likely that more and more interfaces to information systems in the future will contain interfaces of this form.

### 13.5 A Sample of Hypermedia Systems

In this section we outline three early sample applications of hypermedia which demonstrated the feasibility and utility of the technology. It should be emphasised that many commercial organisations such as banks and travel agencies have already implemented products in this area.

#### 13.5.1 *The Oxford English Dictionary*

Darrel and Tampa (1988) report how the Oxford University Press have investigated a hypertext version of its world-famous *Oxford English Dictionary* (OED). The OED is the largest and most scholarly dictionary of written English in existence. In its standard form, the OED consists of 12 volumes containing 41 million words. The main reason for considering hypertext and the use of CD-ROM is to support browsing of a large text database.

#### 13.5.2 *The Dynamic Medical Handbook*

Frisse (1988) describes a project that attempts to demonstrate the feasibility of hypermedia systems for medical information. The idea is to support the various cognitive needs of medical staff in locating diverse types of information (text, pictures, case histories) on specific medical conditions quickly and efficiently. The project is particularly interested in allowing the system to adapt to the needs of its users over a period of time.

#### 13.5.3 *The National Gallery System*

One of the most significant hypermedia developments in the UK made its debut with the opening of the Sainsbury wing of the National Gallery in 1991. Called the micro gallery it is the culmination of three years of intensive development in C and Hypercard 2.0. Designed to aid the adult visitor wanting more information on any of the gallery's paintings it allows one to access via touch screens over 4,000 pages of text and thumbnail reproductions. The user can access the information via a number of routes: e.g., by artist's name, by historical period or area, and by artistic period.

### 13.6 Intranet and Internet

Hypermedia and indeed multimedia tend to get associated these days with the modern phenomenon known as the Internet. The Internet is a set of interconnected computer networks distributed around the globe and can be considered on a number of levels. The simplest level of connection is that defined by electronic mail (e-mail) which allows diverse users to communicate by distributing electronic messages to each other. A step up from this includes electronic data interchange (EDI), a set of standards for transmitting electronic documentation between organisations. However, in this section we particularly concentrate on the technology of the World-Wide Web, as it appears to be the one most readily associated with the Internet concept at the current time.

The World-Wide Web (WWW) (referred to colloquially as ‘the Web’) began as a project in 1992 at the European Centre for Nuclear Research (CERN) based in Geneva. The WWW can be thought of as a collection of documents residing on thousands of servers or Web sites around the world. Each such document is written in the hypertext markup language (HTML) and can be made up of text, graphics, images, audio-clips and video-clips. Documents also include links to other documents either stored on the local HTML server or on remote HTML servers. HTML documents are identified by Universal Resource Locators (URL): a unique address for each document on the Web. Links are activated by ‘hotspots’ in the document: a word, phrase or image used to reference a link to another document. It is this feature of linking diverse documents together which defines hypertext as a technology.

To access the WWW one needs a browser: programs that let the user read hypertext documents, view any in-built images and activate hotspots. Browsers can also be used to link to FTP servers, gophers and WAISs:

1. File Transfer Protocol (FTP) servers contain large collections of files that can be transferred over computer networks using e-mail and/or browser software.
2. Gopher is the name given to a way of finding one’s way around the Internet using menu-driven interfaces. The name is taken partly from the idea of its human counterpart ‘going for things’ and partly because a gopher is the mascot of the University of Minnesota where the idea was developed. A gopher has two parts: one part resident on the client is used to specify the information required; the other part stored on the server contains an index of terms that the user can navigate.
3. Wide Area Information Servers (WAIS) are facilities which allow free-text retrieval searches against vast quantities of textual material.

Figure 13.1 illustrates the way in which a web brower can be used to access information. The HTML page presented here acts as a switchboard to a number of information sources useful to academics in the Information Systems area.

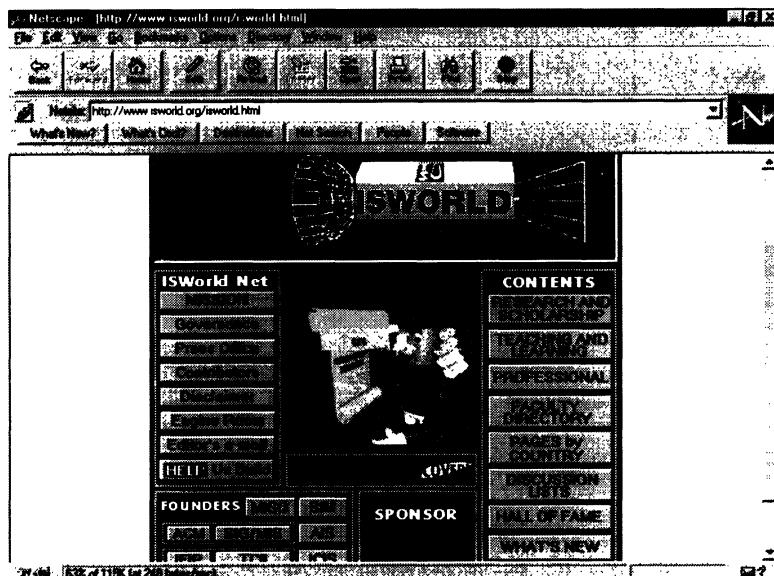


Figure 13.1: A WWW page presented through a Web Browser

Internet technology such as WWW has become particularly popular because of the way it has established readily available 'open' standards for electronic communication. It is therefore not surprising that many organisations are beginning to use Internet technology for communications applications internal to the organisation. Intranet is using internet technology within the context of a single organisation. At its most basic it involves setting up a Web service for internal communications and co-ordination. At its most sophisticated, it involves using Web interfaces to core corporate applications such as corporate-wide database systems.

### 13.7 Thin Clients

Some discussion has recently occurred within the computing fraternity over the way in which Internet technology may cause a profound change from 'fat' client to 'thin' clients. The corporate PC is currently a fat client. It requests information from a server and then processes and presents it at the client end using software resident on the PC. The network computer or NC is a thin client. In its extreme form, only a minimum of software will be resident on the client. Applications will be resident on and accessed from the server. This means that each user will have a desktop system that looks like a PC but which has no secondary storage devices. Consequently, NCs, so the argument goes, are likely to be considerably less expensive than a PC. Also, the network administrator will

only need to buy and maintain one copy of each software application on the server.

To make full application development possible in thin client configurations, current construction tools such as HTML are insufficient. This is where an idea from Sun Microsystems steps in: Java. Java is a development language, a variant of C++, which runs on a virtual machine. This means that it can run theoretically on any machine and can be transmitted using the common communication protocol TCP/IP – Transmission Control Protocol/Internet Protocol. Many application developers are rewriting their software or parts of their software in Java. The language was used to write Sun Microsystem's Hot Java Internet Browser. Running Hot Java on some machine means that you can access a Web page written in Java that may contain one or more applications called applets. These applets can be sent to the client automatically or on some command from the user such as pressing a button. They will then run immediately and return information that can be dealt with coherently by the server.

### **13.8 Conclusion**

In this chapter we have provided a brief overview of a technology which is certain to dominate user interfaces into the next century. In chapter 23 we outline how direct manipulation interfaces popular in hypermedia systems are beginning to dominate the issues surrounding user interface development. Many are predicting that interfaces built using Web-based standards are likely to dominate in the near future because of the benefits that it has in terms of standardisation.

### **13.9 References**

- Apple (1987). *Hypercard User's Guide*. Cupertino, California.
- Bush V. (1945). 'As We May Think'. *Atlantic Monthly*. 176(1). July. 101-103.
- Conklin E.J. (1987). 'Hypertext: An Introduction and Survey'. *IEEE Computer*. 2(9). Sept. 17-41.
- Darrel R.R and Tampa F.W.M (1988). 'Hypertext and the Oxford English Dictionary'. *CACM*. 31(7). July. 871-879.
- Frisse M.E. (1988). 'Searching for Information in a Hypertext Medical Handbook'. *CACM*. 31(7). 880-886.
- Rada R. (1991). *Hypertext: from text to expertext*. McGraw-Hill, London.

### 13.10 Exercises

1. In what way is a hypermedia system different from a database system?
2. How much hype do you think there is in hypermedia?
3. What sort of media might be important to an insurance company handling motor accident claims over and above conventional data?
4. Why do you think object-oriented systems have been seen as good vehicles for implementing hypermedia?
5. What sort of hypermedia systems might be of use to an organisation such as an University?
6. In what area might hypermedia be applicable to the Goronwy Galvanising application?

# **14 Knowledge Based Systems**

## **14.1 Introduction**

Knowledge based systems (KBS), sometimes referred to as intelligent knowledge based systems (IKBS), were much discussed in the literature during the 1980s. In the 1990s they have become an accepted tool in the information systems developer's toolkit. This chapter attempts to explain exactly what is meant by a knowledge based system and makes some suggestions as to the place of knowledge based systems within information systems development.

The easiest way to approach the problem of defining a KBS is to contrast it with a database system, particularly a relational database system as discussed in chapter 10 (Beynon-Davies, 1991). In this chapter we introduce the concept of a semantic data model and the related concept of an object-oriented database. These are used as vehicles to discuss the important concepts of facts, rules and inference.

## **14.2 Information and Knowledge**

Standard relational databases store facts about the properties of objects in some domain and some primitive information about the relationships between objects. For instance, the insurance database in chapter 10 represents two objects: policies and policy-holders. The tables store facts about the properties of the objects such as the renewal date of a particular policy. The tables also store facts about the relationships between particular policies and particular holders.

However, facts can be said to come in five major forms, many of which cannot be handled conveniently in a relational database:

1. Relationships between objects and object classes, sometimes referred to as ISA relationships. ISA is similar to set membership. When we state that Paul Beynon-Davies ISA PolicyHolder we are defining a particular object as being a member of the set of policy holders.
2. Relationships between classes and other more general object classes, sometimes referred to as AKO (A Kind Of) relationships. AKO is similar to the idea of a subset. When we state that PolicyHolder AKO Customer we are defining the set of policy holders to be a subset of the set of customers.
3. Relationships between object classes that do not involve issues of generalisation, sometimes said to be relationships of association. When we state that PolicyHolder Holds Policy, we are stating that there is an inherent link between policy holders and policies in the sense that data about policies is associated with data about policy-holders.
4. Relationships between an object and a property or attribute (HASA relation-

- ships). When we state that Policy HASA RenewalDate we are specifying that members of the class of policies have a renewal date attribute.
5. PARTOF relationships compose an object out of an assembly of other objects. When we state that railwayStation PARTOF railway and railwayLine PARTOF railway we are specifying a railway as being made up of an aggregation of stations and lines.

### 14.3 Semantic Data Models

The relational data model is fundamentally only designed to handle HASA relationships and ASSOC relationships. The various other relationships can be represented in the relational data model, but not in a natural manner. In other words, the relational data model does not offer a sufficiently rich set of constructs for modelling the ‘real world’. The past couple of decades have therefore seen the emergence of a large number of alternative data models. This collection of data models can be loosely categorised as ‘semantic’ since their one unifying characteristic is that they attempt to provide more meaningful content than the relational model.

Semantic data models are conceptual models; that is, they are primarily models expressed in abstract or theoretical terms, much in the sense that the original relational model was expressed. However, just like the relational model, a number of semantic models have achieved a practical realisation. In recent years many of these systems have characterised themselves as object-oriented database systems because they embody many of the concepts discussed in chapter 9.

In the next section we illustrate some of the advantages of a database built using a data model known as the binary relational data model. This data model can encapsulate all of the relationships discussed in section 14.2.

### 14.4 Binary Relations

The binary-relational (BR) approach regards a universe of discourse as consisting of entities with binary-relationships between them. An entity is any thing of interest in this universe. A binary-relationship is an association between two entities. The first entity in a relationship is called the subject and the second entity in a relationship is called the object. A relationship is specified by identifying the subject, the type of the relationship, and the object. We shall write relationships as subject-type-object triples, such as Customer HASA Name.

Binary relations can be used to store both intensional and extensional information. Intensional information defines the structure of information in the knowledgebase. Extensional information comprises the actual occurrences of relationships.

Intensional information is defined through use of four predefined relationship types: ISA, AKO, HASA and PARTOF.

The binary relational store (BRS) is a data structure into which BR triples can be inserted and from which triples may be deleted or retrieved. Triples are inserted singly and retrieved or deleted in sets. Intensional information specified in the manner described above is used to define the allowable structure for the extension of a given BR database. That is, any insertion, amendment or deletion activity made on the BRS is checked against the intensional specification of AKO, ISA, PARTOF and HASA triples.

The BRS may hence be thought of as a black box into which both intensional and extensional information is written and from which both intensional and extensional information is retrieved.

Three basic operations can thus be performed with this store:

1. A triple can be inserted into the store.
2. A triple or set of triples can be deleted from the store.
3. A triple or set of triples can be retrieved from the store.

#### **14.5 An Example Semantic Database**

Suppose we wish to build a semantic database to store information on building society customers and the accounts they hold. We first need to create the object classes in the domain. This we do by inserting the following triples into the BRS:

Customer HASA Name  
Customer HASA Branch  
Customer HASA TelNo  
BuildingAccount HASA StartDate  
BuildingAccount HASA CurrentBalance  
OrdinaryShare ISA BuildingAccount

The first few triples merely build the properties of objects. The last triple declares ordinary share accounts to be a subclass of building society accounts. In a true database system we would also probably need to provide a data type for each of the properties of an object. For example:

Name TYPE CHAR(20)  
StartDate TYPE DATE

Data can then be recorded against each object:

4324 ISA OrdinaryShare  
 4324 StartDate 12/02/1997  
 4324 CurrentBalance 50

Note that to create an object we first have to declare an identifier for an object and assign it to a given object class. The system can then prompt for the relevant properties of the assigned class.

In its present form, the example could be implemented in a relational database, excepting the need to introduce primary keys. Suppose, however, we add the following triples to the BRS:

BuildingAccount HASA InterestRate  
 OrdinaryShare InterestRate 12

Here we have established a fact, an interest rate of 12%, relevant to all ordinary share accounts. Establishing this fact allows us to infer the following triple:

4324 InterestRate 12

#### 14.6 Inference

The word inference is derived from the Latin words *in* and *ferre*, meaning to carry or bring forward. Inference is therefore the process of bringing forward new knowledge from existing knowledge. In the example above, the existing knowledge represented by the collection of triples in the BRS is turned into new knowledge, a new fact, by the application of inference. In our case, this is via the application of the following inference rule:

IF Object ISA ObjectClass  
 AND ObjectClass Property Value  
 THEN Object Property Value

Similar inference rules apply to AKO and PARTOF relations.

We can generalise the concept of a rule to include rules specific to the application domain – domain-specific rules. For instance, we might write the following rule into our building society system:

IF Object1 ISA Customer  
 THEN Object1 Holds Object2  
 AND Object2 ISA BuildingAccount

This statement is basically specifying the fact that all customers of the building society can be assumed to hold building society accounts. A rule such as this can

be used in a number of ways. Probably the most relevant would be for the system to request details of a building society account after completion of customer entry, or to request details of a customer after entry of account details. This example is actually an instance of an integrity constraint, similar in nature to the entity and referential integrity constraints of the relational model (see chapter 10).

Domain-specific rules can also be used in a more proactive sense to propagate changes to the BRS. Proactive rules are referred to as transition constraints in the sense that they specify valid transitions between states of a database system. Consider the rule below:

```
IF Object ISA Employee  
AND Employee Location B  
THEN Object ISA Customer  
AND Customer Branch B
```

Here we are stating that if someone is an employee, he or she is also a customer (i.e. holds a building society account) and his or her branch should be set to the location of employment. In other words, if we entered the triples:

```
PDJames ISA Employee  
PDJames Location Pontypridd
```

into our knowledgebase, the rule above would cause the following triples also to be inserted:

```
PDJames ISA Customer  
PDJames Branch Pontypridd
```

#### **14.7 Semantic Nets, Frames and Objects**

A set of binary relations can be conceived of as a representation of a semantic net: a knowledge representation formalism popular in AI and Cognitive Psychology (Roussopoulos and Mylopoulos, 1989). Many of the concepts embodied in semantic nets, particularly generalisation hierarchies, have influenced developments in the areas of frame-based systems (Minsky, 1975), semantic data modelling (King and McLeod, 1985) and object-oriented databases (Brown, 1991). The characterisation of semantic nets in terms of binary relations is evident in the work of Sowa (1984).

The discussion of rules in section 14.6 has much in common with the idea of production rules in expert systems. Indeed, the type of system described above can be seen as an amalgam of expert system and database system concepts (Beynon-Davies, 1991).

### 14.8 An Expert System Application

An expert system might be defined as a computer system that uses a representation of human expertise in some domain. Consider a simple system designed to help customers select from among a number of investment accounts offered by a building society. Suppose we conduct an investigation and find that the following factors help us to distinguish between types of account:

1. The minimum investment required.
2. The amount of access needed to the money invested.
3. Whether a regular income is required from the investment.

The following rules might constitute an expert system for investment selection:

IF Account Investment I  
AND I >= 5  
AND Account Access instant  
AND Account income none  
THEN Account ISA OrdinaryShare

IF Account Investment I  
AND I >= 500  
AND Account Access instant  
AND Account income none  
THEN Account ISA InstantXtra

IF Account Investment I  
AND I >= 500  
AND Account access 90days  
AND Account income none  
THEN Account ISA 90DaysXtra

IF Account Investment I  
AND I >= 1000  
AND Account Access 90days  
AND Account Income monthly  
THEN Account ISA MonthlyIncome

IF Account Investment I  
AND I > 1000  
AND Account Access 90days  
AND Account Income monthly  
THEN Account ISA GoldOption

The inference mechanism in our system is known as backward or goal-directed chaining. This means that the system is first given a goal to solve. In the simple example above there is only one goal – to find a value for account. To find a value for account the inference mechanism works out that it needs a value for investment, access and income. The system first asks the user for an investment amount. The user replies 550. The system asks for a value for access. The user replies ‘instant’. Finally, the expert system asks for a value for income. The user replies none. The system then fires rule two and returns with a recommendation of an instant Xtra account.

#### 14.9 Conclusion

The term knowledge has been the subject of debate for centuries. Knowledge in the computational perspective might be defined as being the symbolic representation of aspects of some named universe of discourse (Winston, 1984). Note the two assumptions of this definition:

1. We can symbolise knowledge. That is, it can be represented in some way. Hence, AI has often seen itself as being a discipline concerned with symbolic processing (Simon, 1969).
2. We understand that an area of knowledge, often referred to as a knowledge domain, can be named or referenced in some way.

A more pragmatic definition of knowledge is to say that knowledge is made up of three component parts: facts, rules and inference.

Knowledge engineering is the discipline devoted to building knowledgebase systems. Knowledge engineering is normally defined in terms of two major types of activity:

1. *Knowledge elicitation* (sometimes referred to as knowledge acquisition). The process of extracting knowledge from one or more experts in a particular domain.
2. *Knowledge representation*. Where knowledge is stored in a knowledgebase in a form most appropriate for the given application.

Knowledge elicitation is a methodological issue. In the large, it concerns the whole process of interacting with people and documenting the results of this interaction. In contrast, knowledge representation is an implementation issue. It concerns the means of implementing the ‘intelligence’ involved in some particular domain in a computational medium.

The term knowledge representation is conventionally used purely in the domain of Artificial Intelligence. However, this is a narrow usage of the term. Most of what we mean by computing, particularly commercial computing, is

knowledge representation. Knowledge representation is not simply the domain of the AI worker; it is of profound concern to the information systems professional.

#### 14.10 References

- Beynon-Davies P. (1991). *Expert Database Systems: A Gentle Introduction*. McGraw-Hill, Maidenhead.
- King R. and Mcleod D. (1985). 'Semantic Data Models'. In Bing Yao S. (Ed.) *Principles of Database Design, Vol. 1: Logical Organisations*. Prentice Hall, Englewood Cliffs, N.J.
- Minsky M. (1975). 'A Framework for Representing Knowledge'. In Winston P. (Ed.). *The Psychology of Computer Vision*. McGraw-Hill, New York. 211-277.
- Roussopoulos N. and Mylopoulos J. (1989). 'Using Semantic Networks for Database Management'. In Mylopoulos J. and Brodie M. *Readings in Artificial Intelligence and Databases*. Morgan Kaufmann, New York.
- Sowa J.F. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, Mass.
- Winston P.H. (1984). *Artificial Intelligence*. Addison-Wesley, Reading, Mass.

#### 14.11 Exercises

1. Write BR triples to represent the relational database in section 10.7.1.
2. Suppose you are given the brief of building a knowledgebase to represent the planets of the solar system. Specify the planets and their properties as a series of BR triples.
3. Assume we create the following knowledgebase: Zeus ISA Male; Ares ISA Male; Hera ISA Female; Hephaestos ISA Male; Zeus ParentOf Ares; Hera ParentOf Ares; Zeus ParentOf Hephaestos; Hera ParentOf Hephaestos
4. Write a rule for defining that some object is the brother of some other object, and generate the appropriate inference from the knowledgebase above.
5. Discuss some of the advantages that knowledgebases may have over conventional databases.
6. Express the idea that an investment portfolio is made up of a collection of different types of investments using the syntax discussed in section 14.2.
7. Philosophers define knowledge as 'justified true belief'. Contrast this with the computational definition of knowledge given in section 14.9.

## ***Part Three*** ***Techniques***

### **Technique**

Method of achieving one's purpose.

(*Oxford English Dictionary*)

Manner, style, mode, line, procedure, process, way of doing.

(*Roget's Thesaurus*)

In this part we introduce a number of the major analysis and design techniques in the armoury of the information systems engineer. The techniques are divided into three groups: data analysis, process analysis and related techniques. The first two groups relate to the dynamic/static divide discussed in chapter 5 and particularly form those techniques originally proposed in the structured analysis and design movement. The last group contains a series of techniques that either fall outside of classic systems analysis and design, or attempt to bridge across process and data analysis. In the latter area we particularly focus on the issue of object modelling.

## ***Part Three Section One***

### ***Data Analysis***

In this section we discuss the central process of contemporary database development – a process known as data analysis or data modelling. In the next section we discuss a number of process analysis techniques.

The intention of data analysis is to develop stable foundations for application systems. The end-result of data analysis is a set of data structures designed to suit some application. This set is generally called a data model: a set of business rules embodied in some architectural principles such as those described for relational databases in chapter 10.

There are normally held to be two complementary approaches to conducting data analysis: top-down and bottom-up.

Conducting data analysis top-down means that we use a diagramming technique such as entity-relationship diagramming to map what we believe to be the things of interest to the enterprise and the relationships between these things of interest. Top-down data analysis is frequently referred to as conceptual modelling because we remain at a high level or on a fairly abstract plane. The product of such a modelling exercise is usually an entity-relationship diagram. This diagram can be transformed into a set of table-structures – a relational schema – via a straightforward process of translation or accommodation.

Rather than dealing with abstract concepts, bottom-up data analysis deals with concrete data. To do bottom-up data analysis we must have a pool of data items, extracted probably from an examination of existing enterprise documentation. To this pool of data items we apply a series of transformation rules. Bottom-up data analysis is also called normalisation.

We consider normalisation first, as this approach directly relates to the principles of the relational data model. We then consider entity modelling, a technique on which most database development is based.

# **15 Normalisation**

## **15.1 Introduction**

In his seminal paper on the relational data model, E.F. Codd formulated a number of design principles for a relational database (Codd, 1970). These principles were expressed in terms of three normal forms. The process of transforming a database design through these three normal forms is known as normalisation. By the mid-1970s third normal form was shown to have certain inadequacies and a stronger normal form, known as Boyce-Codd normal form was introduced (Codd, 1974). Subsequently, Fagin introduced fourth normal form and fifth normal form (Fagin 1977, 1979).

## **15.2 Why Normalise?**

Suppose we are given the brief of designing a database to maintain information about the infant immunisation programme run by a district health authority (Beynon-Davies, 1992). An analysis of the documentation used by the programme staff gives us the following sample data set with which to work. If we pool all the data together in one table as below, a number of problems would arise in maintaining this data set.

**Immunisations**

<b>practice name</b>	<b>doctor name</b>	<b>infant nhsNo</b>	<b>infant name</b>	<b>parent nhsNo</b>	<b>vacc. code</b>	<b>vaccine name</b>
Ystrad	J Thomas	12456	Jenkins	14534	1523	Mumps
Ystrad	J Thomas	25643	Thomas	22223	1524	Mumps
Ystrad	D Evans	43256	Jenkins	14534	1525	Mumps
Ystrad	D Evans	43256	Jenkins	14534	1425	Polio
Pentre	P Davies	33445	Evans	38976	1626	Polio
Pentre	P Davies	42389	Davies	22447	1627	Polio
Pentre	P Davies	42389	Davies	22447	1342	Rubella
Pentre	I Jones	33129	Howells	35612	1324	Rubella
Treorci	F Evans	32445	Evans	30976	1726	Polio

1. What if we wish to delete patient 32445? The result is that we lose some valuable information. We lose information about a doctor and practice. This is called a deletion side effect.
2. What if we change the doctor of patient 12456 from J. Thomas to D. Evans? We need to update not only the doctor name but also the name of the practice. This is called an update side effect.

3. What if we admit a new patient on to the programme, say 7777, Vaughn? We need to know more information, namely about the patient's doctor, practice, and parent. We also cannot enter a patient record until a patient has had at least one vaccination. These are known as insertion side effects.

The size of our sample file is small. One can imagine the seriousness of the file-maintenance anomalies mentioned above multiplying as the size of the file grows. The above organisation is therefore clearly not a good one for the data of this enterprise. Normalisation is a formal process whose aim is to eliminate file maintenance anomalies.

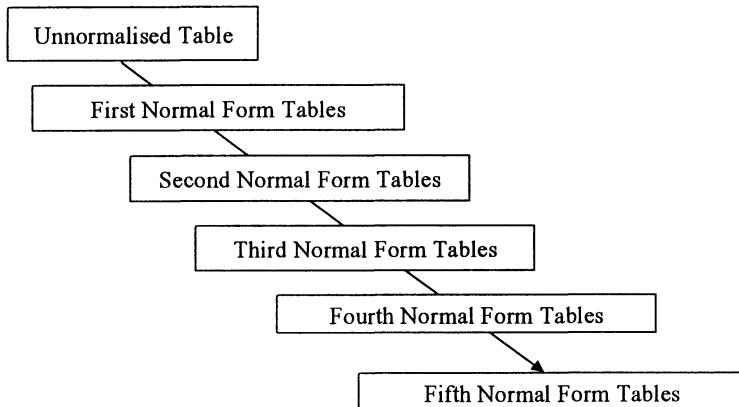
### **15.3 Stages of Normalisation**

Normalisation is carried out in the following steps:

1. Represent the data as an unnormalised table.
2. Transform the unnormalised table to first normal form.
3. Transform first normal form tables to second normal form.
4. Transform second normal form tables to third normal form.

Occasionally, the data may still be subject to anomalies in third normal form. In this case, we may have to perform further steps:

1. Transform third normal form to fourth normal form.
2. Transform fourth normal form to fifth normal form.



**Figure 15.1: Stages of Normalisation**

In this chapter we shall concentrate on normalisation to third normal form. Those interested in fourth and fifth normal forms are referred to Beynon-Davies (1996).

The process of transforming an unnormalised database into a fully normalised database is frequently referred to as a process of non-loss decomposition (see figure 15.1). This is because we continually fragment our data structure into more and more tables (through relational projects, see section 10.7.2) without losing the fundamental relationships between data-items.

#### 15.4 Representing the Data as an Unnormalised Table

Suppose we are given the task of designing a database for a small tiling supplies company. An analysis of this business leads us to suspect that the following data are the most relevant to the order processing section of the company: orderNo, orderDate, customerNo, customerName, productNo, productName, qty, unitPrice.

Our first task is to build a sample data set for this business in the form of an unnormalised table. This table is illustrated below:

**Orders**

order no.	order date	cust. no.	cust. name	product no.	name	qty	unit price
0/23	01/02/1995	2235	Davies	487	tiles (blue)	200	0.25
0/24	08/02/1995	3444	Jones	488	tiles (white)	300	0.20
0/25	10/02/1995	3444	Jones	489	tiles (red)	150	0.25
0/23	01/02/1995	2235	Davies	340	grout (1kg)	10	1.25
0/24	08/02/1995	3444	Jones	340	grout (1kg)	30	1.25
0/24	08/02/1995	3444	Jones	342	cement (1kg)	30	1.50
0/26	01/02/1995	2237	Evans	488	tiles (white)	400	0.20

#### 15.5 Unnormalised Table to First Normal Form

We transform an unnormalised table to first normal form by identifying repeating groups and turning such repeating groups into separate tables. We first choose a key for this data set. Let us suppose that we choose the data-item orderNo.

Having chosen a key for the unnormalised table we look for a group of data-items that has multiple values for a single value of the key. Examining the table above we see that productNo, productName, qty and unitPrice all repeat with respect to orderNo. We therefore form a separate table of all the repeating data-items, called orderLines and transfer orderNo across as a foreign key.

**Orders**

order no.	order date	customer no.	customer name
0/23	01/02/1995	2235	Davies
0/25	10/02/1995	3444	Jones
0/23	01/02/195	2235	Davies
0/24	08/02/1995	3444	Jones
0/24	08/02/1995	3444	Jones
0/26	01/02/1995	2237	Evans

**OrderLines**

order no.	product no.	product name	qty	unit price
0/23	487	tiles (blue)	200	0.25
0/24	488	tiles (white)	300	0.20
0/25	489	tiles (red)	150	0.25
0/23	340	grout (1kg)	10	1.25
0/24	340	grout (1kg)	30	1.25
0/24	342	cement (1kg)	30	1.50
0/26	488	tiles (white)	400	0.20

**15.6 First Normal Form to Second Normal Form**

To move from first normal form to second normal form we remove part-key dependencies. This involves examining those tables that have a compound key and for each non-key data-item in the table asking the question: can the data-item be uniquely identified by part of the compound key?

Take, for instance, the table named OrderLines. Here we have a 2-part compound key. We ask the question above for the data-items productName, qty and unitPrice. Clearly we need both the orderNo and the productNo to tell us what the qty is likely to be. OrderNo however has no influence on the productName or the unitPrice. This leads to a decomposition of the tables as follows:

**Orders**

order no.	order date	customer no.	customer name
0/23	01/02/1995	2235	Davies
0/25	10/02/1995	3444	Jones
0/23	01/02/195	2235	Davies
0/24	08/02/1995	3444	Jones
0/24	08/02/1995	3444	Jones
0/26	01/02/1995	2237	Evans

**OrderLines**

order no.	product no.	qty
0/23	487	200
0/24	488	300
0/25	489	150
0/23	340	10
0/24	340	30
0/24	342	30
0/26	488	400

**Products**

product no.	product name	unit price
487	tiles (blue)	0.25
488	tiles (white)	0.20
489	tiles (red)	0.25
340	grout (1kg)	1.25
342	cement (1kg)	1.50

**15.7 Second Normal Form to Third Normal Form**

To move from second normal form to third normal form we remove inter-data dependencies. To do this we examine every table and ask of each pair of non-key data-items: is the value of field A dependent on the value of field B, or vice versa? If the answer is yes we split off the relevant data-items into a separate table.

The only place where this is relevant to our present example is in the table called Orders. Here, customerNo determines customerName. We therefore create a separate table to be called Customers with customerNo as the key. This is illustrated below:

**Orders**

Order no.	Order date	Customer no.
0/23	01/02/1995	2235
0/25	10/02/1995	3444
0/23	01/02/1995	2235
0/24	08/02/1995	3444
0/24	08/02/1995	3444
0/26	01/02/1995	2237

**OrderLines**

<b>Order no.</b>	<b>Product no.</b>	<b>qty</b>
0/23	487	200
0/24	488	300
0/25	489	150
0/23	340	10
0/24	340	30
0/24	342	30
0/26	488	400

**Products**

<b>product no.</b>	<b>product name</b>	<b>unit price</b>
487	tiles (blue)	0.25
488	tiles (white)	0.20
489	tiles (red)	0.25
340	grout (1kg)	1.25
342	cement (1kg)	1.50

**Customers**

<b>customer no.</b>	<b>customer name</b>
2235	Davies
3444	Jones
2235	Davies
3444	Jones
3444	Jones
2237	Evans

**15.8 The Bracketing Notation**

To represent the relational schema in an implementation-independent form we use a notation sometimes known as the bracketing notation. We list a suitable mnemonic name for the table first. This is followed by a list of data-items or column names delimited by commas. It is conventional to list the primary key for the table first and underline this data item. If the primary key is made up of two or more attributes, we underline all the component data items. For instance, the third normal form tables for our order-processing example would look as follows:

Orders(orderNo, orderDate, customerNo)  
OrderLines(orderNo, productNo, qty)  
Products(productNo, productName, unitPrice)  
Customers(customerNo, customerName)

### 15.9 The Normalisation Oath

A useful mnemonic for remembering the rationale for normalisation is the distortion of the legal oath presented below:

1. No Repeating,
2. The Fields Depend Upon The Key,
3. The Whole Key,
4. And Nothing But The Key,
5. So Help Me Codd.

Line 5 simply reminds us that E.F. Codd originally developed the techniques in the 1970s. Line 2 states that all data items in a table must depend solely upon the key. Line 1 indicates that there should be no repeating groups of data in a table. Line 3 indicates that there should be no part-key dependencies in a table. Finally, line 4 reminds us that there should be no inter-data dependencies in a table. The only dependency should be between the key and other data-items in a table.

Classic normalisation is described as a process of non-loss decomposition. The decomposition approach starts with one (universal) relation. File maintenance anomalies such as insertion, deletion and update anomalies are gradually removed by a series of projections. Non-loss decomposition is therefore a design process guaranteed to produce a data set free from file-maintenance problems. It does however suffer from a number of disadvantages, particularly as a practical database design technique:

1. It requires all of the data set to be in place before the process can begin.
2. For any reasonably large data set the process is: extremely time-consuming; difficult to apply; prone to human error.

This chapter will therefore concentrate on describing a contrasting approach to normalisation that uses a graphical notation. This makes the technique easier to use and less prone to error.

### 15.10 Determinancy and Dependency

Normalisation is the process of identifying the logical associations between data-

items and designing a database that will represent such associations but without suffering the file maintenance anomalies discussed in section 15.2. The logical associations between data-items that point the database designer in the direction of a good database design are referred to as determinant or dependent relationships. Two data-items, A and B, are said to be in a determinant or dependent relationship if certain values of data-item B always appear with certain values of data-item A.

Determinacy/dependency also implies some direction in the association. If data-item A is the determinant data-item and B the dependent data-item then the direction of the association is from A to B and not vice versa.

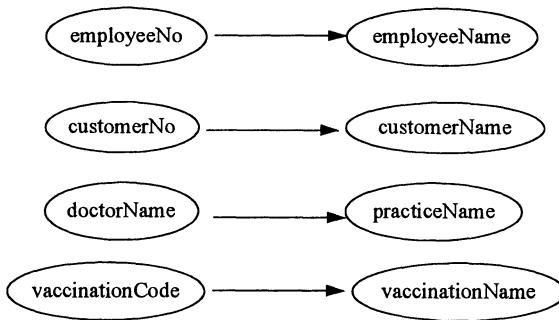
There are two major types of determinacy or its opposite dependency: functional (single-valued) determinacy, and non-functional (multi-valued) determinacy. Most normalisation can be conducted satisfactorily using functional dependencies. We therefore concentrate on functional dependencies in this chapter. Readers interested in non-functional dependencies are referred to Beynon-Davies (1996).

Data-item B is said to be functionally dependent on data-item A if for every value of A there is one, unambiguous value for B. In such a relationship data-item A is referred to as the determinant data-item, while data-item B is referred to as the dependent data-item. Functional determinacy is so called because it is modelled on the idea of a mathematical function. A function is a directed one-to-one mapping between the elements of one set and the elements of another set.

For example, in a personnel database, employeeNo and employeeName would be in a functional determinant relationship. EmployeeNo is the determinant and employeeName is the dependent data-item. This is because for every employee number there is only one associated value of employee name. For example, 7369 may be associated with the value J.Smith. This does not mean to say that we cannot have more than one employee named J.Smith in our organisation. It simply means that each J.Smith will have a different employee number. Hence, although there is a functional determinacy from employee number to employee name the same is not true in the opposite direction – employee name does not functionally determine employee number.

### **15.11 Determinancy Diagrams**

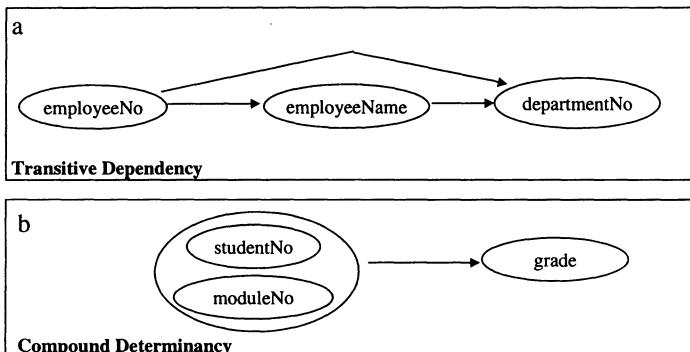
A diagram that documents the determinacy or dependency between data-items we shall refer to as a determinancy or dependency diagram. Data-items are drawn on a determinancy diagram as labelled ovals, circles or bubbles. Functional dependency is represented between two data-items by drawing a single-headed arrow from the determinant data-item to the dependent data-item. For example, figure 15.2 represents a number of functional dependencies as diagrams.



**Figure 15.2: Functional Dependencies as Determinancy Diagrams**

### 15.12 Transitive and Compound Determinancy

Figure 15.3a documents a transitive dependency. A functional dependency exists from manager to department, from department to location, and from manager to location. Any situation in which A determines B, B determines C and A also determines C can usually be simplified into the chain A to B and B to C. Identifying transitive determinancies can frequently simplify complex determinancy diagrams and indeed is an important part of the process of normalisation.



**Figure 15.3: Transitive and Compound Determinancy**

Frequently, one data-item is insufficient to fully determine the values of some other data-item. However, the combination of two or more data-items gives us a dependent relationship. In such situations we call the group of determinants a compound determinant. A compound determinant is drawn as an enclosing bubble around two or more data-item bubbles. Hence, in figure 15.3b we need both **studentNo** and **moduleNo** to functionally determine **grade**. The functional dependency is drawn from the outermost bubble.

### 15.13 Accommodating Functional Dependencies

In this and the following section we examine the process of transforming a determinancy diagram into a set of table structures or relational schema – a process frequently known as accommodation.

Suppose we are given the determinancy diagram in figure 15.4. This diagram documents the dependencies between data-items in the unnormalised table described in section 15.4. We transform the diagram into a set of table structures by applying the rule:

Every functional determinant becomes the primary key of a table. All immediate dependent data-items become non-key attributes of the table.

This is a simplification of what is known as the Boyce-Codd rule after its inventors.

A straightforward way of conducting accommodation with a determinancy diagram is to draw enclosing boundaries around what will fundamentally be table-structures. The dotted boundaries on figure 15.4 indicate the table-structures relevant for this problem.

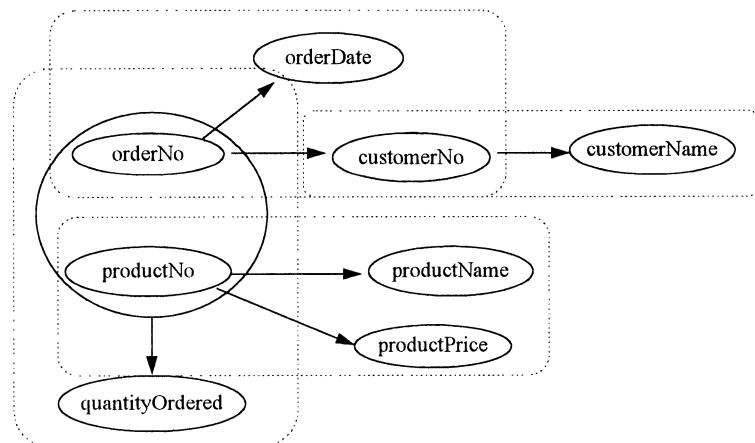


Figure 15.4: A Complete Determinancy Diagram

### 15.14 De-normalisation

Few data analysts would treat the file organisation proposed by third normal form as gospel. In real life, the organisation proposed by third normal form usually represents too many files to be managed practically within the context of a given information system. Many sites would therefore regard first normal form to be good enough, particularly if data volumes are small, or transaction rates are low. Other sites may consider second normal form tables to be sufficiently flexible to meet the demands of higher volumes of data and transactions. Yet

other sites may merge bits of third normal form relations together to optimise processing requirements.

### 15.15 Goronwy Galvanising Case Study

Figure 15.5 represents a determinancy diagram drawn from an analysis of manual documentation such as the document displayed in figure 5.5. Figure 15.6 represents a determinancy diagram drawn from an analysis of the form displayed in figure 5.7. Note the compound determinant in figure 15.6. There are a number of points of communality between the two diagrams. We therefore draw a combined diagram as in figure 15.7.

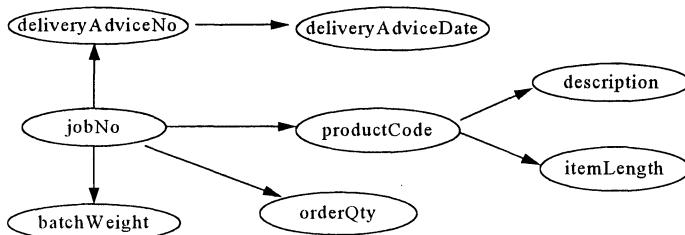


Figure 15.5: Determinacy Diagram for a Delivery Advice

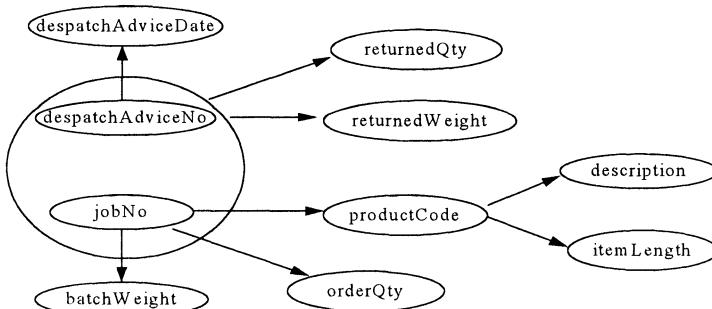
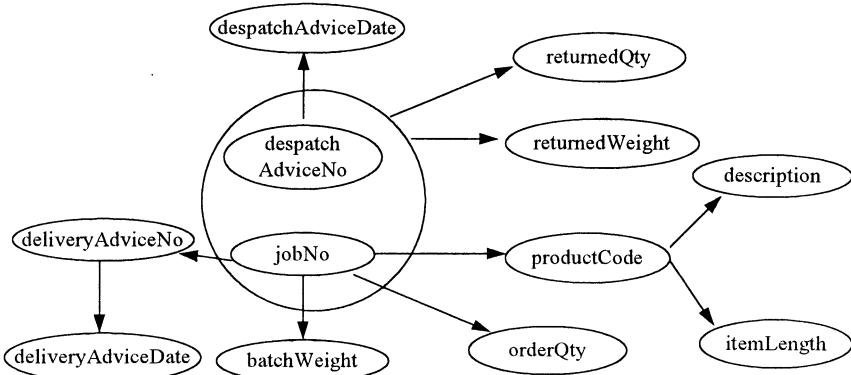


Figure 15.6: Determinacy Diagram for a Despatch Advice

This diagram can then be transformed to the relational schema below by application of the rule discussed in section 15.13. A schema expressed in the bracketing notation is given below:

DespatchAdvices(despatchAdviceNo, despatchAdviceDate)  
DeliveryAdvices(deliveryAdviceNo, deliveryAdviceDate)  
Jobs(jobNo, deliveryAdviceNo, totalWeight, orderQty, productCode)  
Despatches(despatchAdviceNo, jobNo, returnedQty, returnedWeight)  
Products(productCode, description, itemLength)



**Figure 15.7: Determinancy Diagram for a Despatch Advice**

## 15.16 Conclusion

This chapter has considered two distinct approaches to conducting normalisation. The approach, known as non-loss decomposition, is the one mainly discussed in the literature. However, the main advantage of the determinancy diagramming technique is that it provides a mechanism for designing a database incrementally. One does not need a complete data set in hand to begin the process of design. The data analyst can begin his work with a small collection of central data-items and continuously add to this core until the dependencies are fully documented.

### 15.17 References

- Beynon-Davies P. (1996). *Database Systems*. Macmillan Press, Basingstoke.

Codd E.F. (1970). 'A Relational Model for Large Shared Data Banks'. *CACM*, 13 (1), 377-387.

Codd E.F. (1974). 'Recent Investigations into Relational Database Systems'. *Proc. IFIP Congress*.

Fagin R. (1977). 'Multi-Valued Dependencies and a New Normal Form for Relational Databases'. *ACM Trans. on Database Systems*, 2(1).

Fagin R. (1979). 'Normal Forms and Relational Database Operators'. *ACM SIGMOD Int. Symposium on the Management of Data*, 153-160.

### 15.18 Exercises

- Draw a determinancy diagram for each of the following types of marriage: monogamy – one man marries one woman; polygamy – one man can marry many women, but every woman marries a single man; polyandry – one woman can marry many men, but every man marries a single woman; group marriage – one man can marry many women, one woman can marry many men.
- Accommodate each dependency diagram in 1 above into a relational schema.
- Produce third normal form table-structures from the table below:

**Cinemas**

Film No.	Film Name	Cinema Code	Cinema Name	Town	Population	Man. No.	Man. Name	Film Takings
25	Star Wars	BX	Rex	Cardiff	300000	01	Jones	900
25	Star Wars	KT	Rialto	Swansea	200000	03	Thomas	350
25	Star Wars	DJ	Odeon	Newport	250000	01	Jones	800
50	Jaws	BX	Rex	Cardiff	300000	01	Jones	1200
50	Jaws	DJ	Odeon	Newport	250000	01	Jones	400
50	Jaws	TL	Rex	Bridgend	150000	02	Davies	300
50	Jaws	RP	Grand	Bristol	350000	04	Smith	1500
50	Jaws	HF	State	Bristol	350000	04	Smith	1000
30	Star Trek	BX	Rex	Cardiff	300000	01	Jones	850
30	Star Trek	TL	Rex	Bridgend	150000	02	Davies	500
40	ET	KT	Rialto	Swansea	200000	03	Davies	1200
40	ET	RP	Grand	Bristol	350000	04	Smith	2000

- Take any receipt produced by a major supermarket chain and conduct a normalisation exercise.
- The Boyce-Codd rule should actually be phrased as, every functional determinant becomes a candidate key for a relation. In what way does this revision affect the accommodation process?
- Generate a set of fully normalised tables from the following unnormalised table:

**Operating Schedule**

Doctor No.	Doctor Name	Op. No.	opDate	opTime	Patient No.	Patient Name	Admission Date
18654	Smith	AA1234	04/02/1997	08:30	2468	Davies M.	20/01/1997
18654	Smith	BA1598	04/02/1997	10:30	3542	Jones D.	11/01/1997
18654	Smith	FG1965	04/02/1997	16:00	1287	Evans I.	25/12/1997
18654	Smith	AA1235	13/02/1997	14:00	2468	Davies M.	20/01/1997
13855	Evans	LP1564	13/02/1997	14:00	4443	Beynon P.	05/01/1997
18592	Jones	PP9900	15/02/1997	14:00	2222	Scott I.	04/01/1997
18592	Jones	BA1598	04/02/1997	10:30	3542	Jones D.	11/01/1997
18592	Jones	FG1965	04/02/1997	16:00	1287	Evans I.	25/12/1997

7. Goronwy Galvanising employ 30 shift workers. A National Insurance number identifies each employee. This number identifies an employee card on which is written the employee's name, home address, telephone number, date of birth and current weekly salary.

Each employee works a given shift. A unique shift number that documents the start and ending hour of each shift identifies each shift. The current shift number relevant to a given worker is updated on his card each week.

Produce a determinancy diagram for this system. Produce a set of table structures from the determinancy diagram.

# **16 Entity-Relationship Diagramming**

## **16.1 Semantic Data Models**

Database design is fundamentally a task in data modelling. A data model is an architecture for data (chapter 10). Brodie has made a distinction between three generations of data model (Brodie, 1984):

1. *Primitive Data Models.* In this approach objects are represented by record-structures grouped in file-structures. The main operations available are read and write operations over records.
2. *Classic Data Models.* These are the hierarchical, network and relational data models. The hierarchical data model is an extension of the primitive data model discussed above. The network is an extension of the hierarchical approach. The relational data model is a fundamental departure from the hierarchical and network approaches.
3. *Semantic Data Models.* The main problem with classic data models like the relational data model is that they maintain a fundamental record-orientation. In other words, the meaning of the information in the database, its semantics, is not readily apparent from the database itself. The user of databases using the classic approach must consciously apply semantic information. For this reason, a number of so-called semantic data models have been proposed (King and McLeod, 1985). Semantic data models (SDMs) attempt to provide a more expressive means of representing the meaning of information than is available in the classic models.

Probably the most frequently cited of the SDMs is the Entity-Relationship data model (E-R model) (Chen, 1976). In the E-R model the ‘real world’ is represented in terms of entities, the relationships between entities and the attributes associated with entities. Entities represent objects of interest in the real world such as employees, departments and projects. Relationships represent named associations between entities. A department employs many employees. An employee is assigned to a number of projects. Employs and is assigned to are both relationships in the Entity-Relationship approach. Attributes are properties of an entity or relationship. Name is an attribute of the employee entity. Duration of employment is an attribute of the employs relationship.

## **16.2 Entity Models**

An entity model is a model of the entities, relationships and attributes in some domain of discourse. An entity may be defined as ‘a thing which the enterprise

recognises as being capable of an independent existence and which can be uniquely identified' (Howe, 1986). A relationship can be defined as 'an association between entities' (Beynon-Davies, 1992). An attribute is a property of some entity.

### **16.2.1 Entities**

An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a transaction or order. Although the term entity is the one most commonly used following Chen (1976) we should really distinguish between an entity and an entity-type. An entity-type is a category. An entity, strictly speaking, is an instance of a given entity-type. There are usually many instances of entity-types.

Because the term entity-type is somewhat cumbersome, most people tend to use the term entity as a synonym. We shall conform to this practice. It must be remembered however that whenever we refer to an entity we mean an entity-type.

### **16.2.2 Relationships**

More than one relationship can exist between any two entities. For instance, the entities house and person can be related by ownership and/or by occupation. In theory, having identified a set of say 6 entities, up to 15 relationships could exist between these entities. In practice, it will usually be quite obvious that many entities are quite unrelated. Furthermore, the object of entity modelling is to document only so-called direct relationships. For instance, direct relationships exist between the entities Parent and Child and between Child and School. The relationship between Parent and School is indirect; it exists only by virtue of the child entity (Shave, 1981).

### **16.2.3 Attributes**

An entity is an aggregation of attributes. Values assigned to attributes are used to distinguish one entity from another. Hence, deptNo, deptName, and location are all attributes that characterise the Department entity. One or more of the attributes of an entity are normally chosen as an entity identifier. The attribute deptNo is a suitable identifier for the entity Department.

## **16.3 Semantic Modelling**

The data needed to support a given information system does not usually fall irrevocably into one of the three categories: entity, relationship and attribute. A classic example is the data needed to be stored on marriages. Marriage could be regarded as an entity with attributes such as date, place, and names of bride and

groom. It could similarly be regarded as the attribute marital status associated with the entity Person. Finally, it could be represented as a relationship between the entities Man and Woman.

One of the tasks of the entity modeller is to decide which of these viewpoints is the most important for the information system under consideration. Hence, data analysis is frequently referred to as semantic modelling (Date, 1990). The aim is to represent data as it is perceived in the organisation under consideration (Klein and Hirschheim, 1987).

#### 16.4 Notation

Entity models are usually mapped out as entity-relationship diagrams (E-R diagrams). The end product of entity-relationship diagramming is a model of the entities and relationships in a particular domain of discourse.

An entity is represented on the diagram by a rectangular box in which is written a meaningful name for the entity (figure 16.1a). A relationship between entities is represented by drawing a line (sometimes labelled) between the relevant boxes on the diagram (figure 16.1b). An attribute is represented by a circle or oval attached by a line to the appropriate entity. The entity identifier is underlined (figure 16.1c).

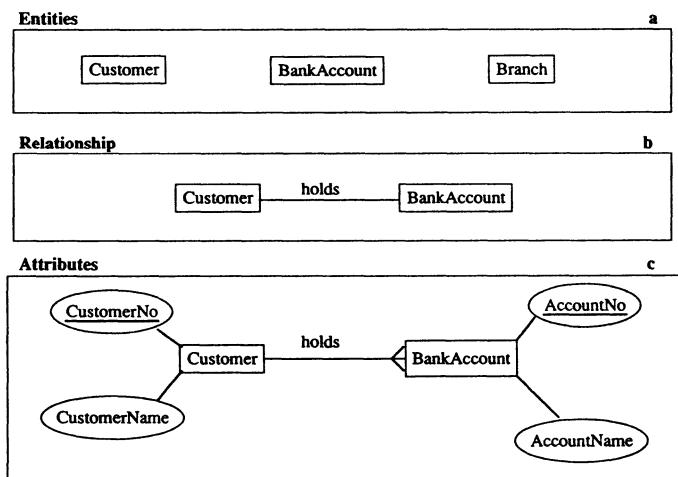


Figure 16.1: Entities, Relationships and Attributes

### **16.3.1 Properties of a Relationship**

There are two properties of the concept of a relationship that are usually considered important: we shall refer to them as cardinality and participation.

Cardinality (or degree) concerns the number of instances involved in a relationship. A relationship can be said to be a 1:1 (one-to-one) relationship, a 1:M (one-to-many) relationship, or a M:N (many-to-many) relationship.

For instance, the relationship between bankaccounts and customers can be said to be one-to-one (1:1) if it can be defined in the following way:

A bankaccount is held by at most one customer.

A customer may hold at most, one bankaccount.

In contrast, the relationship between bankaccounts and customers is one-to-many (1:M) if it is defined as:

A customer holds many bankaccounts.

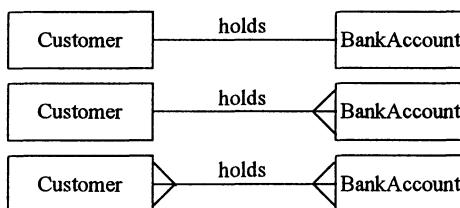
A bankaccount is held by at most one customer.

Finally, we are approaching a realistic representation of the relationship when we describe it as being many-to-many (M:N). That is:

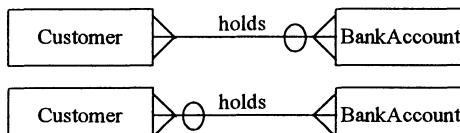
A customer holds many bankaccounts.

A bankaccount may be held by many customers.

#### **Degree/Cardinality**



#### **Optionality/Participation**



**Figure 16.2: Properties of a Relationship**

There are a number of competing notational devices available for portraying the cardinality of a relationship. We choose to represent cardinality by drawing a crow's foot on the many end of a relationship (see figure 16.2a).

Participation (or optionality) concerns the involvement of entities in a relationship. An entity's participation is optional if there is at least one instance of an entity that does not participate in the relationship. An entity's participation is mandatory if all instances of an entity must participate in the relationship. The default participation is mandatory. If the participation is optional we add a circle (an 'O' for optional) alongside the relevant entity (see figure 16.2b).

#### 16.4 Abstraction Mechanisms

The original Entity-Relationship model has been extended in a number of ways (Teorey *et al.*, 1986). One of the most important extensions is the support for generalisation hierarchies (Smith and Smith, 1977). This allows us to declare certain entities as subtypes of other entities. For instance, Manager, Secretary and Technician might all be declared subtypes of an Employee entity. Likewise, SalesManager, ProductionManager, etc., would all be declared subtypes of the Manager entity. The important consequence of this facility is that entities lower down in the generalisation hierarchy inherit the attributes and relationships of entities higher up in the hierarchy. Hence, a sales manager would inherit properties of managers in general, and indeed of employees in general. Likewise, a sales manager would inherit the relationship of an employee to a department.

In relatively simple cases it is convenient to indicate generalisation on an E-R diagram by drawing disjoint or overlapping boxes (see figure 16.3a) (Harel, 1986). As the size of a diagram increases it is more practical to use a notation built on relationship lines. In figure 16.3b we have represented generalisation by an empty triangle (Rumbaugh *et al.*, 1991).

The Abstraction mechanisms of aggregation and association can also be represented on E-R diagrams.

Aggregation is the process by which a higher-level object is used to group together a number of lower-level objects. For instance, ISBN, title, author, publisher and dateOfPublication may all be aggregated together to form a Book entity. In the Entity-Relationship model, therefore, aggregation is implicit in assigning attributes to an entity. Aggregation of entities can also appear on an entity model. Hence, for instance, a Car entity might be built up of an assembly of wheels, chassis, engine, etc. Assemblies of this kind can be represented by a forked link from the aggregate entity to the part entities and a semi-circle at the junction of relationship lines (see figure 16.4).

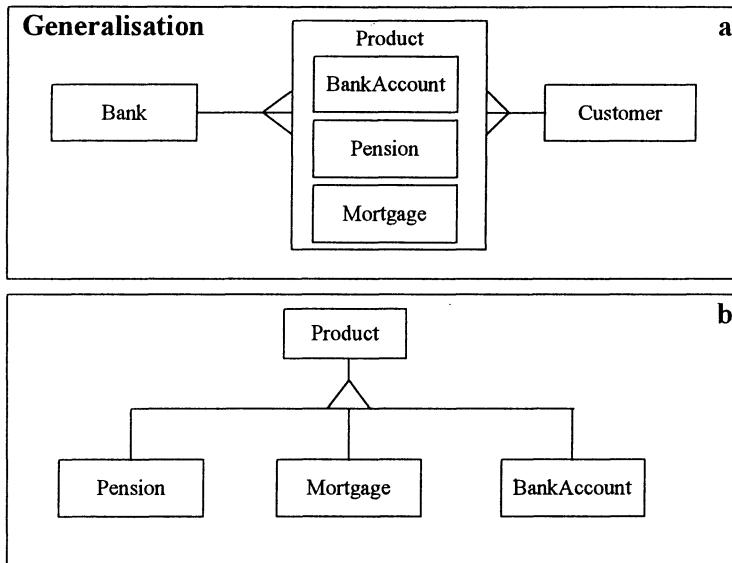


Figure 16.3: Generalisation Notations

Association is a form of abstraction in which instances of one entity are associated with instances of another entity. Association is, of course, implicit in the way we define relationships on entity models.

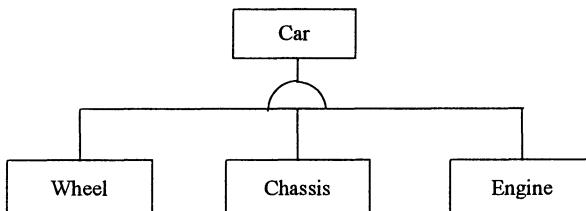


Figure 16.4: Aggregation

## 16.5 Validating an Entity Model against Requirements

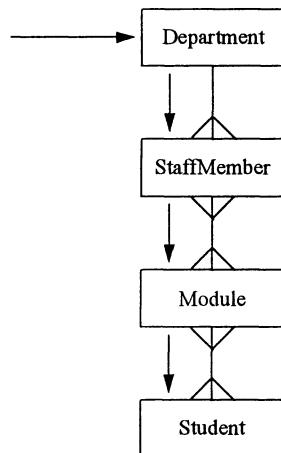
A first-pass E-R diagram represents the basic structure of data needed in a given information system. Most of the adherents of the technique recommend that an entity model should be validated against some definition of processing requirements. This definition will identify which entities and relationships must be accessed, in what order, by what means, and for what purpose. A detailed discussion of this topic is given in Beynon-Davies (1992).

Suppose we have the following extract from an E-R diagram representing an

educational application (figure 16.5) (Shave, 1981). We wish to validate this entity model against a requirement to produce the staff/student ratio for a given department. To perform this processing we need to access:

1. All staff of a department.
2. Each course taught by a member of staff.
3. Each student registered for a course.

This is indicated on the E-R diagram in figure 16.5 as a series of arrows.



**Figure 16.5: Validating an Entity Model**

## 16.6 Producing a Relational Schema

E-R diagrams are generally drawn as a means of designing relational schema. The process of transforming an E-R diagram into a relational database design we refer to as accommodation. The process of accommodating E-R diagrams involves the following steps:

1. Break down each many-to-many relationship into two one-to-many relationships. This is done by interposing a link entity between the two previous entities and drawing crow's-feet to the link entity. The entity identifiers for the two previous entities then become a compound entity identifier for the link entity.
2. For each entity on our diagram we form a table, usually by making the entity names plural.
3. The identifying attribute of each entity becomes the primary key of the table.
4. All other attributes of the entity become non-key attributes of the table.

5. For each one-to-many relationship post the primary key from the one end of the relationship into the table on the many end of the relationship.
6. Optionality on the many end of a relationship tells us whether a foreign key can be null or not. If the many end is mandatory the foreign key is not null. If the many end is optional the foreign key can be null.

Taking the educational application we transform all many-to-many relationships to give us figure 16.6. Note, we have added some sample attributes to the diagram. A skeleton relational schema expressed in the bracketing notation is given below:

Departments(deptNo, deptName, ...)  
 StaffMembers(StaffNo, staffName, deptNo, ...)  
 Modules(moduleNo, moduleName, ...)  
 Students(studentNo, studentName, ...)  
 Allocation(staffNo, moduleNo, ...)  
 Registration(studentNo, moduleNo, ...)

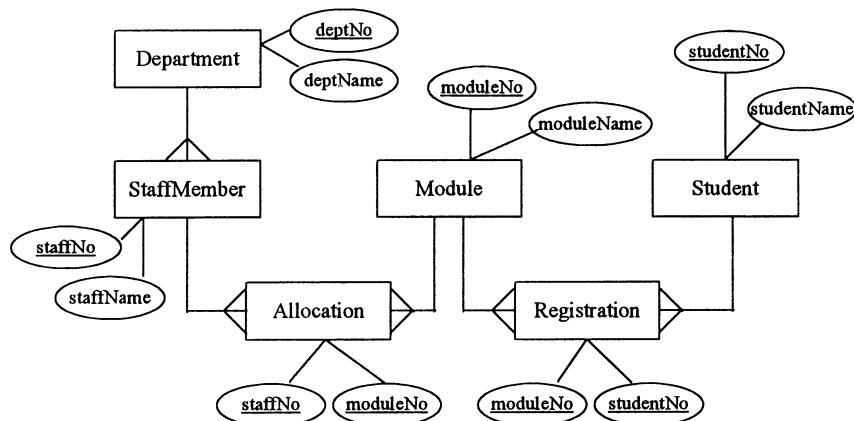


Figure 16.6: Educational E-R Model

## 16.7 Case Study: A General Hospital Appointments and Operations System

Suppose we are given the brief of designing an appropriate information system for the patients' appointments and operations activities of a large general hospital. Our initial analysis work provides us with the following brief description of the existing manual system:

1. Patients must make an appointment at a given clinic session held at one of the hospital's clinics.

2. Doctors are allocated one or more appointments within a clinic session, but only one doctor will be present at each appointment.
3. Operations are scheduled and allocated to one of a number of theatre sessions held in the hospital's operating theatres. Each doctor may perform a number of given operations on patients. A given operation is done on only one patient, but there may be more than one doctor in attendance.

From such a description, we get some initial idea of the important entities involved in this system:

Clinic  
ClinicSession  
Appointment  
Patient  
Doctor  
Operation  
OperatingTheatre  
TheatreSession

Note that we might have been tempted to add the entity Hospital to this list. However, in this system there is only one instance of a hospital. Entities should normally have many instances associated with them.

The text also gives some idea of relationships. A possible list is given below:

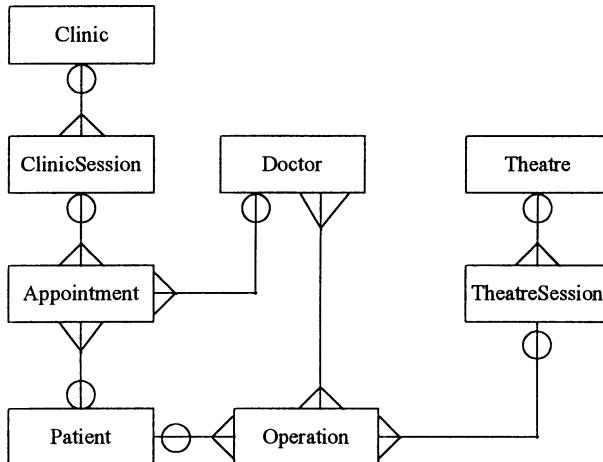
Patient – Operation  
Doctor – Operation  
Doctor – Appointment  
Patient – Appointment  
ClinicSession – Appointment  
Clinic – ClinicSession  
TheatreSession – Operation  
Theatre – TheatreSession

Each of these relationships then needs to be characterised in terms of cardinality and participation. For some relationships the text above does not provide enough information. We therefore make a number of assumptions that would have to be confirmed or otherwise by users. A possible diagram is given in figure 16.7.

From this E-R diagram we arrive at the following relational schema by applying the guidelines discussed in section 16.6.

Clinics(clinicName, ....)  
 ClinicSessions(clinicName, sessionNo, ...)  
 Appointments(clinicName, sessionNo, patientNo, doctorNo, ....)  
 Patients(patientNo, ...)  
 Doctors(doctorNo, ...)  
 Operations(theatreName, theatreSession, operationNo, patientNo, ...)  
 Schedule(doctorNo, operationNo, ...)  
 Theatres(theatreName, ...)  
 TheatreSessions(theatreName, theatreSession, ...)

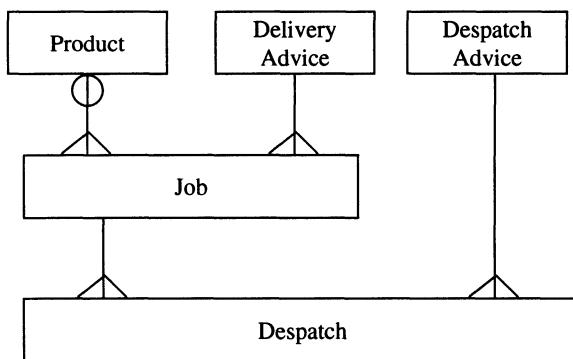
Note how we have formed compound keys in a number of tables, where the foreign keys form part of the primary key. Note also how the table schedule constitutes a link entity between doctors and operations.



**Figure 16.7: Appointments and Operations Model**

## 16.8 Goronwy Galvanising Case Study

Figure 16.8 illustrates an E-R diagram drawn for the case of the Goronwy Galvanising system. Note how we have distinguished between delivery advice notes and despatch advice notes. The entity despatch is a breakdown of the many-to-many relationship between Job and DespatchAdvice. In other words, a given job can be recorded on more than one despatch advice note.



**Figure 16.8: Entity Model for Goronwy**

Applying the rules of accommodation as described in section 16.6 would mean that we arrive at the following set of skeleton table structures:

DespatchAdvices(despatchAdviceNo, ...)  
 DeliveryAdvices(deliveryAdviceNo, ...)  
 Jobs(jobNo, despatchAdviceNo, productCode, ...)  
 Products(productCode, ...)  
 Despatches(despatchAdviceNo, jobNo, ...)

Fortunately, this relational schema matches closely with the one generated from the determinancy diagrams in chapter 10. In many circumstances this will not be the case. The analyst frequently has to reconcile the results from top-down data analysis with the results from bottom-up data analysis. See Beynon-Davies (1996) for a more detailed discussion of reconciliation. Figure 16.9 indicates how we might modify the E-R diagram in figure 16.8 to exploit generalisation. Here we have made delivery advices and despatch advices both subtypes of an advice class.

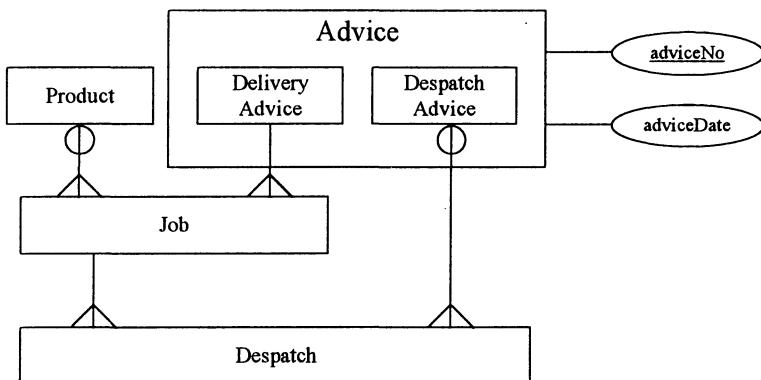


Figure 16.9: Generalisation Applied to Goronwy

## 16.9 Conclusion

Entity modelling as represented by the technique of E-R diagramming is primarily a database design technique. An E-R diagram is conventionally referred to as a conceptual model of a database system. From this conceptual model we derive a logical model expressed usually as a series of table structures in third normal form. The final step in database development is to produce a physical model. That is, a series of record structures expressed in the syntax of some programming language or DBMS. If the DBMS is relational this step is relatively straightforward. If physical structures are COBOL files then the process is a little more convoluted but not difficult (Howe, 1986).

## 16.10 References

- Beynon-Davies P. (1996). *Database Systems*. Macmillan Press, Basingstoke.
- Brodie M.L. (1984) 'On the Development of Data Models'. In Brodie M.L., Mylopoulos J. and Schmidt T.W. (Eds) (1984) *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*. Springer-Verlag, Berlin.
- Chen P.P.S. (1976). 'The Entity-Relationship Model – Toward a Unified View of Data'. *ACM Trans. on Database Systems*. 1. 9-36.
- Date C.J. (1990) *An Introduction to Database Systems*. Vol. 1. 5th Ed. Addison-Wesley, Reading, Mass.
- Harel D. (1986) 'On Visual Formalisms'. *CACM*. 31(5) May 514-529.
- Howe D.R. (1986). *Data Analysis for Database Design*. (2nd Ed.). Edward Arnold, London.
- King R. and McCleod D. (1985) Semantic Data Models. In Bing Yao S. (Ed.).

- Principles of Database Design. Vol 1: Logical Organisations.* Prentice Hall, Englewood Cliffs. N.J.
- Klein H.K. and Hirschheim R.A. (1989). 'Four Paradigms of Information Systems Development'. *CACM*. 32(10) October 1199-1216.
- Shave M.J.R. (1981). 'Entities, Functions and Binary Relations: steps to a conceptual schema'. *The Computer Journal*. 24(1).
- Smith J.M. and Smith D.C.P. (1977). 'Database Abstractions: Aggregation and Generalisation'. *ACM Trans. Database Sys.* 2(2) 105-133.
- Teorey T.J. Yang D., and Fry J.P. (1986). 'A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model'. *ACM Computing Surveys*. 18 197-222.

### 16.11 Exercises

1. Produce an E-R diagram that represents the following information describing a horses breeding register: A racing horse is identified by a unique name. The date of birth of the horse and its sex are also recorded. Each horse has a father and mother. The system must be able to produce a genealogy for each horse.
2. Produce a set of table structures from the E-R diagram produced for 1.
3. Draw an extended E-R diagram to represent the following classification problem: Lions and Tigers are big cats; BigCats are Mammals; Mammals are Animals.
4. Draw an E-R diagram to represent the following assertions: An Employee uses a CompanyCar; a given CompanyCar will be used by a number of different employees; some employees do not use any company car; all company cars are used by at least one employee.
5. Why do you think the accommodation process to COBOL file structures is more involved than that for relational systems?
6. Produce an E-R diagram to handle the data on a supermarket bill or a gas bill.

## ***Part Three Section Two***

### ***Process Analysis***

Process analysis takes a dynamic view of information systems. Process analysis techniques concentrate on the movement of data through systems and the transformation of data within systems. Process analysis and data analysis are necessarily interdependent. The structure of data determines the functions that can be performed with data and vice versa. An information system made up solely of data structures is inconceivable. Most information systems are built to provide some means for data manipulation.

Whereas a data model is an abstraction of the data structures required by some application, a process model is an abstraction of the processes required to be supported by some application. To this end, a number of techniques have been developed of which the most influential is undoubtedly data flow diagramming. Two other techniques are also discussed which would normally be used to support a data flow diagram: process descriptions and data dictionaries. Entity life histories are a useful process-oriented adjunct to entity modelling, and it is for this reason that we include a discussion of the technique here.

# **17 Data Flow Diagramming**

## **17.1 Introduction**

A data flow diagram (DFD) is a representation of a system or subsystem in terms of how data moves through the system.

A number of people have been involved in developing the data flow diagram as an analysis and design tool. Among the earliest exponents of the method were Tom DeMarco and Edward Yourdon (DeMarco, 1979). Gane and Sarson have modified and extended the technique to approach something like the technique discussed in this chapter (Gane and Sarson, 1977). The major difference of the technique used here is in the notation. We shall be using the notation recommended within the British methodology SSADM (Structured Systems Analysis and Design Methodology) (Cutts, 1991).

## **17.2 The Plumbing Analogy**

Probably the easiest way to understand the rationale behind a data flow diagram is to make the analogy between an information system and a household plumbing system. Plumbing systems are designed to handle flows of water. Information systems are designed to handle flows of data. A plumbing system receives its water from external sources such as the public water supply, and deposits its used water in external entities such as drains. An information system receives its data from external sources such as customers, banks, retailers, etc., and communicates the results of its processing to other entities, perhaps other information systems. Household plumbing systems are usually designed to process water in some way. For instance, a boiler engages in the process of heating the water to a given temperature. In information systems far more various forms of processing occur; data is transformed by some process and then passed on to another process, and so on. Finally, in a plumbing system there are usually repositories of water, e.g. sinks, cisterns etc. In information systems, such repositories are referred to as data stores.

## **17.3 Elements**

DFDs are hence made up of four basic elements: processes, data flows, data stores and external entities (sometimes called sources or sinks).

### **17.3.1 Processes**

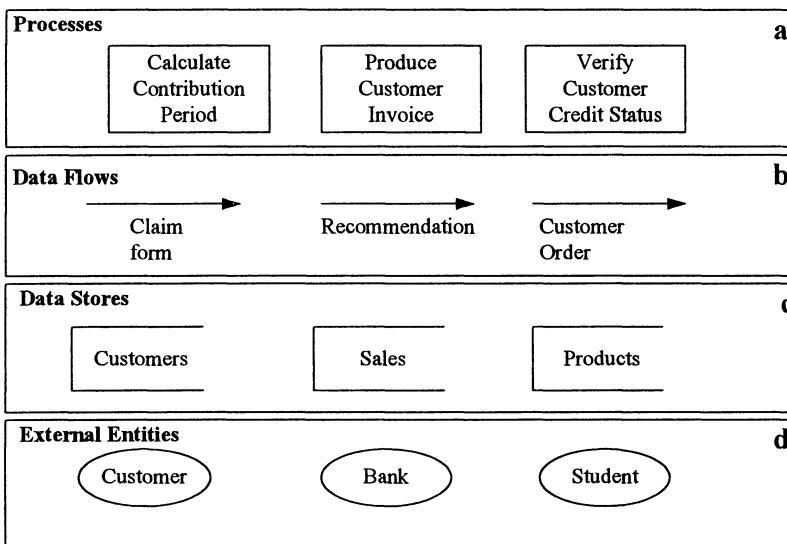
A process is a transformation of incoming data flow(s) into outgoing data flow(s). A process is represented on a DFD by a labelled square or rectangle, as in figure 17.1a.

### **17.3.2 Data Flows**

A data flow is a pipeline through which packets of data of known composition flow. Data flow is represented on a DFD by a labelled directed arrow, as in figure 17.1b.

### **17.3.3 Data Stores**

A data store is a repository of data. For example, a waste-paper basket, a register, a card index, an indexed-sequential file. A data store is represented on a DFD by an open rectangle or box with an appropriate label, as in figure 17.1c.



**Figure 17.1: DFD Notation**

### **17.3.4 External Entities**

An external entity (also called a source or sink) is something (usually a person, department or organisation), lying outside the context of the system, that is a net originator or receiver of system data. It is represented on a DFD by some form of rounded shape – circle or oval – with an appropriate name, as in figure 17.1d.

Note, external entities on DFDs should not be confused with entities on E-R diagrams.

## 17.4 Conventions

A number of conventions are normally applied to each of these constructs:

### 17.4.1 Processes

1. No two processes should have the same name. This is really to enable processes to form unique elements in a data dictionary (see chapter 19).
2. Each process should precisely state a transformation. Labels for processes hence usually include some form of verb, such as verify, store, record, etc.
3. A process is subject to the ‘conservation of data’ principle: a process cannot create new data. New data can only come from external entities. A process may only take its input data and transform it in some way to make output data.

### 17.4.2 Data Flow

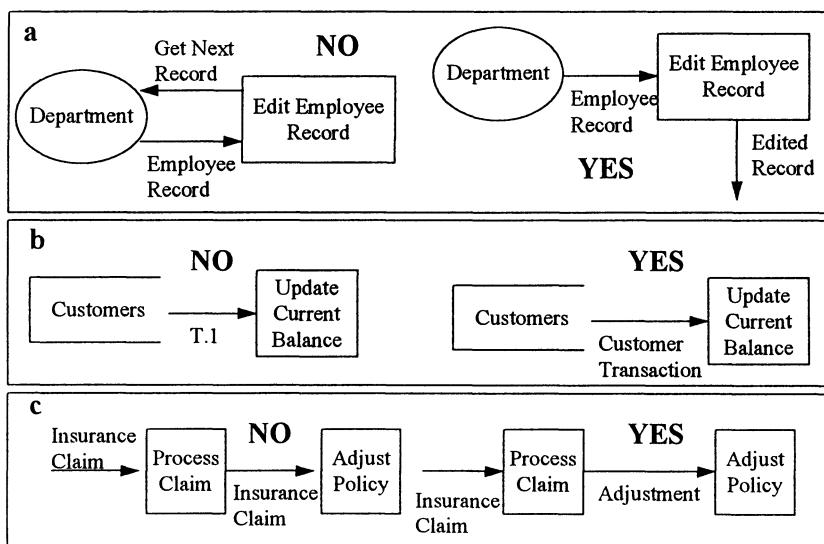


Figure 17.2: Data Flow Conventions

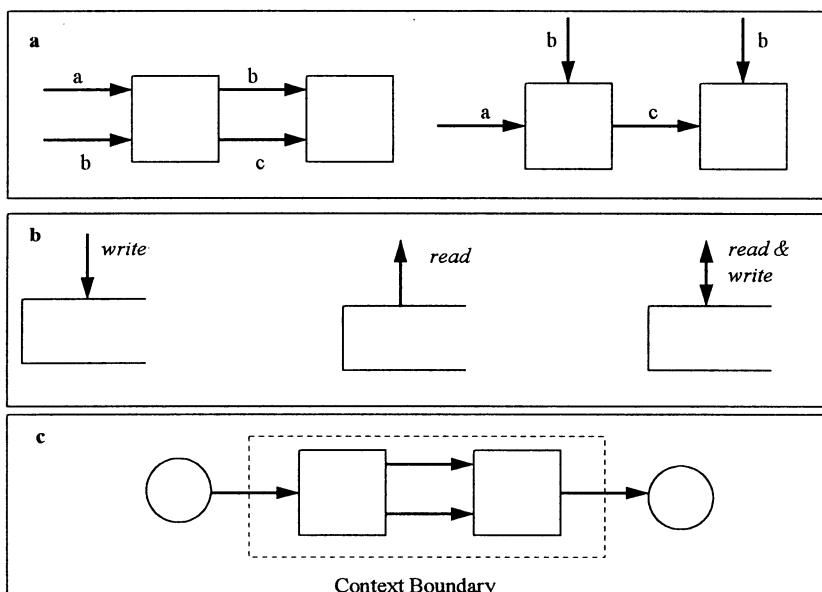
1. A data flow is not a representation of the flow of control (see figure 17.2a).
2. Every data flow should have an associated meaningful label (figure 17.2b).
3. No two flows should have the same name. Input data from a process should

not appear as output data from that process. If it does, then it has not been transformed (see figure 17.2c).

4. If data flows pass through processes to get to other processes then we redraw that part of the diagram so that data is shown as direct input to the processes that use it (see figure 17.3a).

#### **17.4.3 Data Stores**

1. A data store can be read, written to, or read and written to (see figure 17.3b).
2. Data stores are conventionally labelled with plural nouns to emphasise the multiple instances of information contained within the data store. A consistent naming convention is normally applied. In other words, we do not name a store customers on one diagram and customer details on another diagram.
3. A data store is subject to the ‘conservation of data’ principle. What comes out of a data store must first go into the data store. It is not possible for a data store to create new data.



**Figure 17.3: More DFD Conventions**

#### **17.4.4 External Entities**

1. An external entity lies outside the context of an information system. Hence, this construct is used primarily to define system boundaries – a system’s interface to the external world (see figure 17.3c).

- It is conventional to label external entities with a singular noun. We talk of a bank, not of banks, of an employee, not of employees.

### 17.5 Additions to the Basic Notation

When conducting the analysis of information systems it has been found useful to add the following constructs to the basic DFD notation:

- Materials or Goods Flow.** In looking for a place to start, systems analysts often concentrate initially on the physical movement of materials through systems. Such movement can be documented on a DFD as a broad, labelled arrow (see figure 17.4a). It is conventional to draw such materials flow as coming from an entity or process, and going to an entity or process. No connection is made to data stores.
- Documents Flow.** One of the most important requirements of early analysis is to make sure that all relevant documents such as invoices and order forms that apply in a manual information system are collected, examined and analysed. It is therefore useful to record such documents by name on a DFD. This is done by drawing a traditional document symbol on a data flow (figure 17.4b).
- Boolean Constructs.** Sometimes it is useful to specify the operators AND and OR on DFDs. Two flows with a ‘+’ between them are inextricably linked in the sense that both must be input to a process or output from a process at the same time. A ‘\*’ between flows means that either one flow occurs or another flow, but not both (see figure 17.4c).

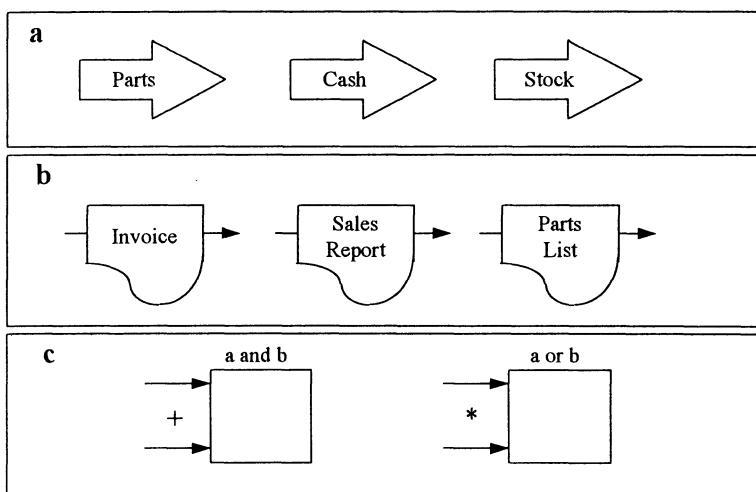


Figure 17.4: Additions to the Basic Notation

## 17.6 Levelling

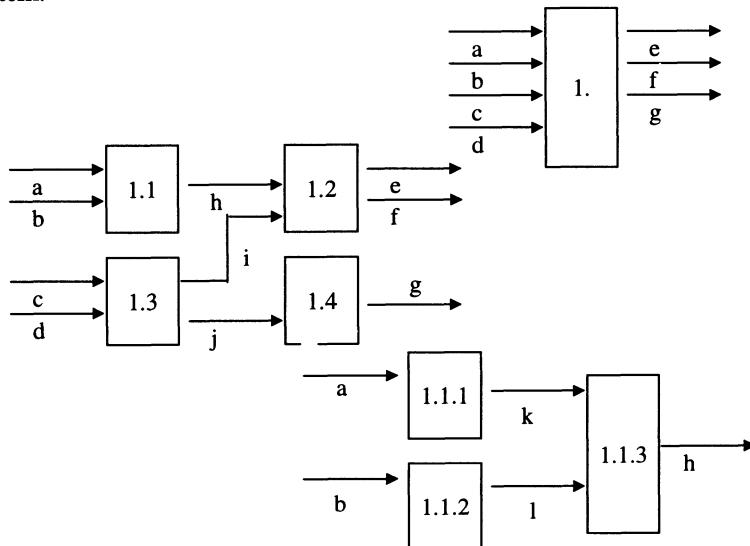
Most real-life systems are too involved to represent as a DFD on one single sheet of paper. In representing systems, analysts therefore usually approach the problem in a top-down manner. They attempt to level the problem beginning with an overview diagram of the entire system. As a minimum this might be made up of one process with associated flows, stores and entities. They then take the process or processes represented on the overview DFD and break them down into their own DFDs. They may continue this decomposition process for a number of levels until they can represent the entire system in sufficient detail. A rough guideline is that an appropriately partitioned system is one in which no individual diagram has more than nine processes represented upon it (Miller, 1967).

Given that our system of DFDs represents a hierarchically organised documentation system, a number of conventions are normally applied to the construction of such a system (see figure 17.5).

First, each DFD is headed with the name of its parent process. In the case of the overview diagram this will of course represent the title of the system. For all other DFDs it will refer to the master process which is exploded in the present DFD.

Second, all inputs and outputs between parent and child diagrams are balanced. If flows A and B input to process 1 and flow C outputs, then the child diagram should also detail flows A, B and C.

Third, to indicate the position in the DFD hierarchy of any particular process it is found useful to number each process uniquely within a documentation system.



**Figure 17.5: Levelling**

## 17.7 Constructing a DFD

Some suggested guidelines for constructing a DFD hierarchy are given below. We illustrate the production of a DFD set to describe a simple manual information system.

Start with a name which adequately describes the information system, for example, Video Shop System.

Identify the main external entities that will be used to define the boundaries of the information system, for example, member, supplier, etc. Place a single process box with a label for the system in the centre of a page and arrange the external entities around it. Consider the major flows from the external entities to the central process. This context diagram (illustrated in figure 17.6) should provide an accurate picture of the essence of the information system.

Identify the major activities of the information system. For example: Creating a Member; Issuing a Video; Returning a Video; Reserving a Video; Customer Enquiries; Acquiring New Videos. Draw an overview diagram to represent the interaction of the major activities of the system. Represent each activity as a process box. Transfer the external entities and flows on the context diagram on to the overview diagram. Consider the internal flows between processes (figure 17.7).

Take each of the major activities on the overview diagram and identify the sub-activities. For example:

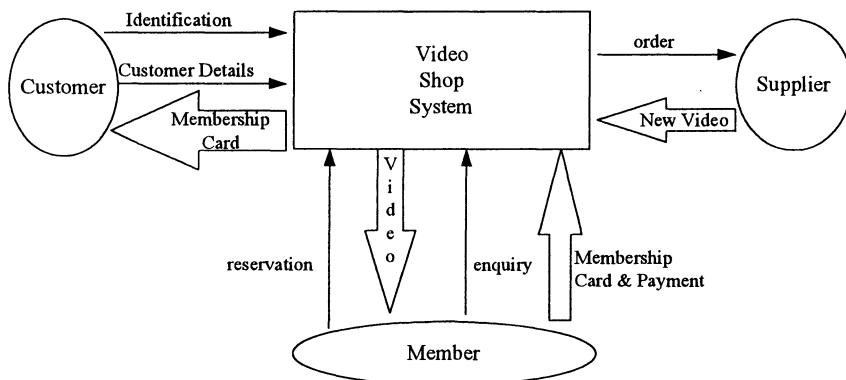


Figure 17.6: Video Shop Context Diagram

### Creating a Member

- Requesting Identification
- Recording Membership Details
- Producing a Membership Card

### Issuing a Video

- Recording Hire Details
- Retrieving the Video

- Requesting Payment
- Handing Over the Video
- Returning a Video
  - Receiving the Video
  - Checking for Reserved Status
  - Checking for Overdue Status
  - Erasing Hire Details
- Reserving a Video
  - Extracting Video Details
  - Making a Reservation Mark
- Customer Enquiries
  - Identifying Purpose of Query
  - Answering Query
- Acquiring New Videos
  - Ordering a New Video
  - Receiving a New Video

Take each of the main activities and draw a DFD to represent it. Ensure that any relevant flows, entities and stores on the overview diagram are duly represented on the first level diagram. Figure 17.8 illustrates a first-level DFD drawn from the creating a member process.

Iterate up and down between the various levels of the DFD. If we add extra process boxes and flows at lower levels, consider the effect on higher levels.

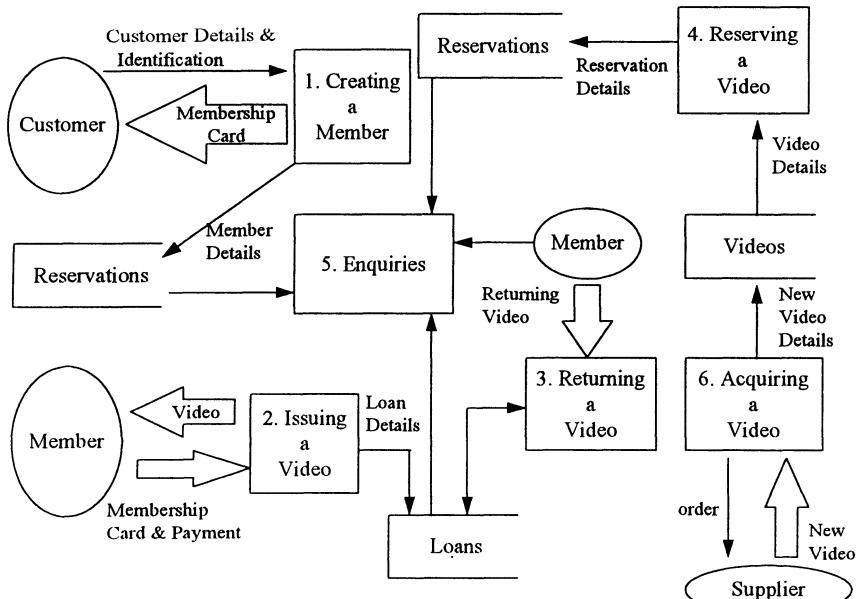


Figure 17.7: Video Shop Context Diagram

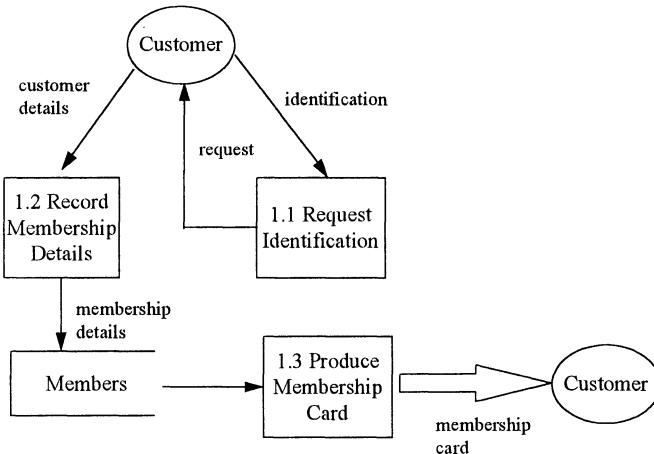


Figure 17.8: First-Level DFD

## 17.8 Case Study: the Fast Foods System

Fast Foods is a business serving small retailers. Such retailers order canned foods and packaged items such as sugar, tea and rice from the company in bulk quantities. Fast Foods holds no stock, but orders in bulk from wholesalers at correspondingly discounted prices. When the bulk supplies arrive, the retailer's orders are filled and despatched.

Before orders are processed, retailers are checked for credit status. If such credit status is unacceptable, the retailer is requested to send cash with the order. This usually means that Fast Foods puts the order on hold while the customer is requested to make prepayment.

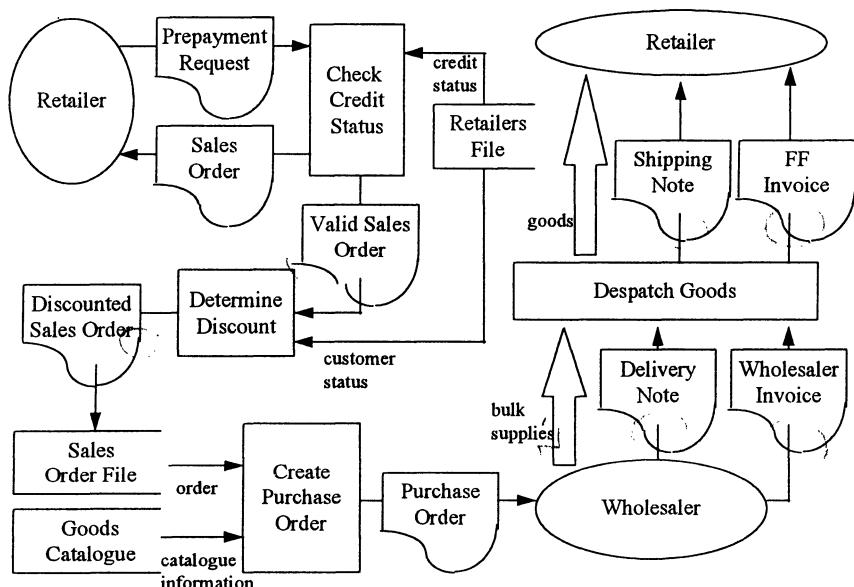
Fast Foods operate a discount policy. Such discounts are determined by the total value of the order and the status of the retailer, particularly whether or not they pay their bills promptly.

Retailers order goods by detailing manufacturer and product name or by specifying a standard product code, which identifies the manufacturer as well as the product. Such information is written on a standard sales order by the retailer and sent to Fast Foods. The company distributes a catalogue of their products to retailers, together with a set of order forms on a regular basis.

The main documents used in the system are:

1. A sales order from the retailer
2. A purchase order to the wholesaler
3. A delivery note to the wholesaler
4. A shipping note to the retailer
5. Invoices to the retailer and wholesaler
6. Remittance advices from the retailer to the wholesaler

An overview DFD representing this manual system is illustrated in figure 17.9. A number of assumptions have been made in drawing this DFD. These are necessary in the sense that the narrative description detailed above does not provide enough information to produce a sensible solution to the problem. Most of the assumptions made incorporate background knowledge or commonsensical reasoning. For instance, there is nothing in the text to indicate how a purchase order is produced from sales orders. It seems sensible therefore to indicate that some form of batching activity occurs. Sales orders are accumulated until a satisfactory bulk order for a particular product is produced.



**Figure 17.9: Fast Foods - Current Physical DFD**

Drawing a DFD before all the information is known is a perfectly valid exercise. In this sense, we are using the DFD as an analysis tool. We are making a first attempt at understanding the problem by constructing an initial description. This exercise will generate many questions that we will need to ask our users. The diagram is then redrawn, taken back to the users, and the whole process repeated until we are happy that our diagrams accurately specify the workings of the current system.

## 17.9 Logical and Physical Systems

The diagram in figure 17.9 is usually referred to as a model of the current physical system. It is a high-level description of how the current manual system

at Fast Foods works. In contrast, figure 17.10 is a model of the current logical system. In this figure we have abstracted out as much of the implementation detail as possible. In particular, we have removed references to goods and documents flow. We are left with a description of abstract data flow.

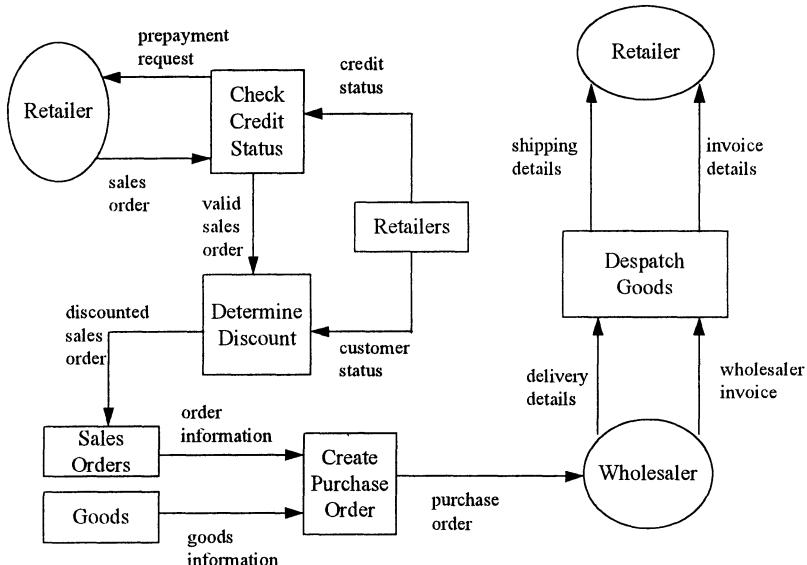


Figure 17.10: Fast Foods - Current Logical DFD

DFDs are therefore normally used in at least four different ways in structured systems analysis and design:

1. To represent the current physical system.
2. To represent the current logical system.
3. To represent the proposed logical system – the current logical system plus new requirements.
4. The proposed physical system – a detailed representation of the workings of the proposed system.

## 17.10 Goronwy Galvanising Case Study

Figure 17.11 illustrates how we might draw a context DFD for the existing Goronwy system. Note how we have included materials flow to help the reader see at-a-glance the primary function of the information system – to manage the flow of goods.

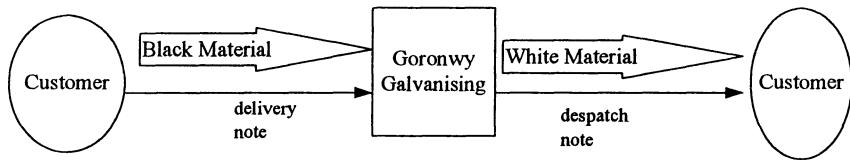


Figure 17.11: Context DFD - Goronwy

Figure 17.12 represents an overview diagram for the manual system. We identify four major processes: delivery, job-sheet creation, updating job-sheets, and despatch. Note that galvanising is really a physical process. For the sake of brevity we have included both materials and data flow into and from this process.

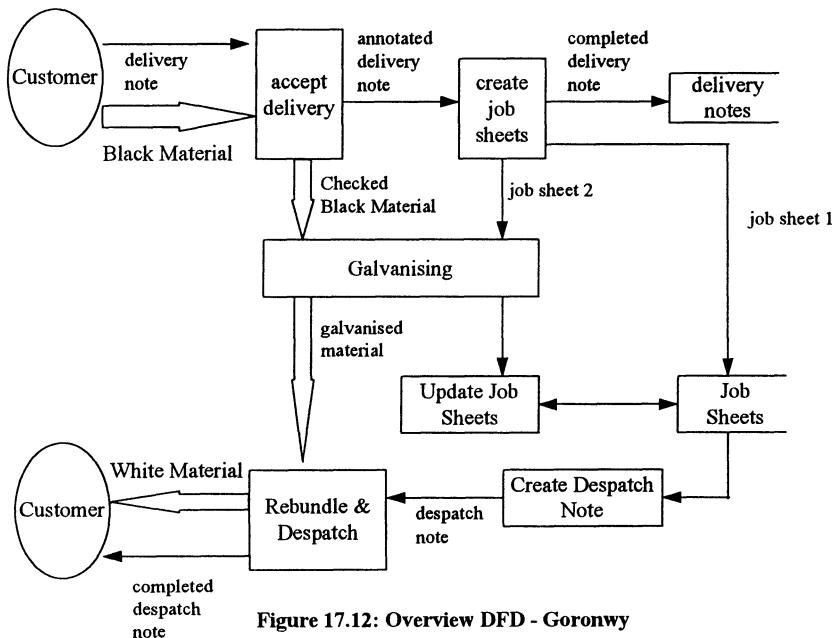


Figure 17.12: Overview DFD - Goronwy

## 17.11 Conclusion

Data flow diagramming as a technique might be summarised as follows:

1. A DFD is composed of processes, flows, external entities and data stores.
2. A DFD forms part of a levelled set of diagrams.
3. A DFD is supported by a data dictionary.

Most contemporary structured information systems development methodologies (chapter 27) exploit this technique in some form. Some object-oriented methods use the technique to specify behaviour of objects (chapter 30). This is because it is an extremely effective process analysis technique applicable in various ways to a number of different stages in the standard project life-cycle.

### 17.12 References

- Cutts G. (1991). (2nd Ed.). SSADM: *Structured Systems Analysis and Design Methodology*. Blackwell Scientific, Oxford.
- DeMarco T. (1979). *Structured Analysis and System Specification*. Prentice-Hall, Englewood-Cliffs, NJ.
- Gane C. and Sarson T. (1977). *Structured Systems Analysis: tools and techniques*. Prentice-Hall, Englewood-Cliffs, NJ.
- Miller G. (1967). 'The Magic Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information'. In *The Psychology of Communication: Seven Essays*. Penguin. Harmondsworth.

### 17.13 Exercises

1. Draw a context diagram for the Fast Foods system.
2. What are the main advantages of using a DFD as opposed to a narrative description of some problem?
3. What is wrong with the DFD extracts in figure 17.13a?
4. What is wrong with the DFD hierarchy in figure 17.13b?
5. Why is an external entity different from an entity as defined in chapter 12?
6. In a house removals system how would you distinguish between the contents being moved and an inventory of the contents on a DFD?
7. Why is the idea of levelling so important in the documentation of large-scale systems?
8. Why do you think we refer to data flow and not information flow?
9. Draw a level 1 DFD for the accept delivery process on figure 17.12.

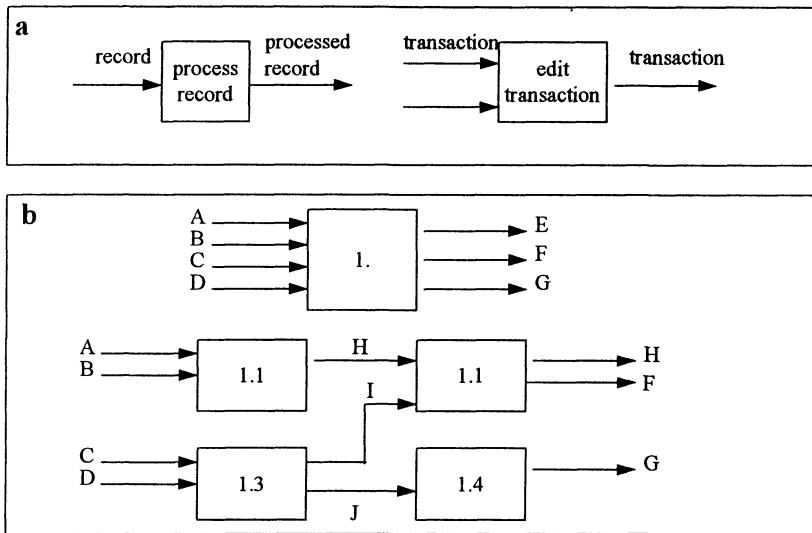


Figure 17.13: DFD Extracts

# **18 Process Descriptions**

## **18.1 Introduction**

According to the structured development tradition, information systems are generally described in terms of three major components: data flows, data structures and processes. Data flows are the subject of the chapters on DFDs (chapter 17) and data dictionaries (chapter 19). Data structures have been discussed using the techniques of normalisation (chapter 15) and E-R diagramming (chapter 16). The third component, processes, are the subject of the current chapter.

One of the major reasons for using structured approaches to systems development is the need to remove as much ambiguity as possible from system descriptions. Such ambiguity frequently arises in the natural language description of processes. For this reason, structured systems development has utilised a number of techniques for describing processes which are less subject to ambiguity. As we shall see (chapter 30), many of the same techniques have been adopted by the object-oriented methods and process mapping has become popular within the domain of Business Process Engineering (chapter 25). The techniques to be covered in this chapter are:

1. Structured English/ pseudo-code
2. Formatted charts and diagrams (Warnier–Orr diagrams, Nassi–Schneidermann charts, action diagrams)
3. Decision tables/trees

## **18.2 Structured English and Pseudo-code**

The constructs of Structured English (sometimes referred to as Program Description Language (PDL) or Program Specification Language (PSL) are a direct emulation of the constructs of structured programming. Descriptions written in structured English look very similar to programs written in block-structured languages such as Pascal (chapter 8). Structured English puts narrative descriptions in a form which removes much ambiguity without losing the benefits of English narrative.

Some people make a distinction between structured English and pseudo-code. A structured English description is an attempt at logical description. As such it remains as high-level, or as implementation-independent as possible. When implementation details become important then such descriptions are best expressed as pseudo-code. Pseudo-code descriptions are physical descriptions. It is a small step from a pseudo-code description to a program.

Another way of looking at it is that structured English is primarily a technique used in the analysis of information systems. Pseudo-code, in contrast, is a technique used in the design of information systems.

### ***18.2.1 The Components of Structured English***

A structured English description of any process is likely to be made up of blocks of imperative sentences embedded within suitable control structures.

#### ***Imperative Sentences***

An imperative sentence usually consists of a verb followed by a number of data items. For example:

READ salesRecord FROM sales

The data items in this sentence reference a data flow (salesRecord) and a data store (sales) on a DFD, each of which would be specified in more detail in a data dictionary. It is conventional to group a collection of imperative sentences into a block, delineated with the keywords BEGIN and END. For example, the block below represents the process of handling sale-slips.

HandleSales

Begin

```
    Receive salesSlip
    READ customerRecord FROM customers
    READ productRecord FROM products
    totalSaleValue = qtySold * unitPrice
    customerCredit = customerCredit + totalSaleValue
    UPDATE customerCredit ON customerRecord
    CREATE salesRecord
    CREATE adviceNote
    WRITE customerRecord to customers
    WRITE salesRecord to sales
    SEND adviceNote to customer
```

END

Note that we provide a name for each process description. Such names will normally refer to templates within a data dictionary (chapter 19). Note also that it may prove useful for a given organisation to standardise on the use of certain keywords such as READ, UPDATE, CREATE and WRITE in the use of structured English.

### ***Control Structures***

According to the advocates of structured programming, only three control structures are needed to describe any process: the sequence, the condition and the loop.

#### *The Sequence*

This is the simplest control structure consisting of a sequence of imperative statements. The process description above is a sequence.

#### *The Condition*

This control structure allows us to deal with a number of different situations. Without the condition statement, a different description would be needed to handle every minor variation in the specification of a problem. Conditions can be divided into three types: single-branched, double-branched and multi-branched. Examples of each type are given below:

#### **Single-Branched**

```
HandleOrders
BEGIN
    Receive Order
    READ customerRecord FROM customers
    IF creditRating is 'good' THEN
        BEGIN
            READ stockRecord FROM stock
            qtyInStock = qtyInStock - orderQty
            UPDATE stockRecord
            WRITE stockRecord to stock
        END
    ENDIF
    CREATE invoice
    SEND invoice TO customer
END
```

#### **Double-Branched**

```
HandleOrders
BEGIN
    Receive Order
    READ customerRecord FROM customers
    IF creditRating is 'good' THEN
        BEGIN
```

```

READ stockRecord FROM stock
qtyInStock = qtyInStock - orderQty
UPDATE stockRecord
WRITE stockRecord to stock

END
ELSE
BEGIN
    CREATE requestForPayment
    SEND requestForPayment TO customer
END
ENDIF
CREATE invoice
SEND invoice TO customer
END

```

#### Multi-Branched

```

HandleOrders
BEGIN
    Receive Order
    READ customerRecord FROM customers
    CASE creditRating is 'good'
        DO updateStockRecord
    CASE creditRating is 'average'
        DO requestForPayment
    CASE creditRating is 'bad'
        DO settlementRequest
    ENDCASE
END

```

The assumption is here that the use of the keyword 'DO' signifies the call of another block of process description.

#### *The Loop*

The loop is an important control structure in that it permits the analyst to repeat a series of sentences without a need for duplicating the necessary descriptions. There are three types of looping structure: do-while, repeat-until and for-next. For example:

```

WHILE there are more orders
BEGIN
    DO handleOrders
END

```

```
ENDWHILE  
  
REPEAT  
BEGIN  
    DO handleOrders  
END  
UNTIL there are no more orders  
  
FOR 100 orders  
BEGIN  
    DO handleOrders  
END  
NEXT
```

### 18.3 Formatted Diagrams

The principles of structured development are also used in a number of other techniques such as Warnier–Orr diagrams, Nassi–Schneidermann charts and action diagrams. These techniques differ from structured English and pseudo-code in providing a structured a diagrammatic layout for process descriptions. Warnier–Orr diagrams use large curly brackets and other special symbols to indicate process structure. Nassi–Schneidermann charts include statements within a segmented box structure. Action diagrams use nested, square brackets, and it is this approach we consider in this section (Martin and McClure, 1985).

Figure 18.1 illustrates the basic constructs of action diagrams. Sequences are represented by statements enclosed in a large square bracket. This substitutes for the BEGIN and END statements of structured English. Conditions are represented by one or more forks in the bracket structure. Loops are represented by a double bar at the top of the enclosing bracket.

Martin and McClure maintain that the addition of a graphical element to structured English improves the understandability of a process description. Action diagrams are much used in the Information Engineering approach to information systems development (chapter 27).

### 18.4 Decision Tables and Decision Trees

Processes that involve many nested conditions are difficult to describe using structured English. For this class of problems, two other techniques, namely decision tables and decision trees, are more appropriate.

Let us assume, for instance, that the process of admitting people into a zoological garden can be described in everyday terms as follows:

A child under 3 years of age is not to be charged an admission fee. A person under 18 is to be charged half full admission. If a child under 12 is accompanied by an adult however, then that person is to be charged quarter full admission. For persons over 18 full admission is to be charged, except for students who are to be charged half admission and senior citizens (women over 60; men over 65) who are to be charged quarter full admission. A discount of 10% is to apply to all persons subject to full admission who are members of a party of 10 or more. Finally, there are no student concessions on the weekend.

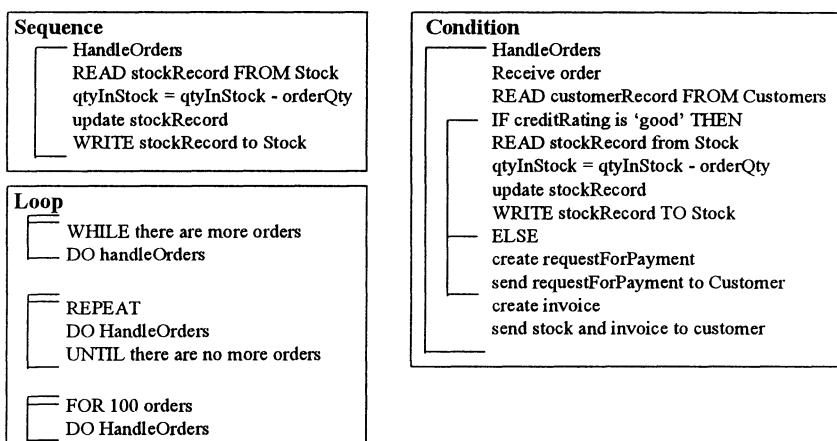


Figure 18.1: Action Diagramming

#### 18.4.1 Decision Table

A decision table for this problem is given below and is made up of four parts:

1. *Conditions stub*. This is the top-left section of the table. It indicates questions to be asked.
2. *Conditions entry*. This is the top-right section. It indicates the particular combinations of conditions that apply.
3. *Actions stub*. This is the bottom-left section.. It indicates the appropriate actions to be taken.
4. *Actions entry*. This is the bottom-right section. It indicates the appropriate action that applies given the combinations of conditions.

The decision table below is an example of an extended decision table. The simpler form of the decision table merely has yes (Y) or no (N) values in the condition entry section. In the admissions problem above, however, age is a crucial factor. Age must use some continuous scale of values.

Condition Stub:	Condition Entry:									
Age	<3	3-12	3-12	12-18	>18	>18	>18	>18	>18	SC
Accompanied		Y	N							
Student					Y	Y	Y	N	N	
Weekend					N	Y	Y			
Party Member						Y	N	Y	N	
Action Stub:	Action Entry:									
Free	X									
Quarter		X								
Half			X	X	X					X
90%						X		X		
Full							X		X	

#### 18.4.2 Decision Trees

Figure 18.2 represents a decision tree for the same admissions problem. A decision tree represents conditions as a series of left to right tests. We first ask the age of the person, as this is the most discriminatory attribute. If the person is between 3 years and 12 years of age, we ask if the child is accompanied. If the child is accompanied he is charged quarter admission. Each node of the tree constitutes an action. A conclusion is reached at a terminal node of the tree.

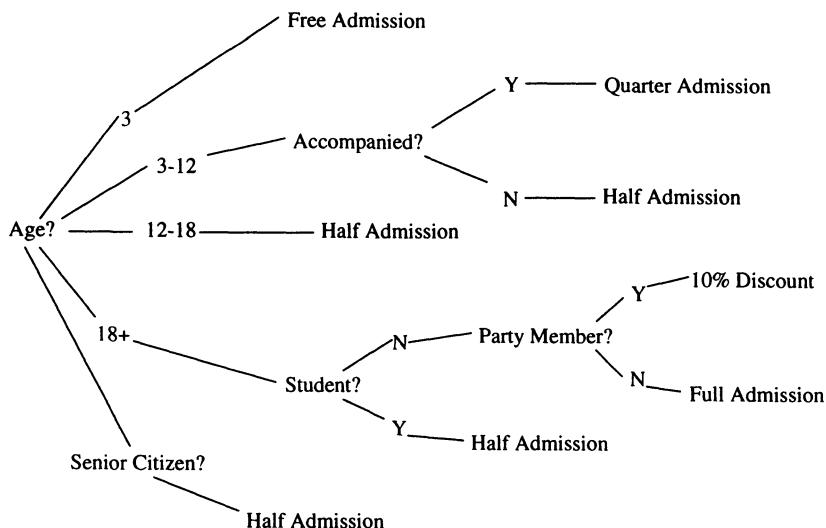


Figure 18.2: Decision Tree - Zoological Garden

### 18.5 Goronwy Galvanising Case Study

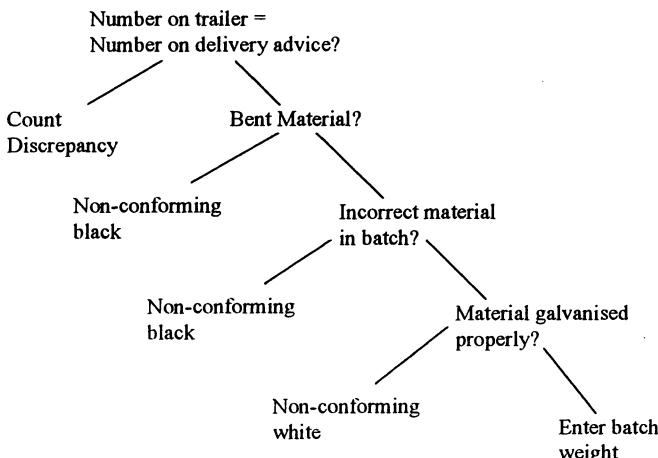
Below we show how structured English can be used to specify in greater detail the processes represented on a DFD. This process description is an attempt to represent the logic employed in the present manual system when job sheets are created:

#### Create Job Sheets

```

DO WHILE there are jobs still left on the annotatedDeliveryNote
    Get blank jobSheet
    Enter deliveryAdviceNo, deliveryAdviceDate
    Enter jobNo, productCode and itemLength
    IF jobEntry is annotated
        CASE count discrepancy
            amend orderQty and batchWeight
            enter discrepancyIndicator
        CASE nonConforming material
            enter nonConforming indicator and nonConforming
            amount
        ENDCASE
    ENDIF
    enter batchWeight on jobSheet
    send jobSheet to shopFloor
    file jobSheet2 in jobSheets
ENDDO
file deliveryNote in deliveryNotes

```



**Figure 18.3: Decision Tree: Non-conforming Material**

Figure 18.3 illustrates a simple application of decision trees to Goronwy. This tree is meant to help production staff fill out the appropriate part of the job sheet dealing with problems in delivery and production.

## 18.6 Conclusion

In this chapter we have considered four techniques for documenting processes. Most of the techniques are equally applicable to design as they are to analysis. This is evident in the hazy distinction between structured English and pseudo-code.

Structured English, or a variant such as action diagramming, is normally the preferred method of documenting processes. Decision tables and decision trees are preferred when one of a large number of actions is to be selected, or where a large number of conditions contribute to the actions undertaken.

## 18.7 References

- Dahl O.J. Dijkstra E.W. and Hoare C.A.R. (1972). *Structured Programming*. Academic Press, New York.  
Martin J. and McClure T. (1985). *Diagramming Techniques for Analysts and Programmers*. Prentice Hall, Englewood-Cliffs, NJ.

## 18.8 Exercises

1. Identify the ambiguities in the following sentence: All customers with more than £1000 in their deposit account who have an average monthly balance in their current account of £100 or who have been customers of the bank for more than five years are entitled to free banking services.
2. Write a structured English sentence for each of the possible meanings in question 1.
3. Draw a decision table or decision tree to represent the following situation: Shady sellers operate a graded commission policy for their salesmen. The company makes a distinction between products selling at more than £10,000 and items selling under this amount. Items above £10,000 are subject to a commission rate of 16% if more than 500 items are sold and if the salesman's salary is below £12,000 per annum. Salaries in the region of £12,000 to £20,000 gain a 14% commission on such items, and salaries above £20,000 gain 12%. If less than 500 items are sold then the commission is 8%, 7% and 6% respectively for each of the above categories of salesmen. For items having a value under £10,000, sales over 1000 items gain a 12% commission for staff on salaries under £12,000. Those on £12,000 to £20,000

gain 11% and those on £20,000 plus get 8%. Under 1000 items of this type of product gain a 6%, 5% and 4% commission.

4. Represent the rules for transforming an E-R diagram into a relational schema as an action diagram.
5. Produce a structured English description of the following scenario. The current production controller at Goronwy describes how he produces a despatch advice as follows: I look through my jobs file for completed jobs – those where the completion indicator is set to Y. If there are enough of these to fill a trailer I'll simply despatch these. I have to do a rough calculation on job weights as the maximum loading on a trailer in 20 tonnes. If there are not enough completed jobs I'll look through the file for jobs that have been partially completed. I'll then part-despatch some or all of these jobs up to the loading limit.

# **19 Data Dictionaries**

## **19.1 Introduction**

A data dictionary is a means for recording the metadata of some organisation (Navathe and Kerschberg, 1986). That is, data about data. Data dictionaries have been used in three ways within information systems development:

1. Conceptual data dictionaries record meta-data/process at a very high level of abstraction.
2. Logical data dictionaries are used to record process and data requirements independently of how these requirements are to be met. Logical meta-data is however at a slightly lower level of abstraction than conceptual meta-data.
3. Physical data dictionaries are used to record design decisions. In terms of data, physical dictionaries store details relating to actual database or file structures. The system tables at the heart of a relational DBMS are a physical data dictionary. In terms of process, physical dictionaries may store details of programs and program libraries.

In terms of metadata therefore we conventionally mean both resources and requirements. A data dictionary is a mechanism for recording the data resources, process resources, data requirements and process requirements of some organisation. This means that a data dictionary cannot be considered solely as a tool for the analysis and design of database systems. It is also an important implementation tool and an important function of what we might call the corporate information architecture.

Note that the term data dictionary is normally stretched to accommodate process detail. Perhaps a data dictionary should really be called something like a project directory or information systems encyclopaedia. Many integrated CAISE tools (chapter 12) use such encyclopaedias to drive development effort. However, the term data dictionary has been the term historically in common use, so we shall continue to use it in this chapter.

## **19.2 Active and Passive Data Dictionaries**

The classic data dictionary is a passive repository for meta-data. In other words, the first data dictionaries were built as systems external to a database and DBMS. They were systems built to record the complexities of a given database structure for use by application development staff and database administrators (DBAs).

If we equate the term data dictionary with the set of system tables in a relational database, then a data dictionary in a relational system is an active repository. It is an inherent, internal part of the database system designed to buffer the user from the base tables.

As we have seen (section 10.2.4), many DBMS are beginning to extend the functionality of such active data dictionaries, particularly in the area of integrity management.

Traditionally, integrity issues have been external to the database system. Integrity has primarily been the province of application systems written in languages such as COBOL or C which interact with the database. Programs are written in such languages to validate data entered into the database and ensure that any suitable propagation of results occurs.

Many have argued, however, that the logical place for integrity is within the domain of the data dictionary under control of the DBMS. The argument is that integrity cannot be divorced from the underlying database. Two or more application systems interacting with one database may enforce integrity differently or inconsistently. Hence, there should be only one central resort for monitoring integrity. Integrity should be the responsibility of the DBA. Mechanisms should be available therefore for the DBA to define and enforce integrity via the DBMS.

### **19.3 Logical Data Dictionaries**

Logical data dictionaries have traditionally been associated with DFDs. DFDs cannot provide all the necessary information to define an information system. For example, a DFD gives us very little idea of the composition of flows, processes and stores. Data dictionaries are used to overcome this deficiency. They record additional information relating to the elements of a set of DFDs.

Logical dictionaries have also been used as an alternative method of representing an entity model. A diagrammatic technique for representing entity models was given in chapter 16. However, when E-R diagrams become sufficiently detailed, the information making up the entity model is more conveniently held in a data dictionary.

Given that we can represent both process and data information in a logical data dictionary, such artefacts are also useful in connecting the elements of DFDs with that of E-R diagrams.

### **19.4 Data Structures and Data Elements**

Gane and Sarson (1977) define data flows as data structures in motion, while data stores are data structures at rest. Hence, both data flows and data stores are made up of data structures. A data flow represents the way in which data moves

between processes. A data store represents some repository for data structures. To this we might add that a process is a mechanism for transforming data structures into other data structures.

Gane and Sarson define a data structure as being a set of data elements that are related to one another and which collectively describe some component of an information system. An invoice is a classic example of a data structure. It is made up of a number of data elements such as invoice number, invoice date and amount due. A data element, sometimes known as a data item, is hence one of the smallest and most fundamental units in an information system.

## 19.5 Contents of a Logical Data Dictionary

There is no readily acknowledged standard for the contents of a logical data dictionary. As a minimum we might limit ourselves to some description of the data structures used by an information system. This is the approach employed in this section.

Analysts often use a notation to limit the amount of narrative needed to describe relationships between data structures and data elements. The one used here is an adaptation of BNF (Backus-Naur form), a notation traditionally used to specify the syntax of programming languages. The notation is made up of the following symbols:

1. An equals sign, = , means that the data structure to the left of the sign consists of whatever is to the right.
2. A plus sign, +, is equivalent to the word ‘and’. It is used to aggregate items together into a data structure.
3. Square brackets, [..], mean that the structure being defined consists of one, and only one of the elements in brackets. This is equivalent to the idea of selection. Each contained item is separated by a semi-colon.
4. Curly braces, {..}, mean that the defined item contains from zero to an infinite number of occurrences of the item contained within braces. This can be used to express the concept of iteration.
5. Information contained in parentheses. (..), is optional. That is, zero or one occurrence of the item may appear.
6. Information contained in asterisks, \*..\*, is regarded as a comment. It hence does not form part of the definition as such.

Hence, we might define an order using this notation in the following terms:

order = orderNo + orderDate + customerNo + salesmanNo + {productNo +  
productName + price + qty + orderValue} + (totalOrderValue)

Or:

```

order = orderNo + orderDate + customerNo + salesmanNo + {lineItem} +
(totalOrderValue)
lineItem = productNo + productName + price + qty + orderValue

```

This second example is an illustration of how we may express a normalised schema for data structures in BNF (chapter 15).

## **19.6 The Gane and Sarson Approach**

For many purposes, the type of data dictionary described above is insufficient since a mechanism for representing a more complete set of information systems requirements is required. What follows is a brief description of a type of data dictionary advocated in the now classic text *Structured Systems Analysis*.

In this approach, each data flow, data store, or process on a set of DFDs is given a unique, meaningful name which is applied consistently throughout the given DFD hierarchy. The first part of the data dictionary lists such names suitably nested to reflect the levelling of DFDs. This comprises the table of contents to the data dictionary.

Each item referred to in the table of contents is then given a template description of its own.

1. The template for data flows details the name, a description of the flow, from what process the flow is coming, and to what process the flow is going, followed by a list of data structures used by the flow.
2. The template for processes details a name and description followed by a list of inbound and outbound data flows. When we have reached a sufficient level of detail, we might even include a process description in something like structured English.
3. Data stores are described by items such as the name of the store, a description, data flows and a list of the data structures stored. At the design stage we might include some indication of the volume of data and the type of access required to data.

When we reach a sufficient level of detail we may seek to document the data structures and elements used by the constructs above.

### **19.6.1 Case Study: An Order-processing System**

Suppose we have to document a simple order-processing system. Extracts from the table of contents might look as follows:

*Data Stores*

**CustomerAccountsReceivable**

**customerName**  
  **customerAddress**  
  **dateBilled**  
  **amountBilled**  
  **interestCharges**  
  **currentBalance**

**Invoice**

**invoiceNo**  
  **customerName**  
  **productNo**  
  **qty**  
  **totalCharge**

*Data Flows***acknowledgement**

**customerName**  
  **customerAddress**  
  **acknowledgementBody**

**order**

**customerName**  
  **customerAddress**  
  **productNo**  
  **qty**

**statement**

**customerName**  
  **customerAddress**  
  **currentBalance**

*Processes***Enter Order**

**Acknowledge Order**  
  **Verify Order**  
  **Approve Order**  
  **File Order**

**Produce Invoice**

**Prepare Invoice**  
  **Send Invoice**  
  **File Invoice**

Each of the elements in the table of contents can be expanded as below:

<b>Data Flow Name:</b>	invoice
<b>Description:</b>	details of the order for which the customer is billed
<b>From Processes:</b>	4.1 prepare invoice
<b>To Processes:</b>	4.2 assign invoice number
<b>Data Structures:</b>	invoice

<b>Process Name:</b>	1.0 Enter Order
<b>Description:</b>	customer order received and approved for further processing
<b>Inbound Flows:</b>	approved order
<b>Outbound Flows:</b>	order details

<b>Data Store:</b>	Approved Invoices
<b>Description:</b>	Itemises merchandise received, cost of each and contains signature of receiving employee
<b>Inbound Flows:</b>	invoice
<b>Outbound Flows:</b>	batchedInvoiceDetails
<b>Definition:</b>	approvedInvoice = invoice + signature
<b>Volume:</b>	200 daily
<b>Access:</b>	accessed in batches. Sequentially processed from within batch

## 19.7 Data Dictionaries for Entity Models

Representing an entity model in a data dictionary has a number of advantages over its diagrammatic representation. Perhaps the most important is that large entity models are more easily maintained in the form of a data dictionary than in the form of a diagram.

We consider here a simple scheme for representing entity models based upon the template approach of Gane and Sarson. This scheme can be quite easily extended to handle the requirements of an object model as described in chapter 22. Each entity in the dictionary is given a template made up of the name and the attributes of the entity. Each relationship is also given a template in which we detail the names of participating entities and the properties of the relationship as below:

<b>Entity Name:</b>	Customer
<b>Identifier:</b>	customerNo
<b>Attributes:</b>	customerName, address, telNo

<b>Relationship Name:</b>	makesOrder
<b>Participating Entity Name:</b>	Customer
<b>Cardinality:</b>	1
<b>Optionality:</b>	optional
<b>Participating Entity Name:</b>	Order
<b>Cardinality:</b>	N
<b>Optionality:</b>	mandatory

## 19.8 Entity Models and DFDs via Data Dictionaries

A data dictionary may act as a useful means of connecting up the static information provided by entity models with the dynamic information represented on data flow diagrams. The easiest way to achieve this integration is to recognise the commonality between these two forms of representation in terms of the concept of a data structure. Both data stores and entities are mechanisms for clustering data elements into data structures. Data flows are data structures in motion. Processes are mechanisms for transforming data structures.

Data structure references can hence be transferred between data and process models in a data dictionary. In this way, a data dictionary frequently acts as the centralised repository for many computer aided information systems (CAISE) engineering tools (chapter 12).

## 19.9 Goronwy Galvanising Case Study

The major data structures at Goronwy are delivery advices/notes and despatch advices/notes. These can be formally specified using our modified BNF as below:

deliveryAdvice = deliveryNo + deliveryDate + {jobNo + productCode + description + itemLength + orderQty + batchWeight}

deliveryNo = {blackheadsNo; otherNo}

jobNo = {blackheadsNo; otherNo + lineNo}

despatchAdvice = despatchNo + despatchDate + {jobNo + productCode + description + itemLength + orderQty + batchWeight + returnedQty + returnedWeight}

Note how we have specified a delivery number as having two possible definitions: a Blackheads number and another manufacturer number.

The Gane and Sarson approach would also involve us in documenting each of the flows, stores and processes in the Goronwy application. A sample template given for one of the data stores on the overview DFD specified in chapter 17 is given below:

<b>Data Store:</b>	Job Sheets
<b>Description:</b>	Record of incoming black material and processed white material
<b>Inbound Data Flows:</b>	jobSheet1
<b>Outbound Data Flows:</b>	jobSheet
<b>Definition:</b>	$jobSheet = jobNo + productCode + itemLength + orderQty + batchWeight$

### 19.10 Conclusion

Besides its use as an addendum to DFDs and/or entity models, a data dictionary has a number of other uses within the process of information systems development:

1. As a means of enforcing a standard set of data representations for a team undertaking a software project.
2. As a means of checking the consistency and completeness of a representation of some information system.
3. If the dictionary is sufficiently comprehensive and rigorous, the generation of data definitions and even programs becomes possible (see chapter 12).

### 19.11 References

Gane C. and Sarson T. (1977). *Structured Systems Analysis: Tools and Techniques*. Prentice Hall, Englewood Cliffs, NJ.

Navathe S.B. and Kerschberg L. (1986). Role of Data Dictionaries in Information Resource Management. *Information and Management*. 10. 21-46.

### 19.12 Exercises

1. Write a BNF description of a textbook.
2. Write a BNF description of a data dictionary.

3. What is wrong with the following BNF description:  
totalItemPrice = unitPrice + vat?
4. A clerk within Fast Foods describes an invoice in the following way: Each invoice is given a unique number. The customer's name and address are written at the top of the invoice and the date of despatch may be written on by the despatch handler. Underneath the customer information we list for each product ordered the product number, description, quantity ordered, unit price and value of the order. At the bottom of the invoice we write the total value of the order. Write a modified BNF definition for this type of invoice.
5. In what way do you think modified BNF can be used to aid the process of normalisation?
6. At what size of entity model do you think a data dictionary representation becomes more convenient than a diagrammatic representation?

# **20 Entity Life Histories**

## **20.1 Introduction**

The entity life history (ELH) technique was originally designed to extend the available database design techniques such as E-R diagramming (chapter 16) and normalisation (chapter 15) (Robinson, 1979). Such data analysis techniques concentrate almost exclusively on a static view of the information system being modelled. ELHs, in contrast, were developed as a technique for the explicit modelling of system dynamics (Rosenquist, 1982). In this sense, the ELH may be seen as a competitor to the DFD (chapter 17) and associated process specification techniques (chapter 18). However, the ELH is more closely linked to the logical modelling ideas underlying data analysis.

A number of structured methodologies, such as SSADM (chapter 27), have cast the ELH in a more integrative role, mediating between such techniques as E-R diagrams and DFDs (Cutts, 1991). Jackson (1984) used a similar representation to one of the notations discussed in this chapter as the basis of his structured programming methodology (chapter 21). More recently, a number of object-oriented methods have used a similar technique to the ELH to specify elements of object behaviour (chapter 22).

## **20.2 Definition and Objectives**

From a traditional data modelling point of view, the entities that we have considered in our discussion of E-R diagramming are usually thought of as static or structural concepts. An entity model represents a time-independent slice of reality.

An entity may however be considered from a contrasting point of view. An entity is realistically in a state of flux. It is the subject of a large number of processes or events which change its state. In a library, for instance, the entity *Book* may proceed through a number of different states: it is first acquired, then it is catalogued, it is lent to borrowers, and perhaps finally it is sold off.

An ELH is a diagrammatic technique for charting the usage of a particular entity by the processes or events making up an information system. Its primary objective is therefore to offer a means to connect up the entities detailed on E-R diagrams, with the processes present on a set of DFDs.

As a result, ELHs are useful in two senses. First, developing an ELH helps analysts to understand entities better. Second, they are the first step in documenting the detailed outline of processes. As such, they are an extremely useful technique for validating DFDs.

## 20.3 Technique

There are at least three different conventions used for drawing an ELH:

1. A notation based on the theory of Petri nets or state transition diagrams.
2. A network-like notation derived from PERT scheduling.
3. A hierarchical convention similar to the symbols used in JSP (Jackson Structured Programming) which we shall discuss in chapter 21.

Each of these conventions incorporates the basic characteristics of the ELH as an analysis and design tool. Therefore, primarily for the sake of brevity, we choose to concentrate on describing the hierarchical notation. We also briefly discuss the notation of state transition diagrams. Those interested in more detail on the Petri net and network-like notation are referred to Martin and McClure (1985). We further assume that ELHs are to be used in conjunction with other techniques such as DFDs and E-R diagrams. The production of any set of ELHs is a four-step process:

1. List entities from E-R diagrams.
2. List events from DFDs.
3. Construct an ELH matrix.
4. Produce an ELH for each entity.

### 20.3.1 List Entities

The entities used in a set of ELHs are usually taken directly from an E-R diagram or set of E-R diagrams. In a simple library system we might have the following entities: *Book, Borrower, Reservation, Loan*.

### 20.3.2 List Events

Events are usually taken from the set of DFDs documenting an information system. An event can be thought of as a real-world transaction, such as the arrival of a sales order or the receipt of goods. The net effect of an event is to change the state of an entity. Hence, we might say that a catalogue event changes the state of a *book* entity from acquired to catalogued. However, events can be categorised in a more abstract sense in terms of the effect that they have on the life of an entity. Four such primary events are possible: create events (C), read events (R), amend events (A) and delete (D) events. A list of possible events for our simple library system might be:

1. Creating a borrower: a C event for Borrower.
2. Acquiring a book: a C event for Book.

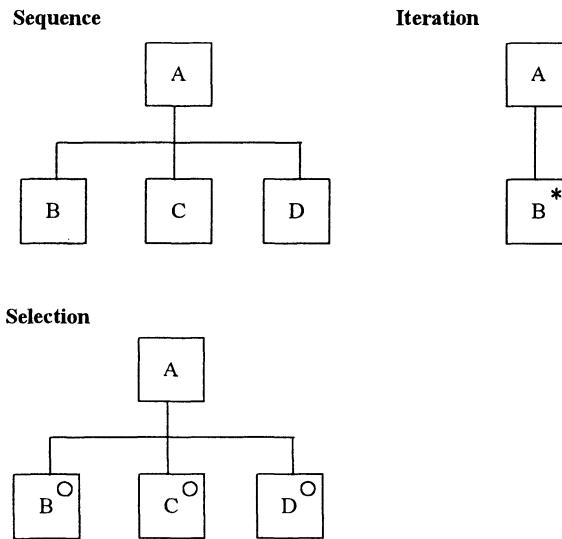
3. Cataloguing a book: an A event for Book.
4. Loaning a book: an A event for Borrower and Book. A D event for Reservation and a C event for Loan.
5. Selling a book: a D event for Book.

### **20.3.3 Constructing an ELH Matrix**

A 2-dimensional matrix can be produced in which each cell represents the action of an event on an entity.

	Book	Borrower	Reservation	Loan
Create a Borrower		C		
Acquire a Book	C			
Catalogue a Book	A			
Loan a Book	A	A	D	C
Reserve a Book	R		C	A
Sell a Book	D			

### **20.3.4 Constructing an ELH**



**Figure 20.1: ELH Notation**

An ELH matrix suffers in its ability to document the sequence of events which impact on an entity. It also fails to document the effect of abnormal or exceptional events. ELHs are drawn to overcome these inadequacies. An ELH charts the sequence of events in the life of an entity. They also chart the effect of abnormal events.

In our notation an ELH constitutes a hierarchical system of boxes. The root box in the hierarchy indicates the name of the entity under consideration. The lower-level boxes represent the events that act upon the entity. ELH diagrams use three basic constructs (figure 20.1):

1. *Sequence*. Represented by a horizontal row of boxes read from left to right.
2. *Selection*. Represented by boxes at the same level in the hierarchy with an ‘o’ for optional within each box.
3. *Iteration*. Represented by boxes with an asterisk, ‘\*’, within the box.

#### 20.4 Creation of ELHs

The easiest way to construct an ELH is to start with a simple entity life first and add complexities later. Many entities have simple lives in which an entity is created, read and/or modified a number of times, and eventually deleted. In our library system, for instance, a *book* is first created by an acquisitions process, then it is modified by a cataloguing and lending process, and finally it is deleted by a process which sells off old books (see figure 20.2a).

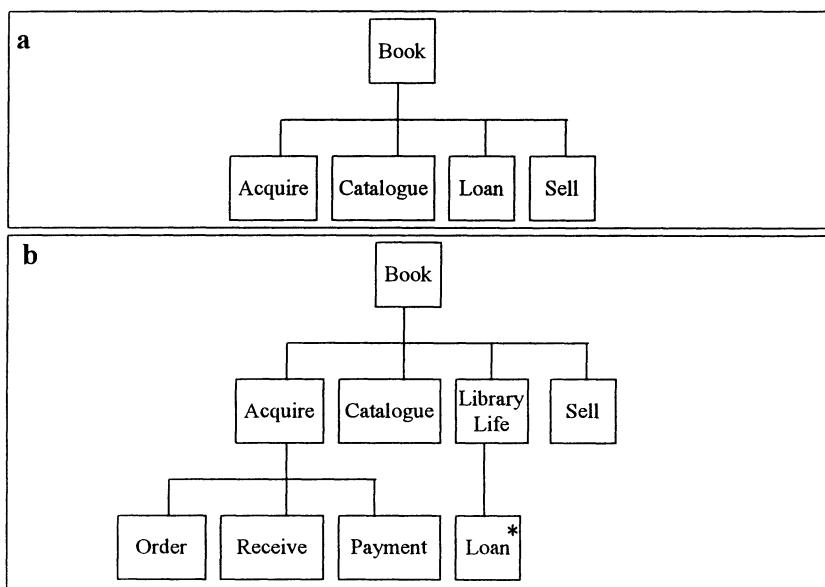


Figure 20.2: Building up an ELH

A complexity arises when we consider that a book is likely to be loaned a number of times throughout its life. We therefore demote the loan box to a position underneath a box which we now label library life. We also designate a loan to involve iteration. Similarly, we break down the acquisition process into the sub-processes order, receive and payment (see figure 20.2b).

Further complexities occur if we consider a loan event in greater detail. The loan of a book actually involves two separate processes which we label issues and returns. The issuing process is relatively straightforward. We record details of the borrower and the book, we stamp the book with a return date, and we issue the book to the borrower. Most returns will, we hope, take place before or on the return date, but inevitably some books will run overdue. In this case we issue as many return requests as it takes to get the book returned (see figure 20.3).

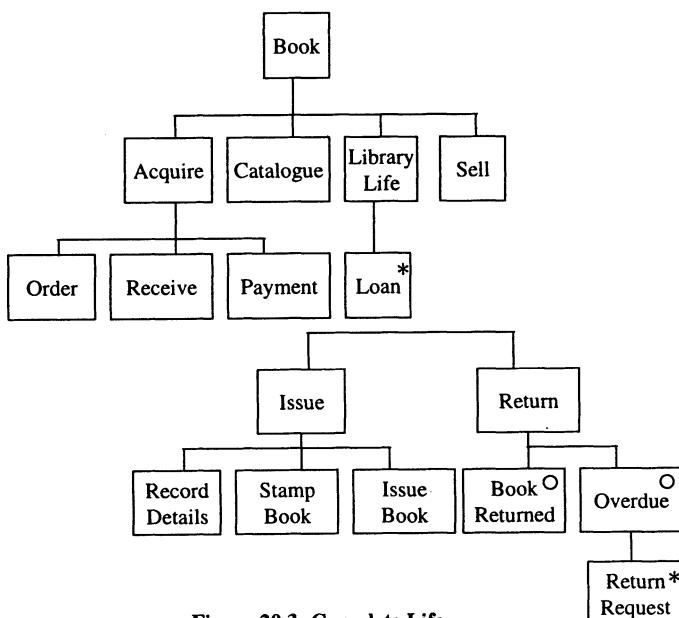


Figure 20.3: Complete Life

## 20.5 Goronwy Galvanising Case Study

Consider one of the fundamental entities appropriate to the information system at Goronwy, *Job*. Figure 20.4 illustrates how we might begin drawing a life history for this entity. A *Job* has a standard life composed of creation at the time of delivery, modification while at the plant and deletion at time of despatch. Delivery is made up of unbundling and checking batches. If a batch has some discrepancy then an annotation is made to the delivery note. The plant life of the job consists of creating a job sheet, processing the job and marking completion.

The despatch either directly emulates the original batch delivered, in which case it is a full despatch, or it goes back on separate trailers, in which case it is made up of a series of partial despatches.

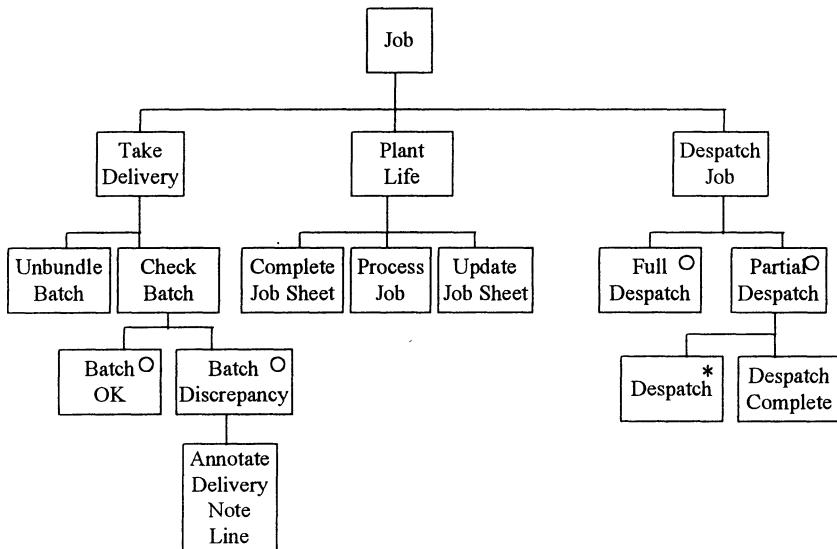
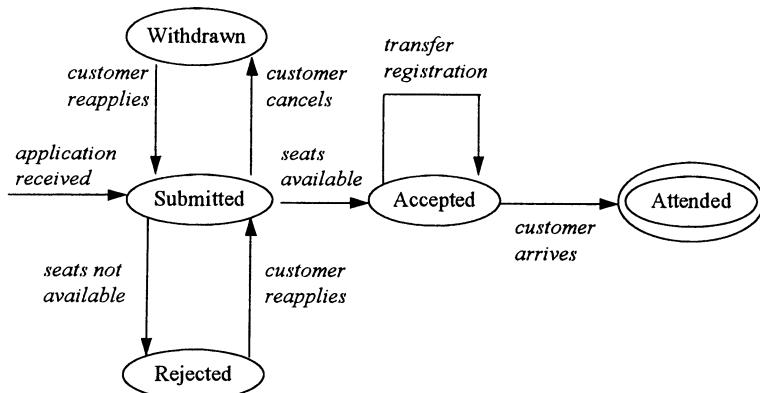


Figure 20.4: ELH for Job entity

## 20.6 State Transition Diagrams

An alternative notation for entity life histories is considered briefly in this section. This notation is based on the idea of a finite state machine. A finite state machine is a hypothetical mechanism which can be in one of a finite number of discrete states at one time. Events cause changes to the machine's state. A given process can therefore be represented as a series of finite state machines.

Suppose we are interested in recording the process of people registering for attendance on a technical seminar. A given registration can be in one of the following number of states: submitted, accepted, rejected, withdrawn, or attended. The relationships between states and events can be represented on a diagram as in figure 20.5. The states are represented here as bubbles, the events as directed arrows between bubbles. An incoming arrow without an associated starting state initialises the process. A double-bubble indicates the terminating state. Note also how the transfer registration event cycles on the same state.



**Figure 20.5: State Transition Diagram**

## 20.7 Conclusion

A sequence of steps for the generation of ELHs might be summarised as follows:

1. Select from the ELH matrix an entity and the events that affect it.
2. Order the events in sequence.
3. Include selections and iterations.
4. Rework the ELH if the iterations and selections alter the sequence.

As an aid to E-R diagramming and perhaps data flow diagramming, ELHs can be used at the feasibility study, analysis and design stages of the standard project life-cycle. Jackson has used a similar technique as a tool for deriving program structures (chapter 21). In this respect, ELHs can be seen as relevant to the implementation phase.

## 20.8 References

- Jackson M.A. (1984). *Principles of Program Design*. Academic Press. New York.
- Martin J. and McClure T. (1985). *Diagramming Techniques for Analysts and Programmers*. Prentice Hall, Englewood Cliffs, NJ.
- Robinson K.A. (1979). 'An Entity/Event Data Modelling Method'. *The Computer Journal*. 22. 270-281.
- Rosenquist C.J. (1982). 'Entity Life Cycle Models and their Applicability to Information System Development'. *The Computer Journal*. 25. 307-315.

## 20.9 Exercises

1. Consider a customer in an order-processing system such as Fast Foods. The event that makes the system aware of this entity is when an account is opened and details are first taken of the customer. During the Customer's active life he or she might place orders, take deliveries, be invoiced and make payments. Assume that the only reason the warehouse may lose a customer is if he or she goes out of business. Draw a simple life for this system.
2. Draw an entity life history for an order within the Fast Foods system. An order is first produced by an order entry process, then modified by a process which produces a delivery note and invoice, and finally deleted by a process which receives the payment of invoice.
3. Combine the ELH produced for question 1 with the ELH for question 2.
4. A new customer opening an account with Fast Foods may negotiate a discount. The discount rate offered to a customer is dependent on the delivery distance. Occasionally, customers will move to new premises. In this case, a new discount rate needs to be renegotiated. Incorporate this analysis into the ELH produced from 3.
5. Redraw the ELH of question 1 as a state transition diagram.
6. Redraw the ELH of question 2 as a state transition diagram.
7. Discuss the relative merits of the hierarchical and state transition notations for ELH.
8. Draw a state transition diagram for the states of unmarried, married, divorced, widowed, and separated.
9. Redraw the ELH from Goronwy Galvanising in figure 20.4 as a state transition diagram.

## ***Part Three Section Three***

### ***Other Techniques***

In this section we describe a number of techniques which cannot logically be placed either in the data or process analysis camp. Structured program design fundamentally involves taking the results from process and data analysis and building designs for procedural programs. User interface design is an area whose product, a model of the user interface, has achieved an equal place alongside a process model and data model within the context of an information systems model. Object orientation has had a crucial influence on contemporary approaches to analysis and design. In chapter 22 we portray how many structured techniques have been adapted and extended for the purposes of object-orientation. Finally, the process of building a prototype of an intended system has become significant particularly in participative approaches to development work. This is the topic of chapter 24.

# 21 Structured Program Design

## 21.1 Introduction

In previous chapters we have concentrated on a range of techniques applicable to the analysis and design of information systems. In this chapter we begin to concentrate on implementation. We discuss a number of techniques applicable to the process of program design.

Program design can be described as the process whereby the requirements defined in the documentation of some system are turned into a representation the purpose of which is to direct actual coding. Since the essential role of requirements is to show how for each process in a system input is transformed into output, the program design process may be represented schematically as in figure 21.1.

From this diagram three basic alternatives to program design are possible. Two are structured approaches; one is founded in the principles of object-orientation.

One approach based on the idea of functional decomposition has been promulgated by Yourdon and Constantine (1979). The other, data-oriented approach is due to Jackson (1975) and Warnier (1981).

An object-oriented approach to program development attempts to encapsulate both data and functions. It is exemplified by the likes of Booch (1990). This topic is covered in chapter 30.

In this chapter we present an overview of the structured approaches to program design. Readers wishing to learn more of the various approaches are referred to the works cited above.

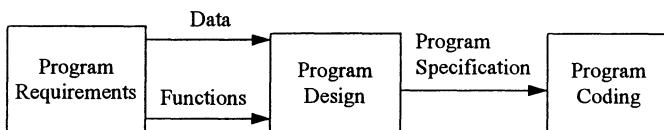


Figure 21.1: Program Design

## 21.2 The Functional Approach

In the late 1970s, Edward Yourdon and Larry Constantine published an influential book entitled *Structured Design: Fundamentals of a Discipline of Computer Program and System Design* (1979). The major theme of this work was that the fundamental problem with software is complexity. Yourdon and Constantine introduced a number of principles for managing such complexity

which they bundled together under the term structured design. Structured design seeks to conquer the complexity of large information systems in three ways:

1. By partitioning systems into black boxes.
2. By organising these black boxes hierarchically.
3. By ensuring that each black box demonstrates the properties of high cohesion and low coupling.

#### ***21.2.1 Partitioning the System into Black Boxes***

A black box is a software module in which:

1. The inputs to the module can be clearly specified.
2. The outputs from the module can be clearly specified.
3. The function of the module can be clearly specified. That is, what it does to the inputs to produce the outputs.
4. One does not need to know how the module carries out its function in order to use it.

An example of a black box would be a module which produces a list of customers in sorted order of name. This is a specification of its function. Its inputs constitute a customers file. Its outputs constitute an ordered listing of customers. We do not need to know how the module is implemented in order to use it.

The principles of the black-box approach are as follows:

1. Each black box should solve one well-defined piece of the problem. This normally means that each module should perform at most one function.
2. The system should be suitably partitioned so that the function of each black box is easy to understand. Division along ‘natural’ boundaries is hence preferred to division along the lines of pure efficiency.
3. Partitioning should be such that any connection between modules is introduced only because of a connection between pieces of a problem.
4. Partitioning should aim to achieve the simplest connection between modules.

#### ***21.2.2 Organising Black Boxes into Hierarchies***

It is useful to make the analogy between the structure of a computer system and the structure of a business organisation. This helps us to suggest some general guidelines for the design of information systems:

1. Work and management should be separated in a system. Work should be done by subordinates with no managerial duties. Managers should be restricted to co-ordinating the work of subordinates.
2. A manager should not have more than seven immediate subordinates (Miller, 1967).
3. Every department should have a well-defined function: Every job should be allocated to the proper department; Messages between functions should be clear and meaningful; Managers should give only as much information to a subordinate as that person needs to do his job.

This analogy is useful in emphasising that most viable information systems are subject to some form of restricted hierarchy. Control in the system is vested in manager modules in the upper regions of the hierarchy. Work in the system is done at the lowest levels in worker modules. Communication between modules is through well-defined channels using simple messages. This is similar to the generic model of an information technology system presented in chapter 5.

### 21.3 Coupling and Cohesion

As well as black-boxedness and hierarchy, Yourdon and Constantine provide two other concepts which they use to assess the quality of any software design: coupling and cohesion.

#### 21.3.1 Coupling

Coupling is a measure of the degree of interdependence between modules. The objective of good systems design is to minimise coupling; i.e. to make modules as independent as possible. Low coupling indicates a well-partitioned system:

1. The fewer connections there are between two modules, the less chance there is of a bug in one module appearing as a symptom in another.
2. We want to be able to change one module with the minimum risk of having to change another module. We also want each change to affect as few modules as possible.
3. While maintaining one module, we do not want to worry about the internal working of another module.

Low coupling can be attained in three ways:

1. By eliminating unnecessary relationships between modules.
2. By reducing the number of necessary relationships.
3. By easing the ‘tightness’ of necessary relationships.

Low coupling occurs if two modules communicate solely by parameters, each parameter being an item of data. Data coupling constitutes the necessary communication between modules. High coupling occurs when, for instance, two modules refer to the same common data area. Common coupling is dangerous, particularly for its tendency to propagate software errors.

### **21.3.2 Cohesion**

Cohesion is a function of how closely elements within a single module are related together. The objective is strong, highly cohesive, modules – modules in which all the elements are genuinely related together.

High cohesion is functional cohesion. A functionally cohesive module is one in which all the elements contribute to the execution of one, and only one, task. Read transaction, determine customer mortgage repayment, calculate net employee salary, are all probably functionally cohesive modules.

Low cohesion is temporal cohesion. A temporally cohesive module is one in which the elements are related together in time. The classic example of a temporally cohesive module is an initialisation module. The problem with such a module is that it is difficult to reuse. If the module involves such activities as clearing matrices and setting pointers to the start of sequential files, it is impossible to initialise any one file without resetting the entire system.

Coupling and cohesion are clearly interrelated. The greater the cohesion within individual modules of a system, the lower the coupling between modules.

## **21.4 DFDs and Design**

Data flow diagrams can be used as the first stage of the program design process. Yourdon and Constantine demonstrate a technique of moving from DFDs to structured programs via a notation called structure charts. A structure chart is a graphic representation of the hierarchical decomposition of a system in terms of black boxes. A structure chart is made up of three elements:

1. The module. A named, bounded, contiguous set of statements, represented by a labelled, rectangular box.
2. The connection. Any reference from one module to another. A connection normally means that one module has called another. It is represented by a directed arrow or line between two modules.
3. The couple. An item that moves from one module to another. It is represented by a short arrow with a circular tail. A dotted circle means that an element of control is involved. An open circle means that an element of data is being passed between two modules.

Figure 21.2 illustrates a schematic structure chart.

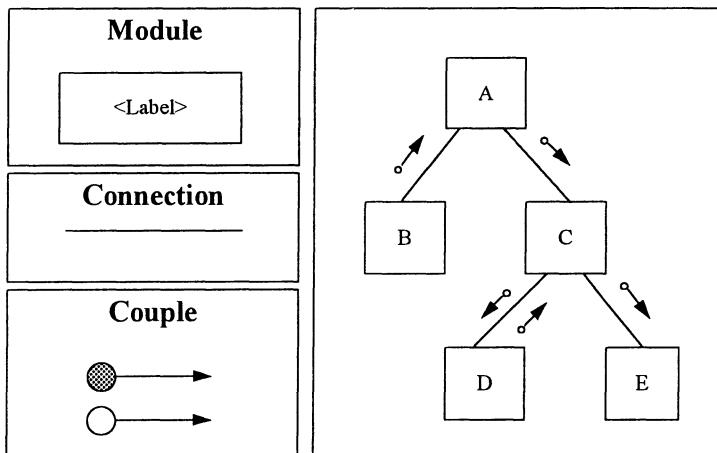


Figure 21.2: Structure Chart Notation

A DFD is a statement of requirement. It declares what has to be accomplished. A structure chart is a statement of design. Design must be derived from requirement. An appropriate structure chart must be derived from a corresponding DFD. Yourdon and Constantine specify two methods for such derivation: transform analysis and transaction analysis. These two approaches deal with two different types of DFD. The main objective of transform analysis is to identify the primary processing functions in a system. In contrast, transaction analysis applies to those cases where a process splits an input data stream into several discrete output streams. Transform analysis is undoubtedly the technique applicable to more systems than transaction analysis. We will therefore concentrate on this technique in the present chapter.

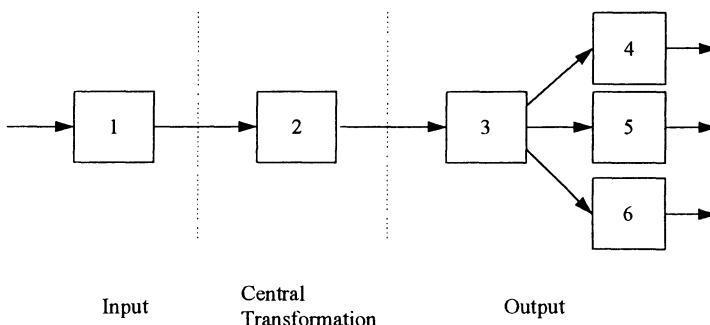
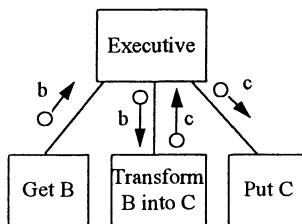


Figure 21.3: Transform Analysis

## 21.5 Transform Analysis

Transform analysis involves the following steps:

1. State the problem as a DFD. Figure 21.3 represents a schematic DFD from which we wish to derive a structure chart.
2. Identify high-level inputs and high-level outputs. High-level inputs are those elements of data that are furthest removed from physical input, yet still constitute inputs to the system. To identify high-level inputs we start at the extreme physical inputs to the system and move inward along the flows of the DFD until we identify a set of data flows that can no longer be considered as incoming. We perform this for each input stream. High-level outputs are those data elements that are furthest removed from physical outputs but which still may be regarded as outgoing. To identify high-level outputs, we start at physical outputs and perform the same process as for inputs. These steps usually leave a set of processes (transforms) which are neither high-level inputs nor outputs. These are designated the central transforms of the system. They constitute the main work of the system in that they transform major inputs into major outputs. Process 2 has been designated the central transform on figure 21.3.
3. First Level Factoring. Having identified the central transform, we then begin the process of constructing the structure chart. All such charts begin with a top-level control module called the executive. Underneath this executive we need three modules: one module to get the data flow B; one module to transform B into C; and one module to output C. Figure 21.4 represents this first level factoring of the DFD in figure 21.3.
4. Lower-level Factoring. We continue this factoring process until we are sure that we have adequately represented the problem. Figure 21.5 represents the factoring of the get B module, and put C module.



**Figure 21.4: First Level Factoring**

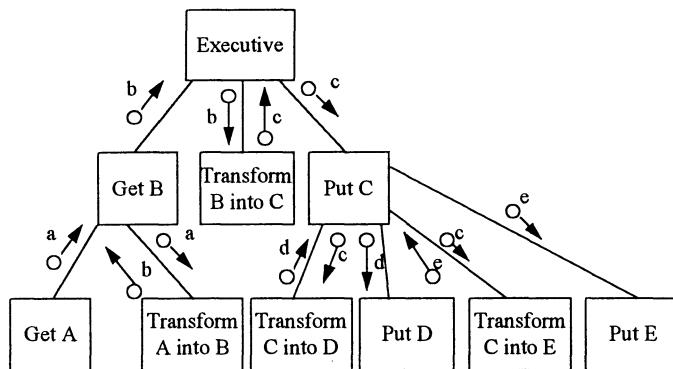


Figure 21.5: Lower Level Factoring

## 21.6 Case Study: An Examinations System

The diagram in figure 21.6 describes the data flow through a system for handling student examination results. Sets of examination results are first read from a file and validated. Invalid marks are written to another file, and statistics of the number of valid and invalid marks are kept.

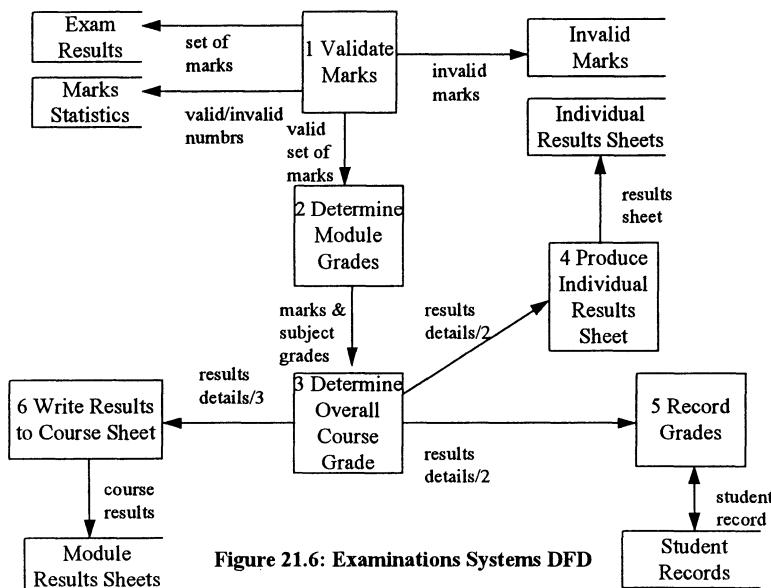


Figure 21.6: Examinations Systems DFD

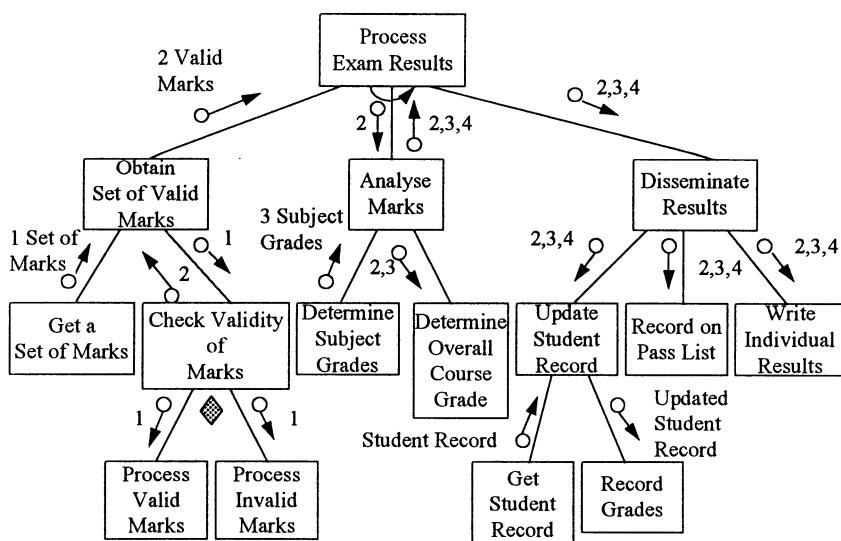
The valid marks are then processed to determine the relevant student grades in each subject, and the overall grade for the course. Marks and grades for each student are recorded in a student records file. An individual results sheet is also

produced for each student, and an entry is made in the overall course results sheet.

The task is to convert this network representation into a hierarchical representation. The first step is to identify the central transform or transforms of the system. In our example, these are the processes labelled determine subject grades and determine overall course grade. In more complex, and probably more realistic situations, identification of the central transform usually requires considerable analysis. Indeed, different analysts, working independently, might identify different processes as being the central transform.

The next step is to identify those data flows which input data to the main processes, and distribute data from the main processes. In our example, the data flow named valid sets of marks inputs to the transform and the data flows named valid set of marks, subject grades and overall course grade outputs from the central transform as a data package called results detail.

The structure chart is then produced by defining the executive module. This we have chosen to call process exam results. Underneath the executive we construct a data collection module (obtain set of valid results), a processing module (analyse marks), and an output module (disseminate results) (figure 21.7).



**Figure 21.7: Exam System Structure Chart**

These modules in turn control further modules at lower levels in the hierarchy which perform subordinate processes. The sequence of execution of the various modules making up the structure chart is from left to right. Iteration is shown on the chart by a circular arrow. In our example this circular arrow indicates that the top-level modules are to be repeated. Selection can also be

indicated by the use of a diamond between the appropriate connections. On our structure chart, the module check validity of marks has the choice of performing either the process valid marks module or the process invalid marks module.

It remains to annotate the structure chart with the data couples that are passed between modules. The emphasis in structured design is on the use of data couples as opposed to control couples. It is possible, for instance, to incorporate the process valid marks and process invalid marks into one module. However, a control couple would then be needed to indicate the appropriate function to be carried out. The existence of control couples results in a high degree of coupling between modules which, in turn, leads to increased amendment problem.

The resulting structure chart is thus an example of good software design. It is characterised by low coupling between modules and high cohesion within modules. Each module comprises a black box in that it is a simple module which performs one function well. The only communication between modules is through data couples. Control is exercised at appropriate points in the hierarchy via the simple calling of appropriate modules. All these characteristics lead to systems that are easy to understand, construct and maintain.

## 21.7 Data-oriented Program Design

The basic premise of data-oriented program design is that the structure of the data determines the structure of the program. Probably the best known, and certainly the best documented, program design technique was developed by M. Jackson in the early to mid-1970s (Jackson, 1975). Jackson Structured Programming (JSP) was subsequently chosen as a standard for all UK government installations as Structured Design Methodology (SDM).

## 21.8 Structure Diagrams

In order to design programs from data, Jackson needs a notation to represent both data structures and process structures. Jackson uses a hierarchical diagram, known as a structure diagram, to represent both.

For any programming problem, structure diagrams may be constructed from the three basic constructs of sequence, selection and iteration. The notation used is identical to that described in the chapter on entity life histories (chapter 20).

## 21.9 Stages of JSP

JSP comprises five distinct stages performed in strict sequence:

1. Draw a structure diagram for each input and output in the problem.

2. Identify correspondences between data structures.
3. Use correspondences to combine separate data structures into a single program structure.
4. List the executable operations and allocate them to appropriate places within the program structure.
5. Write schematic logic for the program.

### 21.10 Case Study: A Simple Banking Program

Let us assume that we are given the following simple problem:

A program is required to process a file containing a series of transactions in account number sequence to produce a printed summary report for a manager. In addition to the account number, each transaction record contains the amount of the transaction and an indicator to type the transaction as credit or debit. The summary report is to be headed 'account summary' and is to have one line per account showing the total value of transactions for each account.

A context DFD for this simple system might be drawn as in figure 21.8a.

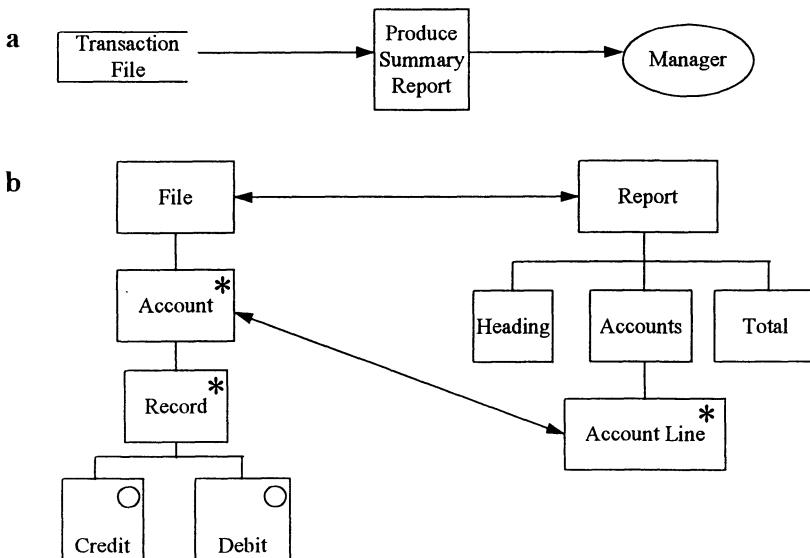


Figure 21.8: Banking System DFD and Structure Diagrams

Our input data flow is the file of transactions; our output data flow is the summary report. The first step is to draw structure diagrams for both of these flows. Two such diagrams are presented in figure 21.8b.

The second step is to identify correspondences between these data structures. For correspondences to exist, the following conditions must be satisfied.

1. Each item in the data structures must occur the same number of times.
2. The items must be in the same sequence.

In our example, File corresponds with Report, since there is only one of each for each program execution. Similarly there is a correspondence between Account and Account Line. Having identified correspondences, we indicate them by drawing a double-headed arrow between corresponding components as in figure 21.8b.

The third step in JSP involves taking the corresponding component data structures and combining them into a single box in the program structure. Then the non-corresponding boxes are taken from each data structure and added to the program structure in turn while preserving the original hierarchy. This is illustrated in figure 21.9.

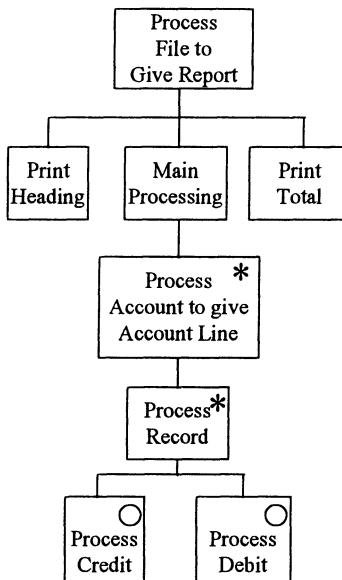
The fourth step is to list program operations such as terminating the program, opening and closing files, performing calculations, reading records, etc. Having listed the operations, the next step is to allocate them to appropriate points in the program structure. Strictly speaking, operations can only be allocated to sequences. Therefore, this may entail adding dummy boxes to the program structure. Two questions must be asked for each operation in turn:

1. With which component box or boxes is the operation associated?
2. Whereabouts in the sequence does it belong – beginning, end, or elsewhere?

Hence, we might allocate the various operations in the following way:

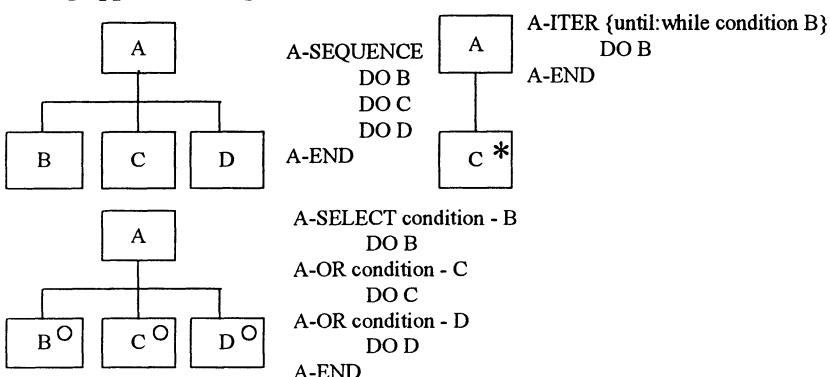
- A) 1) stop
- B) 2) open FILE, REPORT  
3) close FILE, REPORT
- C) 4) print title  
5) print account line  
6) print total
- D) 7) ACCOUNT = ACCOUNT + CREDIT  
8) ACCOUNT = ACCOUNT - DEBIT  
9) TOTAL = TOTAL + ACCOUNT

- E) 10) read FILE record  
 F) 11) ACCOUNT = 0  
 12) TOTAL = 0  
 13) store account number



**Figure 21.9: Banking Program Structure**

The fifth step is to write the schematic logic or structured text. The syntax for each of the structured concepts is illustrated in figure 21.10. Schematic logic is a language-independent pseudo-code designed to translate the program structure diagram into a form more amenable to coding. The schematic logic for the banking application is given below:



**Figure 21.10: Schematic Logic**

```
A - SEQUENCE
  open FILE, REPORT
  read FILE record
  TOTAL = 0
  B1 - print title
  B2 - ITER until end of file (FILE)
C - SEQUENCE
  D1 - ACCOUNT = 0
  store account number
  D2 - ITER until account number <>> stored account number or end of file (FILE)
    F1 - SELECT credit record
      G1 - ACCOUNT = ACCOUNT + CREDIT
    F1 - OR debit record
      G2 - ACCOUNT = ACCOUNT - DEBIT
    F1 - END
  F2 - SEQUENCE
    read FILE record
    TOTAL = TOTAL + ACCOUNT
  F2 - END
  D2 - END
  D3 - print account line
C - END
B2 - END
B3 - print total
close FILE, REPORT; stop
A-END
```

### 21.11 Goronwy Galvanising Case Study

Figure 21.11 represents a structure diagram for a key process in the new computerised system to be built for Goronwy – the production of a despatch advice. This process is built from a sequence of print header, produce despatch line and print footer. Note that the produce despatch line iterates for each completed job. The production of pseudo-code from this diagram is left as an exercise for the reader.

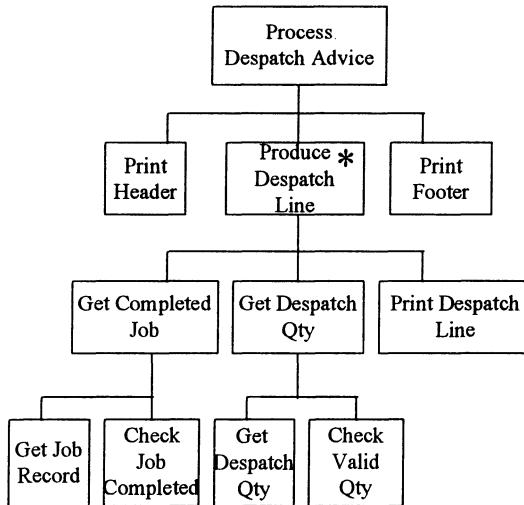


Figure 21.11: Structure Diagram - Produce Despatch Advice

## 21.12 Conclusion

In this chapter we have considered two contrasting methods for structured program design: Yourdon and Constantine's functional design and Jackson's data-oriented design. Functional design works from other techniques such as DFDs and data dictionaries. Jackson's technique is less easy to place within a standard systems development methodology. Probably for this reason Jackson has created his own.

However, to use JSP effectively the information systems engineer must have a clear idea of data structures. This suggests that techniques such as E-R diagramming and normalisation have a useful role in preparing for data-oriented program design.

## 21.13 References

- Booch G. (1990). *Object-Oriented Design with Applications*. Benjamin-Cummings. Redwood City, Calif.
- Jackson M.A. (1984). *Principles of Program Design*. Academic Press, New York.
- Yourdon E. and Constantine L.L. (1979). *Structured Design: Fundamentals of a Discipline of Computer Program and System Design*. Prentice Hall, Englewood Cliffs, NJ.

### 21.14 Exercises

1. Distinguish between a structure chart and a structure diagram.
2. Compare ELHs and JSP. What do you think are the major similarities and differences?
3. Compare the syntax of structured English as discussed in chapter 15 with the syntax of schematic logic.
4. Create a structure chart from the DFD in figure 21.12.
5. Discuss whether functional and data-oriented program design are suitable for an object-oriented approach to programming.

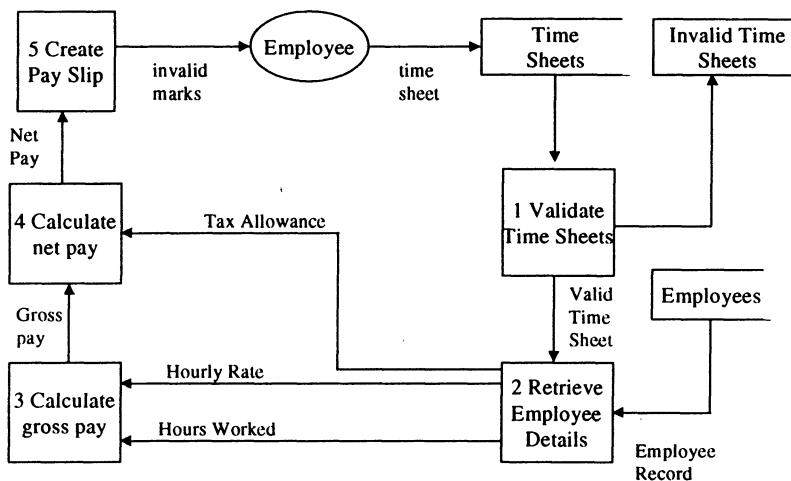


Figure 21.12: Structure Chart Problem

# **22 Object-Oriented Analysis and Design**

## **22.1 Introduction**

Object-Oriented (OO) is definitely having an impact upon systems analysis, systems design, programming (chapter 8), and most recently database systems (chapter 10). The main aim of this chapter is to portray how conceptual modelling as applied to database systems can move relatively painlessly into the domain of OO (Beynon-Davies, 1992). We will discuss how OO analysis, an approach primarily directed at the building of applications in procedural or object-oriented languages, is equally relevant to the development of database systems.

## **22.2 Approaches to OO Analysis and Design**

Object-orientation is becoming a ubiquitous paradigm in modern-day computing. In this respect it has many of the features of the structured paradigm described in chapter 21. Just like the structured movement that influenced approaches to analysis, design and programming, so object-orientation has been associated with distinct approaches to these three classic phases of development work.

Most paradigms (chapter 7) are based on a small number of firmly held principles. One of the principles which is relevant to most of the contemporary literature on OO analysis is that object-orientation is an attempt to heal the divide between process analysis methods and data analysis methods.

Proponents of OO rightly point to the deficiencies of each of these perspectives on an information technology system. Process analysis techniques such as data flow diagramming and functional decomposition over-emphasise the functional or dynamic side of an IT system. In contrast, data analysis techniques such as normalisation and E-R diagramming over-emphasise the data or static side of an information system.

Objects are meant to represent both a structural (data) and a behavioural (process) perspective on information technology systems. Hence, the use of such constructs should facilitate the production of more integrated and intelligible system specifications. However, few of the so-called OO analysis and design methods divorce themselves entirely from the ‘structured’ battles of the 1980s. Most of the current OO methods pay lip service to having some ancestry in the process-oriented or data-oriented analysis and design traditions. What we might call process-directed OO methods build objects out of clusters of behaviours and functions (Gibson, 1990). In contrast, data-driven methods, which seem to be the largest set, start with an analysis of structure and introduce functions later (chapter 30).

The dominance of the data-directed approaches to OO analysis and design has led some people to portray OO analysis and design as little more than extended data modelling (Eckert and Goulder, 1994).

### 22.3 Data Modelling and Object Modelling

Data modelling using the E-R approach has, as we have described it in chapter 16, much in common with object modelling (Blaha *et al.*, 1988). In this chapter we consider some of the common threads between entity-relationship and object-oriented analysis and discuss some proposals for extending entity-relationship modelling into an object modelling approach.

In chapter 16 we defined an entity as being some thing of interest to an organisation which has an independent existence. This abstract definition can serve equally well for defining objects. The major difference between entities and objects lies in the way we apply these constructs in the modelling process. Entities are primarily static constructs. An entity model gives us a useful framework for painting the structural detail of an information technology system. In other words, an entity model usually translates into data structures and relationships between data structures in the data management layer of some information technology system.

Objects however have a more ambitious purpose. An object is designed to encapsulate both a structural and a behavioural aspect. In other words, an object model gives us a means for designing not only a data structure but also how that data structure is to be used.

The easiest way to begin to build an object model is to exploit some of the inherent strengths of entity-relationship modelling and extend them with structural and behavioural abstractions. In this sense, an object model is a data model plus structural and behavioural abstractions.

### 22.4 Structural Abstraction

It is important to reiterate the distinction made between objects and object classes. An object is an instance of something. An object class is a grouping of similar objects. Hence, Paul Beynon-Davies is an object, Lecturer might be an appropriate object class for this object.

Three types of relationships or relations can occur in an object model: association relationships, generalisation relationships and aggregation relationships. Association relationships were considered in chapter 16. Relationships on an E-R diagram are relationships of association. In this chapter we consider generalisation and aggregation relationships. These relationships are sometimes referred to as abstraction relationships, or mechanisms, because they provide a means of hiding detail in abstraction hierarchies.

## 22.5 Generalisation

Generalisation is the process of extracting common features from a group of object classes and suppressing the detailed differences between object classes. In practice, generalisation allows us to declare certain object classes as subclasses of other object classes. For instance, CurrentAccount and DebitAccount might be seen as subclasses of a BankAccount object. In this way a generalisation hierarchy may be built up in which classes further up the hierarchy cover more real-world phenomena than classes lower down the hierarchy.

### 22.5.1 *Inheritance*

The important consequence of this process of generalisation is that objects lower down in the generalisation hierarchy are said to inherit the attributes and relationships of objects higher up in the hierarchy. Hence, CurrentAccount will inherit the currentBalance attribute of BankAccount, and both CurrentAccounts and DepositAccounts will inherit the class BankAccount's relationship with the class Customer.

### 22.5.2 *Generalisation, Cardinality and Optionality*

A generalisation relationship is always characterised by the same optionality and cardinality. At the superclass level, the optionality is always optional and the cardinality is always one. At the subclass level optionality is always mandatory and cardinality is always one. In other words, if BankAccount generalises CurrentAccount then every CurrentAccount is a BankAccount but not every BankAccount is a CurrentAccount.

### 22.5.3 *Generalisation Lattices and Multiple Inheritance*

Many applications of the concept of generalisation do not fall into neat hierarchies. In such cases we speak of a generalisation lattice. In other words, a given object class may be a subclass of more than one superclass. Such an object class is then said to be subject to multiple inheritance. It inherits the attributes and relationships from more than one superclass. A DepositAccount class, for example, could be said to be a subclass of a BankAccount class and an Investment class.

### 22.5.4 *Specialisation*

The opposite of generalisation is referred to as specialisation. Specialisation is the process of creating a new object class by introducing additional detail to the

description of an existing object class. For instance, CurrentAccount is a specialisation of BankAccount.

#### ***22.5.5 Generalisation and Classification***

A distinction is sometimes made between generalisation and classification (Brachman, 1983). Generalisation can be considered as the process of extracting from one or more object classes the description of a more general class. Classification involves grouping objects that share common characteristics into a class object. For instance, the statement P Beynon-Davies is a Lecturer indicates classification. The opposite process to classification is said to be instantiation. For example, P Beynon-Davies would be said to be an instance of a Lecturer.

#### ***22.5.6 Partial and Covering Subclasses***

It is useful to make a distinction between partial and covering subclasses. If subclasses are partial then other subclasses can be included. If subclasses are covering then no further subclasses are permitted. Hence, if we regard Administrator and Lecturer as partial subclasses of UniversityStaff then other subclasses are possible. If these subclasses are covering then lecturers and administrators would be the only types of staff permitted in the university.

#### ***22.5.7 Disjoint and Overlapping Subclasses***

Disjoint subclasses do not overlap. Hence, CurrentAccount and DepositAccount are disjoint subclasses of BankAccount. A BankAccount cannot be both a CurrentAccount and a DepositAccount. However, we can conceive of real-world situations where the concepts embodied in object classes do overlap. Hence Administrator and Lecturer are two overlapping subclasses of UniversityStaff since Lecturers can act as Administrators. If all subclasses in an object model are disjoint we have a strict hierarchy of classes. If some are overlapping we have a lattice structure. Overlapping classes are another facet of multiple inheritance.

#### ***22.5.8 Graphical Notation***

One of the most natural of notations is to indicate generalisation on an extended E-R diagram by drawing nested boxes (see figure 22.1) (Harel, 1988). Overlapping boxes can then indicate disjoint generalisation. Boxes that do not completely fill the superclass box can represent partial generalisation. Hence, the way in which the subclasses of Employee are drawn in figure 22.1b indicates that these are disjoint, covering subclasses.

## 22.6 Aggregation

An aggregation relationship occurs between a whole and its parts (Smith and Smith, 1977). An aggregation is an abstraction in which a relationship between objects is considered a higher level object. This makes it possible to focus on the aggregate while suppressing low-level detail. For example, a FinancialPortfolio can be considered as an aggregate of securities, insurance policies, and savings accounts. A country can be considered an aggregate of regions that are aggregates of counties that are aggregates of districts, and so on. In this way, an aggregation hierarchy can be assembled.

The opposite of aggregation is decomposition. That is, the process of decomposing an object class into its constituent parts.

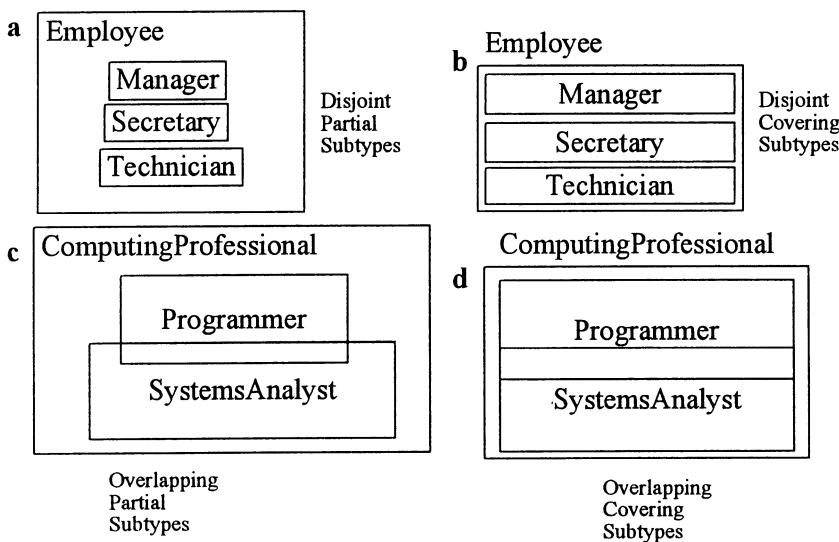


Figure 22.1: A Notation for Generalisation

### 22.6.1 Difference between Aggregation and Generalisation

It is possible to distinguish between aggregation and generalisation in the following way. If two classes are defined in terms of a generalisation relationship then both subclass and superclass effectively refer to the same thing. Hence if we state that 434692 is a CurrentAccount then we are also saying that 434692 is also a BankAccount. When two classes are defined in terms of an aggregation relationship they refer to different things. A Share is a distinct thing from a FinancialPortfolio; a Car is a distinct thing from a Wheel.

### 22.6.2 Object Classes and Attributes

An object class could be considered as an aggregation of attributes. Similarly an object could be considered as an aggregation of its properties. We prefer to consider the relation between an object class and an attribute to be an instance of an attribution relation.

### 22.6.3 Graphic Notation

Graphically we may depict aggregation as a forked line where the class at the head of the fork is the whole and the classes at the fork ends are the parts. Figure 22.2 illustrates a sample aggregation relationship.

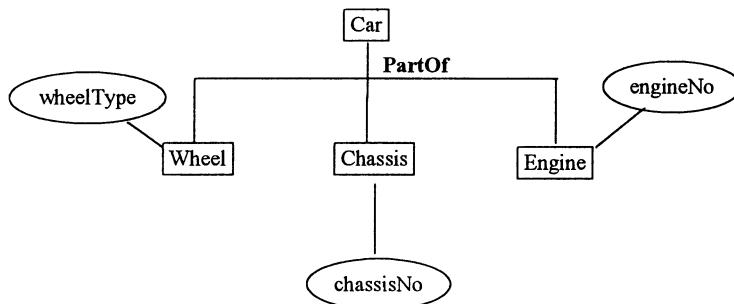


Figure 22.2: Aggregation

## 22.7 Behavioural Abstraction

Behavioural abstraction is included in an object model in terms of methods. The difference between an entity and an object class lies in the specification of methods associated with an entity.

We turn an entity into an object class by first drawing an interface around it. A rounded box represents this. Then we write within the interface the names of the most important methods defined for the object. In figure 22.3 we represent the classes Customer and BankAccount. Six methods are defined for these classes:

enrolCustomer, deleteCustomer, openAccount, creditAccount, debitAccount, closeAccount.

In a truly OO manner transactions impact upon data (attributes) through messages. Hence, a basic form of behaviour would involve sending a message to a class and invoking a method of that class such as openAccount.

Methods may also be used to incorporate integrity constraints. Hence, the enrolCustomer method might perform a check to see that no Customer currently exists with the specified details.

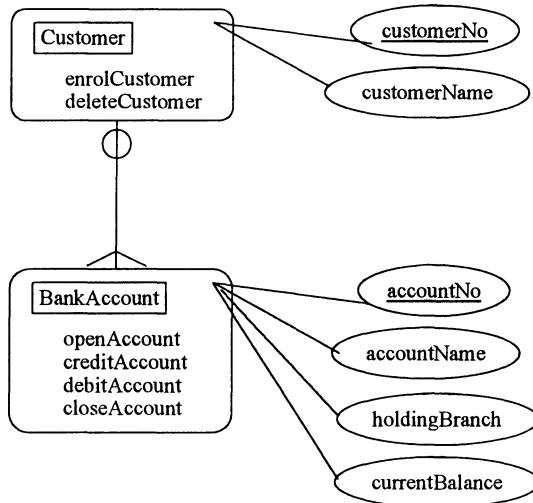


Figure 22.3: A Simple Object Model

## 22.8 Object Life Histories

In the section above we illustrated how we can expand an E-R diagram to contain behavioural information. However, an object model of this form can only tell the developer the names of the major methods contained within each object. It gives us no idea of how methods relate both within a given class and between classes. In this section we look at how one technique widely used in conventional structured analysis, the entity life history, can be adapted to the needs of object modelling.

### 22.8.1 Entities in Flux

From a traditional data modelling point of view, the entities on an E-R diagram are usually thought of as static or structural concepts. An entity model represents a time-independent slice of reality.

An entity may however be considered from a contrasting point of view. An entity is realistically in a state of flux. It is the subject of a large number of processes or events that change its state. In the Stock Market, for instance, the entity *share* may proceed through a number of different states: it is first issued, then it is traded, finally it is withdrawn from the market. An ELH is a diagrammatic technique for charting the usage of a particular entity by the processes or events making up an information system. An ELH is a technique that relates events to states (chapter 20).

### 22.8.2 Events

An event is something that happens at a moment in time. For conceptual purposes however we assume that an event has no duration. Events may be related in that one logically precedes or follows another. Events may also be unrelated, in which case we say they are concurrent. In the stock market, a typical event might constitute a deal made between an investor and a financial intermediary. An event concurrent with this one might be the issue of a new share.

Every event is unique. However, we may group events into event classes based on common features. Event classes are organised hierarchically in the same way as we organise object classes hierarchically. For example, *Student Andrew James enrols on Module relationalDatabaseSystems* and *Student P Davies enrols on Module relationalDatabaseDesign* are both instances of the event class *Student enrolledOn Module*.

### 22.8.3 States

A state is a moment in the life of an object. When people talk of states they are really referring to state classes. A state class is an abstraction of the values of objects and the relationships of objects. For instance, the state of a BankAccount might be characterised as being within balance or overdrawn. It is within balance if the current balance of the account is greater than zero, and overdrawn otherwise.

A state is the response of an object to input events. For instance, various debits from a BankAccount may cause it to become overdrawn. A state represents the interval of time between two events impacting upon an object. Events represent points in time; states represent periods of time. A state therefore has duration; it occupies an interval of time.

### 22.8.4 State Transition Diagrams

To model the behaviour of objects we need some notation that relates states and events in some form of history. In this chapter we shall illustrate the use of the notation of state-transition diagrams for this purpose.

A state transition diagram documents a view of the world in which each object class in a database system can be regarded as a finite state machine. A finite state machine is a hypothetical mechanism that can be in one of a finite number of discrete states at one time. Events occur at discrete points in time and cause changes to the machine's state.

Note the assumption that change is not continuous. Change is modelled as a series of discrete events. This assumption is in fact implicit in the OO paradigm via its adoption of the ideas of methods and messages.

Figure 22.4 illustrates a simple state transition diagram that documents the changes possible between the states of the class *module*. The arrows indicate the only allowable transitions between states. Hence, a student can only be assessed for a module in which he or she has enrolled.

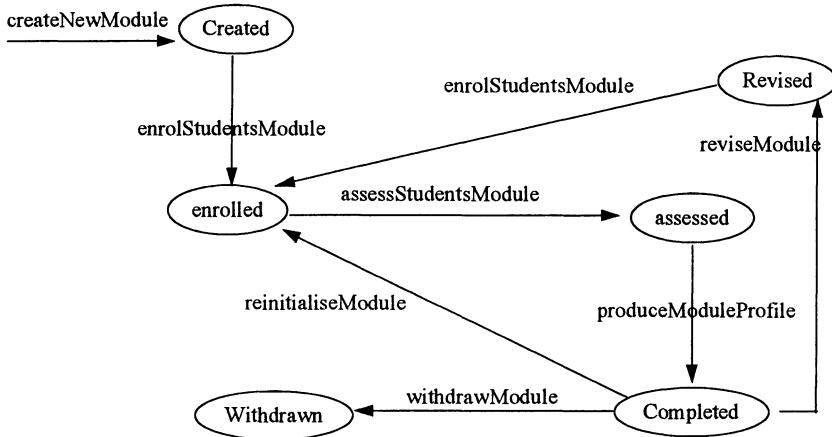


Figure 22.4: State Transition Diagram

## 22.9 Specifying Methods

The method names indicated on a class diagram represent part of what is known as the signature of a method. The signature should specify the name of the method and any parameters needed to be passed to the method. For instance, suppose we attach the following method to the *Student* class:

```
enrolStudent(studentNo: Student, courseCode: Course)
```

In this case, `enrolStudent` is the name of the method and `studentNo` and `courseCode` are the two parameters. These two parameters are both identifiers defined on the respective classes of *Student* and *Course*.

The class diagram does not provide any indication of the logic of the method. The body of this method might be expressed in terms of a series of conditions or constraints that an instance of the method must satisfy, and a series of actions that proceed. This is similar to the concept of an If-Then rule discussed in chapter 14. Hence, the `enrolStudent` method might be specified in pseudocode as follows:

```

body enrolStudent(courseCode: Course)
conditions
student exists
courseCode exists
roll of course not exceeded
actions
create new instance of Enrolment class
update Student dateOf Enrolment

```

## 22.9 Composing an Object Model

It is impracticable for an object model to be specified on a single piece of paper. We would expect an object model to be specified as a series of related parts. For instance, figure 22.5 represents an E-R diagram for a university domain. This sets the context for the object model. It details the major classes in the domain and the major association relationships.

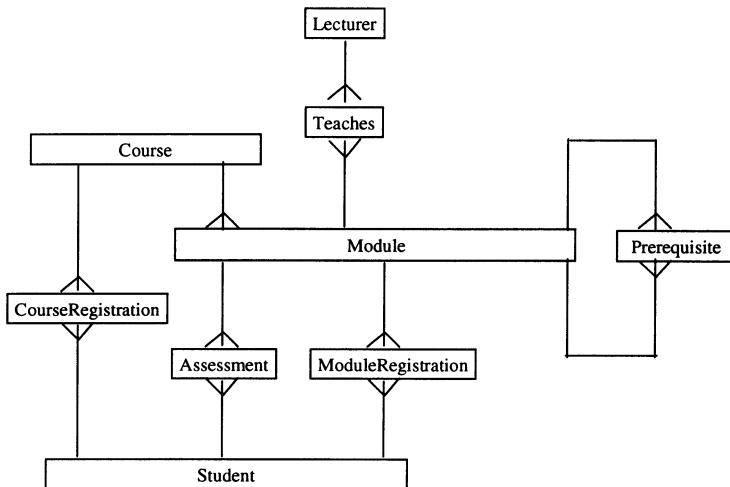


Figure 22.5: E-R Diagram for the University Domain

Figure 22.6 illustrates how we would add generalisation hierarchies to the object model. Here, two classes from the context diagram, Course and Student, have been elaborated in terms of major subclasses.

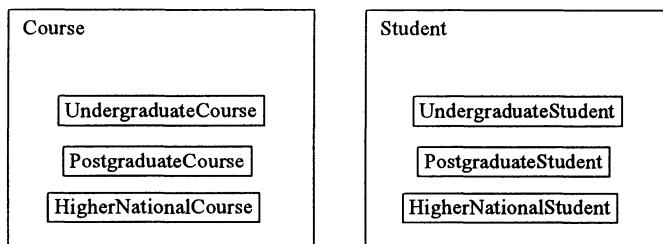


Figure 22.6: Generalisation in the University Domain

Figure 22.7 provides a series of examples of classes with associated methods. Note how most of the methods are constructor and destructor methods.

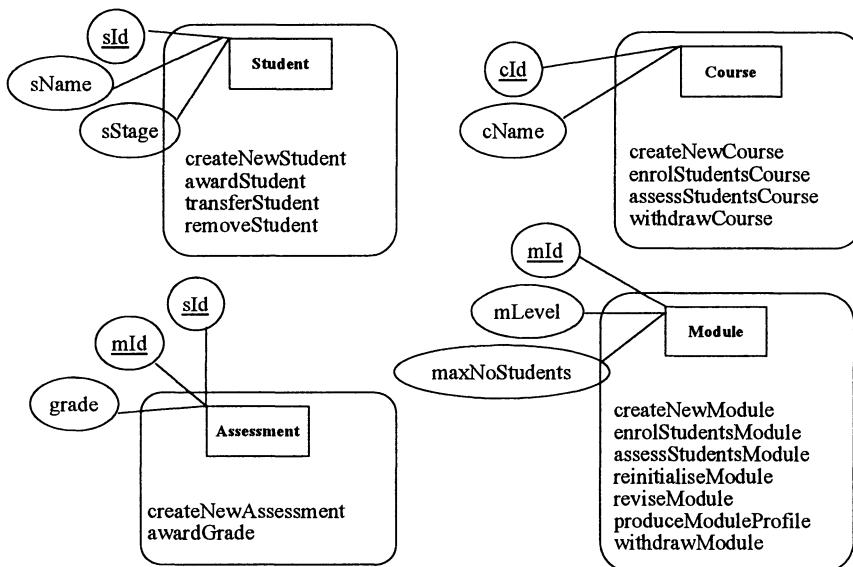


Figure 22.7: Class Diagrams

## 22.10 Conclusion

In this chapter we have considered extending classic E-R diagramming with two types of extensions: structural abstraction and behavioural abstraction. Structural abstraction incorporates the relationships of generalisation and aggregation. Behavioural abstraction incorporates the concepts of methods and messages.

## 22.11 References

- Beynon-Davies P. (1992). 'The Realities of Database Design: the sociology, semiology and pedagogy of database work'. *Journal of Information Systems*. 2. 217-222.
- Brachman R.J. (1983). 'What ISA is and is'nt: an analysis of taxonomic links in semantic networks'. *Computer*. 16(10). 30-36.
- Eckert G. and Golder P. (1994). 'Improving Object-Oriented Analysis'. *Information and Software Technology*. 36 (2). 67-86.
- Harel D. (1988). 'On Visual Formalisms'. *CACM*. 31(5). 514-529.
- Smith J.M. and Smith D.C.P. (1977). 'Database Abstractions: Aggregation and Generalisation'.

## 22.12 Exercises

1. Indicate how generalisation relationships might apply to lecturers.
2. What aggregation relationships might be identifiable in a university domain?
3. Identify suitable methods for the class Lecturer.
4. Specify each method in terms of conditions and actions as suggested in chapter 14.
5. Attempt to draw state-transition diagrams for the classes Student and Lecturer.

# **23 User Interface Design**

## **23.1 Introduction**

What is the user interface? In one sense we might define the user interface as everything concerning the human side of information systems, an area referred to as human factors or human-computer interaction (HCI) (Preece *et al.*, 1995). Hence, all those topics related to the social dimension of information systems development might be seen as having a bearing on the user interface.

In another, more limited sense, the user interface is normally defined in terms of screen-based interfaces to information systems. It is this latter sense of the term which we use in this chapter: a domain sometimes referred to as usability engineering (Nielsen, 1997). In modern information technology systems a significant percentage of the application code is taken up with managing the user interface, and this percentage is likely to increase as user interfaces get ever more complex. For this reason user interface design is becoming ever more important as a technique within the armoury of the IS developer.

In this chapter we discuss three approaches to user interface development: character-based approaches, windows-based approaches and multi-media approaches. This typology is meant to roughly follow a historical progression. Character-based approaches have been used particularly in the context of large, multi-user applications. Windows-based applications have achieved a certain dominance over the last few years. Multi-media is beginning to impact on many interfaces and is likely to form a significant part of interfaces in the future.

## **23.2 Approaches to User Interface Development**

An essential function of most information technology systems is to interact with users. Even a batch process system has to have some mechanism for the user to launch a process and complete a process. No matter how well-built an information system is, its effectiveness is largely governed by its user interface (Eberts, 1994). Hence, the utility of an information system is heavily affected by its usability.

The first step in developing a user interface is to identify exactly where user interaction is needed in a system. One technique adopted by methodologies such as SSADM (chapter 27) is to use process models represented in the form of specifications such as data flow diagrams to identify user interface elements. For each process on the DFD those processes which are essentially human activities are distinguished from those processes which are essentially computer activities. The human related processes are clearly the elements around which the interface needs to be built.

Interaction within such processes is normally portrayed in terms of a dialogue between the user and the IT system. The dialogue is made up of a series of messages between the user and the IT system.

It is useful to distinguish between three major aspects of such a dialogue:

1. *Content*. This refers to the actual messages travelling between the user and the system.
2. *Control*. This refers to the way in which the user moves from one dialogue to another.
3. *Format*. This refers to the actual layout of messages and data on the screen.

Format is largely an implementation-dependent feature of user interface development. The design of such a feature of dialogue may therefore be called physical dialogue design. Content should be a relatively implementation-independent feature. In other words, it is not dependent on the hardware or software configuration chosen for the system. In this sense, the design of such a feature of dialogue may be called logical dialogue design. Control has both implementation-independent and implementation-dependent features. In this sense, it must be addressed both in terms of logical and in terms of physical dialogue design.

Another way in which we can think of dialogues is in terms of the four areas of semiotics described in chapter 1: pragmatics, semantics, syntaxes, and empirics. Format and control are largely aspects of the syntax of signs: the way in which signs are presented and interrelated. Content does however begin to impinge on the semantics of signs: how signs are given meaning. Many of the claims made for icon-based systems, for instance, are that they are in some way more 'natural', i.e. easier to understand and use, than character-based systems.

### 23.3 Logical Dialogue Design

Because of the increasing importance of interfaces, great interest has been shown in providing systematic techniques for dialogue design. We illustrate here an approach to logical dialogue design used within the methodology SSADM. Other approaches are available (Schneiderman, 1997). In SSADM, for each on-line event in an ELH, i.e. an event that has some human activity associated with it, a dialogue needs to be specified.

A logical dialogue outline is developed to represent a dialogue. The outline consists of a diagram documenting the flow of decisions made by the user and by the IT system, as well as a broad description of processing. The diagrams used bear a resemblance to flowcharts used historically for program design.

Figure 23.1 illustrates a sample logical dialogue outline for a display customer details event. The column on the left of the diagram indicates data input to the event or data output from the event. The column on the right

indicates comments on processing. Each decision branch in the dialogue is numbered and commented.

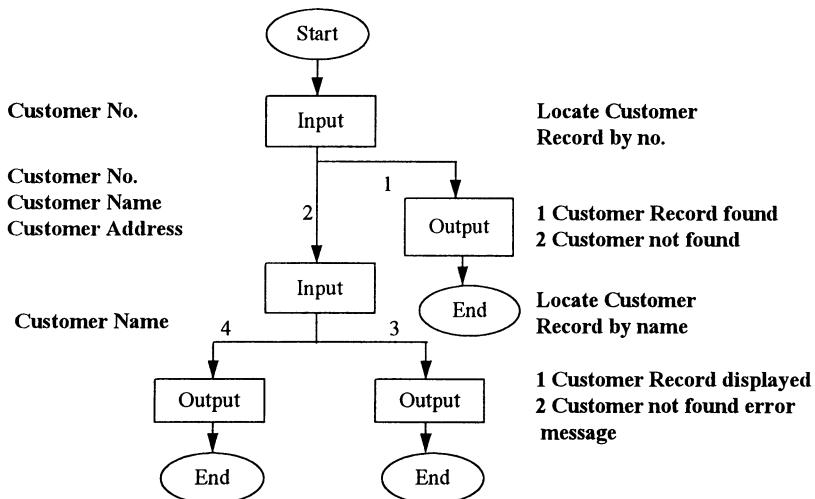


Figure 23.1: A Logical Dialogue Outline

### 23.4 Types of Interface

In terms of format, Schneiderman (1992) identifies five broad categories of interface:

*Menus.* This interface consists of a displayed list of choices. The user selects an item from the list either by pressing some key combination or moving a cursor to the relevant choice and selecting by a mouse click or by typing in some value and depressing the enter key. Figure 23.2 illustrates a menu which might be used to provide access to elements of a system written for Goronwy Galvanising.

*Forms.* These forms of interface are used for data entry and data retrieval purposes. A form is simply a set of fields laid out on a screen, or more readily these days within the context of a window. Each data entry field is normally labelled appropriately. Forms usually have some header area and some area for the display of error messages or prompts. A sample form relevant to the Goronwy system is illustrated in figure 23.3.

*Command language.* In this form of interface the user enters statements in some formal language in order to carry out functions. Historically, command level interfaces were the first type of on-line interface. Operating systems such as MS-DOS and UNIX have command language interfaces. Many DBMS also offer command-level interfaces to SQL (chapter 10).

## Production Documentation System

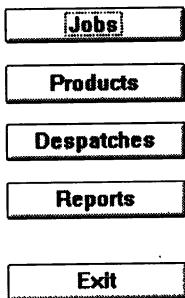


Figure 23.2: A Menu

Delivery Advice No:	Date Received:
jobNo:	
Description:	
Product Code:	Item Length:
Order Qty:	
Batch Weight:	Weight Returned: 0
Qty Returned:	0

Figure 23.3: A Data Entry Form

*Natural language.* So-called natural language interfaces are not truly natural language in the sense that they will not generally accept everyday English input as character strings. They are more accurately described as being restricted language interfaces in that a statement such as: *Give me all the salaries of my employees*, might be acceptable, whereas the statement: *List my employee's salaries*, may not. Such interfaces have achieved some success as front-ends to databases.

*Direct manipulation.* This type of interface is generally associated with icon-based, windows environments (chapter 13). They are referred to as direct manipulation because the user causes events to happen by manipulating graphic objects via mouse-based actions. Such interfaces have now become dominant in most areas of information systems. Figure 23.4 illustrates how a system for Goronwy Galvanising might run in a windows-environment. Here, the user initially selects from a 'switchboard' menu by placing the cursor over the relevant area of the menu and clicking the mouse button. This action causes the data entry form illustrated to appear in a window. Note also the use of a pull-down menu within this form (sometimes known as a list box) to select a value for product code.

## Production Documentation System

Jobs																			
Delivery Advice No:	A3137																		
Date Received:	11/11/97																		
Order No:	13/1193G																		
Description:	Linetels																		
Product Code:	<table border="1"> <tr> <td>Line</td> <td>3</td> </tr> <tr> <td>UL12</td> <td></td> </tr> <tr> <td>UL13</td> <td></td> </tr> <tr> <td>UL135</td> <td></td> </tr> <tr> <td>UL14</td> <td></td> </tr> <tr> <td>UL15</td> <td></td> </tr> <tr> <td>UL16</td> <td></td> </tr> <tr> <td>UL17</td> <td></td> </tr> <tr> <td>UL18</td> <td></td> </tr> </table>	Line	3	UL12		UL13		UL135		UL14		UL15		UL16		UL17		UL18	
Line	3																		
UL12																			
UL13																			
UL135																			
UL14																			
UL15																			
UL16																			
UL17																			
UL18																			
Order Qty:	0																		
Item Length:																			
Batch Weight:	0																		
Qty Returned:	0																		

Figure 23.4: Windowing as applied to Goronwy System

To these five types of interface we may add two further types which are becoming prominent in contemporary systems:

*Multi-media interface.* This is a direct extension of the direct manipulation interface described above in that it employs the same mechanisms for controlling input. The main difference is that rather than having simple menus and data entry forms, a full range of media types is used to build the interface. Many interfaces on the WWW (chapter 13), for instance, now utilise images. Some companies have begun experimenting with the use of animation, video and audio within their interfaces to information systems.

*Virtual reality interface.* In some applications such as those used in simulation it is important for the user to experience a similar environment in terms of an

interface to that available in the real world. Hence, for instance, aircraft simulators have been used for a number of years to train commercial and military pilots. Aspects of this form of interface have now become feasible on the desktop and some organisations are beginning to experiment with the use of such interfaces in the information systems domain. One example here might be an interface which emulates the physical arrangement of publications in a library. Users of such a system would be able to select a given publication by traversing through the virtual space represented in the system.

### **23.5 Guidelines for User Interface Design**

Whichever format is chosen for the interface, the following guidelines are part of established practice:

1. Use consistent and meaningful terminology. This fundamentally means applying a consistent and relevant semantics to a particular domain. For instance, if menu selection is chosen: use a consistent selection mechanism (numbers, characters, etc.); title every menu; align options; have no more than seven options per menu; have a consistent place/mechanism for selection; have a consistent way of displaying error messages; organise menus in a hierarchy which emulates tasks in the system.
2. Design a different interface for each distinct user group. Terms used within the interface should be familiar to the proposed group of users. Naïve users will need different interfaces from experienced users. For instance, naïve users might prefer menu selection, whereas experienced users might prefer a command-line interface.
3. Provide feedback for the user. When the user takes some action, provide some indication to the user of the result of that action.
4. Provide dialogues with a well-defined start, middle and end. This feature, as well as feedback, is sometimes discussed under the concept of ‘closure’. That is, the importance of showing the user when he has successfully completed an operation or why he has been unsuccessful.
5. Have simple, meaningful error messages.
6. Make it easy to correct a mistake. Allowing the user the ability to backtrack to a previous state is normally seen to be useful.
7. Avoid information overload. Do not clutter interfaces with too much information.

### **23.6 Usability Engineering**

Because user interfaces are seen to be a critical element of information systems, a growing literature on the evaluation of the usability of such interfaces has

developed. The discipline devoted to this evaluation is now frequently known as usability engineering.

A number of criteria have been proposed for evaluating user interfaces. Ebert (1994), for instance, distinguishes between:

1. *Effectiveness*. Does the interface do what is required of it?
2. *Learnability*. How easy is it to learn to use?
3. *Flexibility*. How easy is it to adapt the use of the interface?

Many companies specialising in shrink-wrapped software have purpose-built usability laboratories and invest many person-years in assessing the usability of their products. In terms of bespoke software it is considered important to include issues of usability clearly in design. It is also held important to include clear time and resources for assessing usability within the life-cycle.

### 23.7 Goronwy Galvanising Case Study

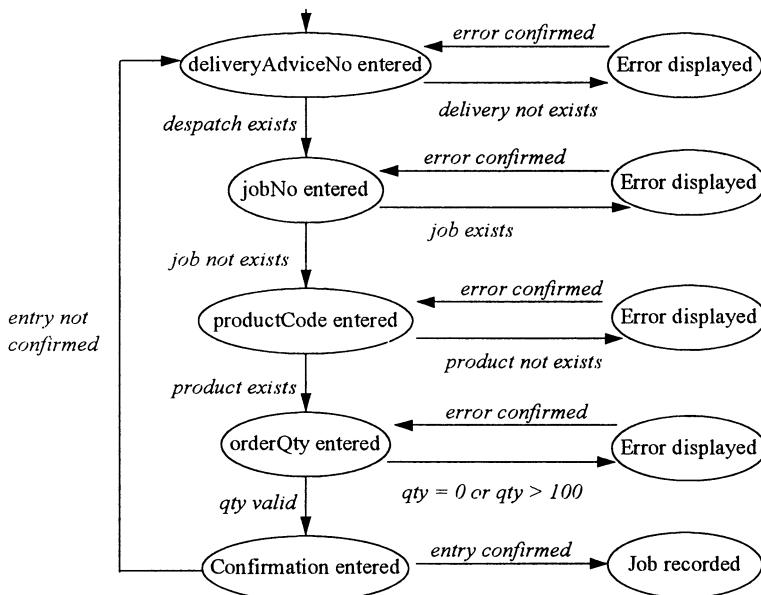


Figure 23.5: A Dialogue Outline for Job Entry

Figure 23.5 represents a logical dialogue design for the job entry process in the proposed information system. We have used a slightly simpler notation than that discussed in section 23.3. This notation is based on the idea of a state transition diagram as discussed in chapter 20. Note how the closure of the dialogue is maintained by informing the user at each point if an error has been made and

returning him or her to a previous state in the dialogue. Note also that at the end of data entry, we request the user to indicate whether he or she wants the record written away (the terminal state) or whether they wish to start again.

Such a dialogue could be implemented in a number of different ways, as a character-based dialogue or as a direct manipulation approach.

### 23.8 Conclusion

User interface development has traditionally been treated as something to be conducted as an afterthought at the back-end of any development effort. The systems analysis is conducted, the system designed, and then some thought is given to in which way the information should be presented or entered.

However, the interface is the one and only means by which potential users can assess the utility of a system. Hence it is extremely important to get the user interface right for the system to be perceived to be a success. For these reasons, usability issues are brought to the foreground in development approaches such as rapid applications development (chapter 29). For this reason also a whole branch of information systems engineering is now devoted to usability issues.

### 23.9 References

- Eberts R. (1994). *User Interface Design*. Prentice Hall. Englewood Cliffs, NJ.  
Nielsen J. (1993). *Usability Engineering*. Academic Press, New York.  
Preece J., Rogers Y. (1995). *Human-Computer Interaction*. Addison-Wesley. Reading, Mass.  
Schneiderman B. (1997). *Designing the User Interface: strategies for effective human-computer interaction*. 3rd Ed. Addison-Wesley. Reading, Mass.

### 23.10 Exercises

1. Direct manipulation interfaces have generally been claimed to be easier to use than character-based interfaces. Why do you think this is so?
2. User interfaces are beginning to exploit the following media: video, animation, photographic images and audio. Discuss the application of each of these media in relation to an application such as that for Goronwy Galvanising.
3. What sort of skills will the information systems developer of the future require in the area of the user interface?
4. Which of the six types of interface discussed is appropriate for which parts of the Goronwy Galvanising application?
5. Produce a logical dialogue design outline for the process of issuing a despatch.

# **24 Prototyping**

## **24.1 Introduction**

In chapter 7 we discussed the waterfall model of information systems development. This model is generally held to be an excellent vehicle for the production of large-scale information systems from well-defined environments. A well-defined environment is usually one in which there is either some manual system waiting to be computerised, or there is an existing computer system that has to be overhauled and perhaps extended.

However, the waterfall model has proved difficult to apply in ill-defined environments. An ill-defined environment is one where there are probably no existing manual procedures or no existing computer system. There may not even be a clear idea of what is required from an information system. For such environments, a more iterative approach to systems development is required. This approach, which we shall refer to as prototyping, is the topic of this chapter.

Prototyping is included as a technique here rather than a complete systems development method. However, prototyping is particularly used in participatory development approaches to developing information systems. Hence, in chapters 28 and 29 we consider the distinctive uses of prototyping in terms of two quite different approaches to participation.

## **24.2 Prototyping and Prototypes**

Prototyping has been discussed since the late 1970s in the IS development literature and has frequently been cast as an alternative information systems development method to the classic waterfall model (Boehm, 1976, 1981). In this context, it appears that the waterfall model is frequently used as a ‘straw man’ to be criticised in terms of the benefits of prototyping approaches. Graham (1989), for instance, cites prototyping approaches as instances of what he calls non-monolithic life-cycle development models. However, Floyd (1984) questions the positioning of prototyping as an alternative method. He maintains that ‘prototyping is not, in itself, a method for systems development.... It should rather be considered as one procedure within systems development that needs to be combined with others.’

It is useful to highlight the difference between the concepts of an information systems prototype and the process of information systems prototyping. Vonk (1990) defines a prototype as being ‘a working model of (parts of) an information system, which emphasises specific aspects of that system’. In contrast, prototyping is an approach to building information systems which uses proto-

types. This activity frequently, but not always, takes the following form (Dearnley and Mayhew, 1983). The information systems engineer, after some initial investigation, constructs a working model which he or she demonstrates to the user. The information systems engineer and the user then discuss the prototype, agreeing on enhancements and amendments. This cycle of inspection-discussion-amendment is repeated a number of times until the user is satisfied with the system.

Note, the use of the word prototype tends to suggest the tentative nature of this artefact. It is early, unfinished or a model of something. Alavi (1984) defines a prototype as being 'an early version of a system that exhibits the essential features of the later operational system'. Floyd (1984) discusses the problem that a prototype literally means 'first of a type'. This makes sense in a mass-manufacturing situation with well-defined requirements. It makes less sense in information systems engineering where normally only one product will be produced and where the desired characteristics of the product may not be well known in advance. There is also no clear consensus about how the final prototype should relate to the final product. As we shall see, there is some debate about whether prototypes should be discarded once built. For these reasons, some people find the term prototype something of a misnomer. They propose alternative terms such as *early-version* or *simulation* to emphasise the clear use to which this system is put.

Prototyping tends to be inherently associated in the information systems development literature with high-level tools for systems development. For instance, Budde *et al.* (1992) summarise the use of fourth generation languages (4GLs) and database-oriented tools for prototyping. However, prototyping is a technique, not just a tool (Hartson and Smith, 1991). The technique has been considered effective even when using pencil and paper or cardboard mock-up prototypes. Muller (1991), for instance, directly advocates low-technology prototyping (in an approach known as PICTIVE) as a reaction to conventional rapid prototyping environments in which 'developers have a disproportionate design impact because of the implicit politics and skill embodied in the technology'. The aim of techniques like PICTIVE is to 'empower' users by using technology that they can understand and manipulate. We discuss this issue in more detail in chapter 28.

#### 24.2.1 An Example of Prototyping

Consider the case of building an information system for the motor claims insurance business discussed in section 2.3. At present, the company runs an entirely manual operation. There is also very little experience of computers in the company. In this type of setting it is very difficult to establish a reasonable set of requirements for the proposed IT system. Staff at the company have very little idea of the potential of the technology. It would therefore seem reasonable

to build a small initial prototype to give them a feel for the functionality and potentiality of the system. A single data entry screen perhaps emulating their existing claims form would be a useful start. The next step might be to build a simple report from the data entered via the screen. The whole purpose of the exercise is to get the company to think about its business and how it might be computerised.

### **24.3 A Taxonomy of Prototyping Practice**

Prototyping as a method or technique within information systems development can be distinguished in a number of ways. Here, we find it useful to characterise this activity in terms of three questions: *what*, *when*, and *how* to prototype.

#### **24.3.1 *What to Prototype***

Floyd (1984) usefully distinguishes between horizontal and vertical prototyping. A horizontal prototype involves shallow (incomplete) development of all or most system functions. A vertical prototype involves deep (complete) implementation of selected features of a system. In practice the degree and extent of horizontal and vertical prototyping will reflect the trajectory of the development. For instance, it is likely that an incremental prototype will be developed in a series of horizontal and vertical moves dependent on the contingencies of a given project.

It is important to recognise that although most of the literature addresses software prototypes, particularly user-interface prototypes and performance/capacity prototypes, business prototypes can also be constructed with very little software contribution to effectively simulate some business processes or functions (*Computing*, 1995). There is some emphasis of this style of prototyping in the recent Business Process Reengineering (BPR) literature (see chapter 25) (Hammer and Champy, 1993).

#### **24.3.2 *When to Prototype***

While recognising that some of the literature portrays prototyping as at the opposite scale to waterfall approaches, one way to distinguish forms of prototyping activity is on the basis of when this activity appears in the standard systems life-cycle (Ratcliffe, 1988):

1. *Early prototyping* occurs at the feasibility or requirements analysis phases. Licher *et al.* (1994) call a prototype used to initiate a software project a presentation prototype. Generally, the purpose of early prototypes is to elicit or validate requirements. Vonk (1990) particularly adheres to this model of prototyping.

2. *Middle prototyping* occurs at the design stage. This form of prototype is generally used to confirm the behaviour of the system in key areas, or to validate key aspects of the design.
3. *Late prototyping* occurs at the implementation or even maintenance phases. This form of prototype is used to investigate key operational parameters, particularly in the area of performance. Luqi (1989) discusses the use of prototyping for evolving systems in this context.

Clearly, the use of prototyping is contingent on project circumstances. Carey and Currey (1989) quote three main reasons for considering when to prototype: when the development is on-line; for all new development; whenever user expectations or requirements are unclear. Klingler (1988) maintains that it is appropriate to prototype for systems that 'are usually dynamic and oriented to transaction processing and user dialogues'.

Hardgrave (1995) (also reported in Doke *et al.*, 1992) conducted a survey amongst 118 IS managers in the United States and identified the top 12 variables reported as influencing the decision to use a prototyping approach:

1. unclear requirements
2. large systems
3. complex systems
4. availability of tools
5. on-line systems
6. project duration
7. feasibility/testing
8. new development
9. user lacks DP experience
10. need user involvement
11. critical system
12. developer's experience

These variables are similar to those reported in other studies. This leads Hardgrave to conclude that a set of factors is being used in practice in industry to decide when to use prototyping. Of course, this may also indicate a certain rationalisation of practitioners' accounts of this activity.

#### ***24.3.3 How to Prototype***

Three main forms of prototype are evident in the literature:

1. A *throwaway prototype* is used to test out some part of a system but then discarded. Budde *et al.* (1992) call this an exploratory prototype. Prototypes for user interface design are frequently discussed in this context (Mulligan *et*

- al.*, 1991). For instance, Lewis *et al.* (1989) discuss the use of graphical user interface (GUI) building tools in the construction of user interface prototypes. A variant of this type is the experimental prototype (Floyd, 1984) where the attempt is to determine the adequacy of some proposed system solution.
2. An *incremental prototype* is one which, by incremental refinement, will form the whole of or part of a delivered system (Budde *et al.*, 1992).
  3. Graham (1989) makes the useful distinction between incremental development and incremental delivery. An incremental prototype appears to comprise a complete system that is developed incrementally; an *evolutionary prototype* forms part of a proposed system which is planned to be delivered incrementally. Hence, all phases of development are conducted for the increment delivered, including coding and testing before moving on to the next increment (Crinnion, 1991), (Lichter *et al.*, 1994).

#### **24.4 Benefits of Prototyping**

It appears to have taken over twenty years for prototyping to become accepted as an IS development approach. Hardgrave (1995) cites his own and Langle *et al.*'s (1984) longitudinal data as evidence of the growth of prototyping in the IS development industry over several years. In 1984, 33% of respondents from a sample of US companies reported using prototyping. This rises to 46% usage in 1987, 49% in 1988 and 61% in 1990. In 1995, Hardgrave himself reports 71% of his respondents from a similar sample using prototyping. In a similar vein, Carey and Currey (1989) report a US survey of 250 companies across sectors in which 75% of respondents reported using prototyping techniques. Carey and Currey's work also maintains that prototyping occurs in all organisational structures, regardless of company size and type. In Gordon and Bieman's (1995) analysis of 39 case studies of prototyping, 33 of the 39 cases reported the prototyping project as a success. 23 of the studies used incremental prototyping; 8 used throwaway.

Therefore, given the apparent ubiquity of this practice, it is not surprising that many benefits are claimed for using a prototyping approach:

1. *Improvements in user communication.* Users and developers can use a prototype as a common reference point for comprehending and developing requirements (Alavi, 1984) (*Computing*, 1995). Users are seen as being better disposed towards a system using a prototyping approach. Carey and Currey (1989) report this as being the most clearly articulated reason for using prototypes.
2. *Better requirements.* Improved functional requirements, improved interaction requirements and easier evolution of requirements are frequent claims (Carey and Mason, 1983), (Budde *et al.*, 1992), (Gordon and Bieman, 1995). A more recent article claimed that prototyping can be used to flush out more than a third of redundant requirements (*Computing*, 1995).

3. *Greater commitment from end-users in a project* (Gordon and Bieman, 1995). Alavi (1984) discusses the way in which prototyping is seen as a means of improving relationships between IS developers and users. Users apparently 'buy-in' to a prototyping project (*Computing*, 1995). The issue of user's 'ownership' of a system is frequently discussed in this context.
4. *Ease of use*. Systems produced by groups using prototypes were judged easier to learn and use than those produced by other methods (Boehm *et al.*, 1984).
5. *Better code*. In a classic experiment conducted by Boehm, code produced by prototyping projects was found to be only about 40% as large as that produced by more conventional, structured methods. This decrease in size was apparently due to the fact that prototyping developers refrained from 'copper-plating' their software (Boehm, 1984), i.e. over-engineering their solution. Also, Gordon and Bieman (1995) report that prototyping code seems to be regularly reported as being more maintainable than code produced by other approaches.
6. *Improvements in the development process*. Carey and Currey (1989) report that the average length of prototyping projects is short – on average about 4.25 months. Boehm's frequently cited experimental study of prototyping reports that prototyping groups accomplished their tasks with 45% less effort than groups using other methods (Boehm *et al.*, 1984). Gordon and Bieman (1995) report that in prototyping case studies analysed, a high percentage of the projects claim a decrease in development effort and an increase in user participation.

## 24.5 Problems with Prototyping

Lichter *et al.* (1994) make the point that many published reports of prototyping are based on investigations conducted in an academic context, e.g. (Alavi, 1984), and hence have restricted validity. A number of survey studies of prototyping have been conducted in the US ((Hardgrave, 1995), Langle *et al.* (1984), (Carey and Currey, 1989)). However, little or no on-site study of prototyping in natural settings appears to have taken place. Most of the material that is available seems to consist of a number of short, anecdotal experiences of prototyping projects. For instance, Carey (1990) and Lichter *et al.* (1994) are representative of this work. Klingler (1988) cogently sums this up with the statement: 'rapid prototypes and their benefits are primarily folklore'.

The case material which is available suggests the following range of problems associated with prototyping:

1. *Different, non-traditional development tools and skills may be required*. The rapidity of prototyping and high user involvement can be stressful for some developers (*Computing*, 1995), particularly because user requirements may change too rapidly. Because of differences between prototyping and con-

- vventional development, management support is seen as particularly important for prototyping projects, as well as the development of a suitable IS infrastructure for this new form of development (Budde *et al.*, 1992).
2. *The cost of development effort*, particularly in the requirements analysis phase, may be greater. *Computing* (1995) quotes a consultant as saying that between 5% and 10% of costs are typically added to the system costs in having early, throwaway, prototyping workshops. Dearnley and Mayhew (1984) confirm that a slightly greater cost is experienced in prototyping projects.
  3. *Project management becomes more difficult*. Prototypes can be difficult to manage and control (Alavi, 1984). Budgeting can be difficult (Gordon and Bieman, 1995). Carey and Currey (1989) speak of the struggles against creeping scope and proliferation of the number and duration of prototyping iterations. Prototypes can get bigger and functionally richer because of user enthusiasm (*Computing*, 1995). A related problem is that prototyping projects seldom produce detailed written specifications (Budde *et al.*, 1992).
  4. *Prototypes can be oversold*. Prototyping can display the tendency to generate unrealistic requirements (Gordon and Bieman, 1995). Managers, on seeing a prototype, may be tempted to release it prematurely to market. Mount (1994) refers to the problem of managing user expectations in the context of rapid application development. This is echoed by Carey and Currey (1989). Carey (1990) cites the problem of convincing users that what they saw in an early prototype is the same as what gets finally delivered.
  5. *Difficult to prototype large systems*. Prototyping as an activity tends to be restricted to small-scale developments (Alavi, 1984) (Gordon and Bieman, 1995). Few organisations in the UK appear to be building large systems using prototyping (*Computing*, 1995).
  6. *Difficulty of recruiting users and maintaining user enthusiasm*. Prototyping depends on the willingness and ability of users to provide useful feedback (Alavi, 1984). User and developer enthusiasm may diminish after a working prototype is provided (Hartson and Smith, 1991). Carmel *et al.* (1993) discuss the difficulties of recruiting the right mix of user types for joint application design (JAD) sessions. They report on how users at JAD sessions tend to be limited to middle managers and not operational workers, who know the work best.
  7. *Difficulty of envisagement*. Recent Scandinavian material has cast doubt on the folk theorem that participation is always a good thing to have in IS development projects (Nielsen and Relsted, 1994). There is some evidence that design teams with heavy user involvement have a tendency to automate existing work practices rather than attempting to envision new ways of working. Licher *et al.* (1994) comment, 'Many developers expect far too much from the users concerning creativity and innovative ideas about the technological design of a prototype'.

## 24.6 Conclusion

Prototyping is an approach to information systems development that has had its greatest support from the tools available on PCs and LANs. It has particularly proved itself on small-scale projects with a short development time-scale. Recently, prototyping has been resurrected with the emergence of object-oriented technology (chapter 9) and the idea of software reuse. Prototyping is also a key feature of rapid application development approaches to building information systems (chapter 29).

## 24.7 References

- Alavi M. (1984). 'An Assessment of the Prototyping Approach to Information Systems Development'. *CACM*. 27(6). 556-563.
- Arthur L.J. (1992). *Rapid Evolutionary Development: requirements, prototyping and software creation*. John Wiley.
- Boehm B.W. (1976). 'Software Engineering'. *IEEE Trans on Computers*. 25(12).
- Boehm B.W. (1981). *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ.
- Boehm B.W. Gray T.E., Seewald T. (1984). 'Prototyping vs Specifying: a multiproject experiment'. *IEEE Trans. Software Engineering*. 10(3).290-333.
- Budde R., Kautz K., Kuhlekamp K., Zullighoven H. (1992). *Prototyping: an approach to evolutionary system development*. Springer-Verlag, Berlin.
- Budde R. Kuhlenkamp K., Mathiassen L., and Zullighoven H. (Eds) (1984). *Approaches to Prototyping*. Springer-Verlag, Berlin.
- Carey J.M. 'Prototyping: alternative systems development methodology'. *Inf and Soft Tech*. 32(2). 119-126.
- Carey T.T. and Mason R.E.A. (1983). 'Information System Prototyping: techniques, tools and methodologies'. *INFOR-Canadian Journal of Operational Research and Information Processing*. 21(3). 177-191.
- Carey M. and Currey J.D. (1989). 'The Prototyping Conundrum'. *Datamation*. June 1.
- Carmel E., Whitaker R.D. and George J.F. (1993). 'PD and Joint Application Design: a transatlantic comparison'. *CACM*. 36(4). 40-48.
- Computing (1995). 'Reverting to prototype'. 25th May.
- Dearnley P.A. and Mayhew P.J. (1983). 'In Favour of System Prototypes and Their Integration into the Systems Development Cycle'. *Computer Journal*. 26(1). 36-42.
- Dearnley P.A. and Mayhew P.J. 'On the use of software development tools in the construction of data processing system prototypes'. In Budde *et al.* (1984).
- Doke E.R., Swanson N.E. Hardgrave B. (1992). 'The Decision to Prototype Information Systems: a pilot study'. *Proc. National Decision Sciences Institute*.

- Floyd C. (1984). 'A Systematic look at Prototyping'. In Budde R., Kuhlenkamp, K., Mathiassen L., and Zullighoven H. (Eds). *Approaches to Prototyping*. Springer-Verlag.
- Giddings R.V. (1984). 'Accommodating Uncertainty in Software Design'. *CACM*. 27(5). 428-434.
- Gordon. V.S. and Bieman J.M. 'Rapid Prototyping: lessons learned'. *IEEE Software*. January. 85-95.
- Hammer M. and Champy J. (1993). *Reengineering the Corporation: a manifesto for business revolution*. Nicholas Brearley, London.
- Harel D. (1980). 'On Folk Theorems'. *CACM*. 23(7). 379-389.
- Hardgrave B.C. (1995). 'When to Prototype: decision variables in industry'. *Inf and Soft. Tech.* 37(2) 113-118.
- Hartson H.R. and Smith E.C. (1991). 'Rapid Prototyping in Human-Computer Interface Development'. *Interacting with Computers*. 3(1). 51-91.
- Klingler D.E. (1988). 'The Ten Commandments of Prototyping'. *Journal of Information Systems Management*. 66-72.
- Kraut R.E. and Streeter L.A. (1995). 'Coordination in Software Development'. *CACM*. 38(3). March. 69-81.
- Kuhn S. and Muller M.J. (1993). 'Participatory Design'. *CACM*. 36(4). 26-28.
- Lewis T.G., Handloser F., Bose S., Yang S. (1989). 'Prototypes from Standard User Interface Management Systems'. *IEEE Computer*. 23(5). May.
- Lichter H., Schneider-Hufschmidt M. and Zullighoven H. (1994). 'Prototyping in Industrial Software Projects – bridging the gap between theory and practice'. *IEEE Trans. Software Engineering* 20(11). November.
- Lipp M. E. (Ed). (1986). *Prototyping: State of the Art Report*. Pergamon.
- Luqi. (1989). 'Software Evolution through Rapid Prototyping'. *IEEE Computer*. 23(5). May.
- Maude T. and Willis G. (1991). *Rapid Prototyping: the management of software risk*. Pitman.
- Mount C. (1994). 'Practical Aspects of Rapid Application Development'. *UNICOM seminar*.
- Muller M.J. (1991). 'PICTIVE – an exploration in participatory design'. *Proc. CHI'91*.
- Muller M.J. (1992). 'Retrospective on a year of participatory design using the PICTIVE technique'. *CHI'92*. May 3-7.
- Nielsen J.F and Relstod N.J. (1994). 'A New Agenda for User Participation: reconsidering the old Scandinavian prescription'. *Scandinavian Journal of Information Systems*. 6(2). 3-20.
- Ratcliffe B. (1988). 'Early and not-so-early Prototyping – rationale and tool support'. *Proc. 12th Int. Computer Software and Applications Conf.* IEEE Computer Society Press. 127-134.
- Vonk R. (1990). *Prototyping: the effective use of CASE technology*. Prentice Hall, Englewood-Cliffs, NJ.

#### **24.8 Exercises**

1. Prioritise the main advantages of prototyping.
2. Prioritise the main disadvantages of prototyping.
3. Contrast prototyping with the waterfall model of development.
4. Prototyping is good for small-scale, PC-based applications, but bad for large-scale, mainframe-based applications. Discuss.
5. In your opinion, how might prototyping be used in the Goronwy Galvanising project?

## ***Part Four***

## ***Methods***

### **Method**

Way of doing something, system of procedure, conscious regularity, orderliness.  
*(Oxford English Dictionary)*

State of order, rule, regularity, discipline.

*(Roget's Thesaurus)*

Whereas a technique is a specific approach to producing some product of information systems development, a method is some organising framework for the application of techniques. We generally prefer the use of the term information systems development method rather than that of an information systems development methodology, since the term methodology strictly speaking means the study of method.

In this part of the book we distinguish between a two different levels of method. The first section considers two distinct approaches to analysing the place of information systems within an organisational context: a process we have called for convenience here business analysis. Chapter 25 covers the topic of Business Process Re-engineering and chapter 26, Soft Systems Methodology.

In the second section we turn our attention to four distinct approaches to the development of information systems. In chapter 27 we address the structured methods which are still used in many companies. These methods mainly adhere to the waterfall model of development discussed in chapter 7. We then contrast this with a more iterative style of development found in both participatory design (chapter 28) and rapid application development (chapter 29). These approaches, although bearing a family resemblance, come from distinctly different philosophical positions. Finally, in chapter 30 we review a number of available methods for object-oriented analysis and design.

## ***Part Four Section One***

### ***Business Analysis Methods***

The term business analysis is used in a rather loose sense to describe a vast range of multi-disciplinary activities. We use it here largely to encapsulate what we might describe as the analysis of the informal systems (chapter 2) of organisations. That is, the process of analysing the overall objectives and needs of an organisation and identifying the place of the organisation within its environment. The intention is usually to in some way redesign the way in which an organisation works and, in particular, clearly identify the most fruitful place for information technology systems in some organisation.

In the same way that we distinguish between a formal and an informal information system we may likewise distinguish between formal and informal systems analysis. Formal or ‘hard’ systems analysis is the process of investigating and documenting the formal information systems of organisations. That is, the system of procedures, rules and regulations present in some organisation. Informal or ‘soft’ systems analysis is the process of investigating and documenting the informal information systems of organisations. That is, the systems of norms, values, attitudes and beliefs that set the context for formal and information technology systems. It is in this latter domain that the term business analysis deserves its place.

In this section we discuss two quite distinct approaches to business analysis. Chapter 25 discusses Business Process Re-engineering. This is an approach which has developed out of the commercial arena and which emphasises a revolutionary approach to organisational change. Chapter 26 discusses Soft Systems Methodology. This is an approach which has developed in the academic arena and which primarily emphasises an evolutionary approach to organisational change.

# **25 Business Process Re-engineering**

## **25.1 Introduction**

In the early 1990s a new term started to appear amongst numerous IS publications – business process re-engineering (BPR). This was initially popularised by a number of influential articles published by Michael Hammer leading to a book of this title written by Hammer and Champy (1993). They define Business Process Re-engineering as: ‘The fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical, contemporary measures of performance, such as cost, quality, service and speed.’ At about the same time another influential, if more conservative book by Davenport and Short (1990), defined BPR as: ‘The analysis and design of work flows and processes within and between organisations.’

Earl (1994) has argued that there are generally two polar viewpoints on BPR: protagonists are convinced that it is a new approach to improving business performance – ‘the new wonder drug’; cynics feel that they have seen it all before in different guises – ‘new wine in old bottles’. As with most things, there is an element of truth in both positions. This chapter attempts to introduce the idea of BPR whilst also attempting to perform a high-level assessment of its relevance to the information systems development area.

## **25.2 What is a Business Process?**

Given that both of the definitions referred to in the introduction refer to processes, this begs the question, what is a business process? Hammer and Champy defines a business process as: ‘a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer’. Davenport and Short (1990) define a process as being ‘a set of logically related tasks performed to achieve a defined business outcome’. Both of these definitions bear a generic resemblance to the definition of a process given in chapter 18. However, note that the emphasis is on organisational or business processes, not necessarily on the processes making up or used to define IT systems.

A related question is how many business processes are there within organisations? Hammer and Champy believe there are just two: manage the product line; manage the order cycle. The UK company BT believes that there are five in terms of its business: manage the business; manage people and work; serve the customers; run the network; support the business.

### 25.2.1 Example

Order fulfilment is a business process. It takes an order as its input and results in the delivery of ordered goods to a customer. The delivery of goods into a customers hands is the value that this process creates.

## 25.3 Background to BPR

In the nature of its role as a product of the information systems industry, BPR has been hyped as a radically new approach to conceiving of the role of business information systems. In reality, of course, BPR can be seen as having a natural ancestry in a number of background areas and issues. In terms of the industrial context, BPR can be seen to have clear links with the debate about organisational structures and business process improvement.

Hammer and Champy argue that organisations have been traditionally structured on the basis of clear, hierarchical and departmental units. Work has consequently been based on the idea of fragmentation of tasks, with control exercised centrally. This form of organisational structure was well suited to a mass-market environment with delineated, local forms of competition and slow rates of technological change. It is held to be unsuitable to the modern market environment characterised by customised markets, intense and diverse global competition, and rapid rates of technological change.

Davenport sees the background to BPR in the ideas underlying Business Process improvement. In this view, IT change enables business change which in turn enables economic change. IT is seen as an important, but not the only enabler of business change. Radical business change is seen as an important enabler of changes in competitive performance.

In terms of the academic context, BPR seems to be related to three established elements much stressed in the IS literature (Earl, 1994):

1. Business change. BPR is another case of socio-technical systems thinking.
2. Change management. BPR has much in common with the elements proposed in the Management in the Nineties research programme (chapter 38).
3. Systems analysis. BPR resurrects the process concept and exploits many of the established tools of systems analysis.

BPR also seems to incorporate three new elements:

1. The idea of process and its importance to organisations.
2. IT as a key enabler of business performance.
3. Transformation via a revolutionary, top-down approach, as compared to more evolutionary, bottom-up approaches such as continuous process improvement.

In terms of the IS context, BPR demands a change of focus. The traditional view of IS development focused solely on the technology and on supporting the existing functional division of the organisation. The traditional IS development strategy also only offers incremental improvements in business performance. In BPR information systems are considered as affording the opportunity to radically redesign organisational work. As a consequence, radical redesign of work can offer radical improvements in organisational performance.

#### **25.4 Example: IBM Credit**

IBM Credit (Hammer and Champy, 1993), an organisation devoted to the financing of customer's purchases of IBM equipment, engaged in re-engineering. We contrast the old with the new process below:

*Old process.* The process was initiated by a salesperson calling in with a request for financing. One of fourteen telephone personnel logged the request on a paper form. The form was then taken to a department, which entered details of the request into a computer system and determined the creditworthiness of the customer. Credit check details were written on to the form and the form was passed to the business practices department that modified the standard form of loan to suit the customer. Any special terms were attached to the form and it was passed to a pricer who determined, with the help of a spreadsheet, the appropriate interest rate to charge the customer. The interest rate was then added to the form and delivered to a clerical department, which turned the information on the form into a quote.

The main problem with this process was that it consumed six days on average. This led to customers being lost in the intervening period.

*New process.* In the new process specialist staff were replaced with generalists. One person called a deal structurer now deals with one straightforward order. A computer system was developed to support the work of the deal structurer. These changes slashed turnaround time from six days to 4 hours.

#### **25.5 Example: Accounts Payable – Ford**

Hammer (1990) describes an accounts payable process at the Ford Motor Company that was re-engineered. This accounts payable department originally employed 500 employees. A competitor's accounts payable department had 5 people. Therefore, Ford set out to reduce the workforce in this area by hundreds.

*Old process.* The process began with the purchasing department writing an order. A copy of this order was then sent to the accounts payable department. Later, when the materials control department received goods, it sent a copy of

the receiving document to accounts payable. Meanwhile the vendor also sent an invoice to accounts payable. This process is illustrated in figure 25.1.

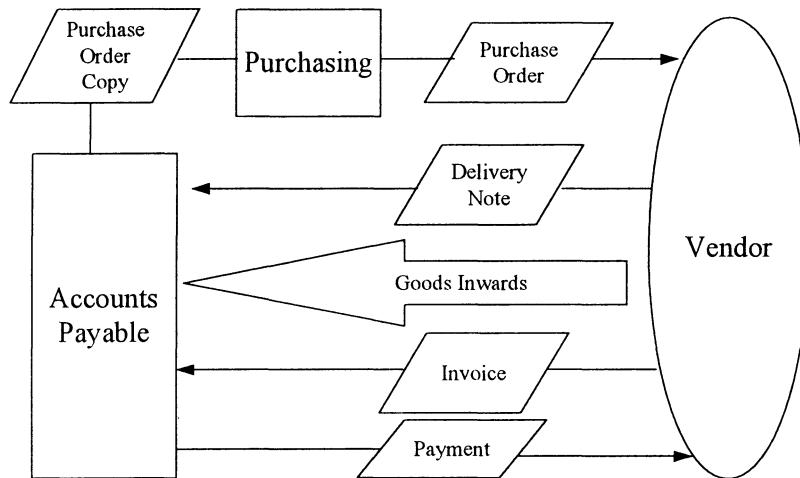


Figure 25.1: Original Process: Ford

The end-result of this meant that the accounts payable department were involved in matching fourteen data items between the receipt order, the purchase order and the invoice before it could issue payment to the vendor. In fact, the department spent most of its time trying to sort out mismatches between these three documents.

*New process.* The new process involved ‘invoiceless processing’. The purchasing department now entered order information into a database. No copy of this order is sent to anyone. When goods arrive at the receiving dock, the receiving clerk checks the material against the outstanding purchase record in the database. If they match, he accepts the goods and payment is automatically sent to the vendor. If they do not match, the order is returned. This means that matching of only three data items is required: part number, unit of measure and supplier code between a purchase order and a receipt order. As a result, Ford achieved a 75% cut in head count. This process is illustrated in figure 25.2.

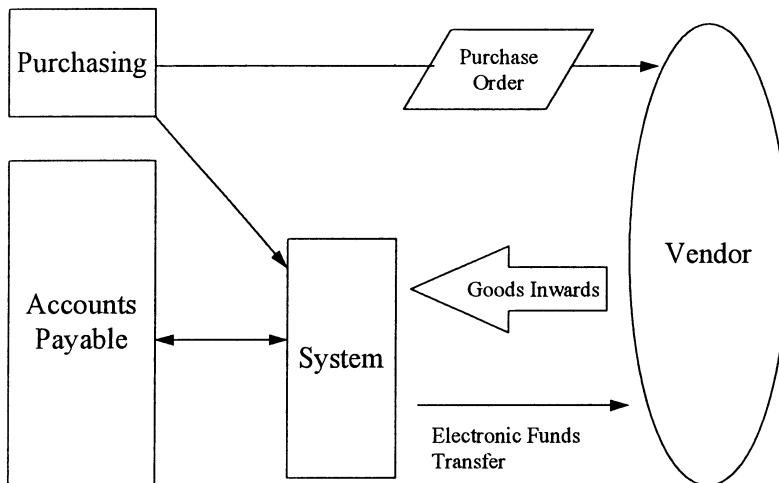


Figure 25.2: Re-engineered Process: Ford

## 25.6 Common Themes

A number of common themes recur in the BPR literature:

1. Several jobs are combined into one. Jobs evolve from narrow and task-oriented to multidimensional.
2. Workers make decisions. Work units change from functional departments to process teams. Processes cross traditional departmental boundaries.
3. The steps in a process are performed in a natural order.
4. Processes have multiple versions tailored to specific inputs.
5. Work is performed where it makes the most sense.
6. Checks and controls are reduced. People's roles change from controlled to empowered. Focus on performance measures; compensation shifts from activity to results.
7. A case manager provides a single point of contact for a customer.
8. Hybrid centralised/decentralised operations are prevalent.
9. Advancement criteria change – from performance to ability.
10. Organisational structures change – from hierarchical to flat.
11. Information technology is a key enabler in process re-engineering. It enables in particular traditionally fragmented activities to be stitched back together.

## 25.7 How to Re-engineer

There is actually surprisingly little guidance as to how to conduct a re-engineering project. Below, we describe a number of high-level stages:

1. Needs enthusiastic support of high-ranking manager.
2. Set up a re-engineering team.
3. Communicate the principles of re-engineering.
4. Develop a process map of the organisation.
5. Choose processes to re-engineer.
6. Investigate and understand processes.
7. Conduct redesign sessions. Challenge assumptions.

## 25.8 Problems with BPR

One of the biggest problems faced by a BPR project is its revolutionary nature. This prompts a quote from the Renaissance philosopher Niccolo Machiavelli (1469-1527) who stated: ‘There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success than to take the lead in the introduction of a new order of things.’

Earl (1994) comments on the way in which IS tend to dominate BPR work because of the systems analysis background of its promoters. However, Moad (1994) discusses how many IS people find it difficult to participate in BPR projects. IS is seen as the necessary enabler of process re-engineering, but IS departments are typically seen as slow in delivering expected promise. IS people frequently take a traditional attitude of automating/fixing existing processes rather than considering radical change. Most analysts and developers are used to creating applications for a single department. They find it difficult to adapt when asked to support newly designed business processes that span several departments.

Hammer states that as much as 70% of BPR projects fail. However, IS is not seen as the primary culprit of such failure; faltering support from upper management sponsors is primarily blamed. BPR projects by their very nature are high risk/high payoff projects.

*Computing* (1995b) reports on some of the conclusions of an Oxford Templeton College study of BPR projects in the UK. Most UK companies seem to be engaging in Business Process Improvement rather than re-engineering exercises. Perhaps, as a consequence, less than a quarter of UK companies surveyed that had completed a BPR project reported appreciable gains from the exercise.

### **25.9 The Death of BPR?**

A number of persons have felt that the problems associated with BPR projects are serious enough to warrant consideration of the possible death of BPR as an approach.

*Computing* (1995a) discusses what it sees to be something of a backlash facing the BPR concept in the UK. Although BPR was not explicitly initiated as a means of slashing headcount and cutting costs, its popularisation coincided with the economic recession of the 1990s. An approach designed to make people more creative and organisational structures more flexible was frequently used as a 'brutal management tool to impose unpopular changes'.

In a similar vein, De-Marco (1995) questions the idea that being lean and mean is necessarily a good thing: 'Becoming lean is not a sign of future health, but of present failure.' He believes that 'Being mean is a failure of intent, a failure to recognise that one can be competitive without being malicious to your own employees and to other organisations'.

### **25.10 Conclusion**

In summary, BPR is the radical redesign of business processes to enable radical improvements in business performance. BPR is important in focusing on the necessary interdependence between IS and organisational work. It proposes considering new forms of organisational structure made possible by innovations in IT. BPR can be seen to have its roots in business change, change management, and systems analysis ideas. It has wedged these with ideas of transformation, IT-enablement and a process view of organisations. BPR projects appear to have suffered a high failure rate and also the idea of BPR has suffered something of a backlash in recent years

### **25.11 References**

- Computing* (1995a). 'Death by a thousand cuts'. 6th July.
- Computing* (1995b). 'Big spenders fail to see benefits of BPR'. 13th July.
- Davenport T.H. and Short J.E. (1990). 'The New Industrial Engineering: information technology and business process redesign'. *Sloan Management Review*. 31(4). 11-27.
- Davenport T.H. (1993). *Process innovation: reengineering work through information technology*. Harvard Business School Press, Boston, Mass.
- De-Marco T. (1995). 'What 'Lean and Mean' really means'. *IEEE Software*. Nov.
- Earl M.J. (1994). 'The New and the Old of Business Process Redesign'. *Journal of Strategic Information Systems*. 3(1). 5-22.

- Hammer M. and Champy J. (1993). *Reengineering the Corporation: a manifesto for business revolution*. Nicholas Brearley, London.
- Hammer M. (1990). 'Reengineering Work: don't automate, obliterate'. Reprinted in the *Information Infrastructure*. Harvard Business Review.
- Moad J. (1994). 'Reengineering: report from the trenches'. *Datamation*. March 15th.
- Moad J. (1993). 'Does Reengineering really Work?' *Datamation*. 1st Aug.

### 25.12 Exercises

1. Consider an organisation known to you. Investigate whether they have engaged in a BPR project in recent times. If they have, was it considered a success, and in what terms?
2. Identify what you feel to be the fundamental business processes at work in a university. What are the boundaries of these processes? What are the fundamental activities involved in these processes?
3. Choose a business process from an organisation known to you. Produce a process map of the current business process. Identify the problems or limitations in terms of this process. Produce a process map of a re-engineered process. Describe some of the benefits of the new process.

# **26 Soft Systems Methodology**

## **26.1 Introduction**

As we have mentioned, there is no one agreed approach to conducting business analysis. Therefore, most of the discussion in this chapter draws upon the work of Peter Checkland and his associates at the University of Lancaster. Over the last couple of decades Checkland has been developing the framework of an approach known as soft systems methodology (SSM) (Checkland, 1987). This method has been adapted to the needs of government departments and renamed Business Analysis Technique (CCTA, 1990). Avison and Wood-Harper (1990) have demonstrated also how the techniques of soft systems methodology can lead into conventional ‘hard’ systems analysis.

## **26.2 Soft Systems Framework**

Figure 26.1 illustrates the essential elements of Soft Systems Methodology. We first must have a situation in everyday life that is regarded by at least one person as being problematical. The situation, being part of human affairs, will be the product of a particular history. Facing up to the problem situation are some ‘would-be-improvers’ of it. These persons are the users of SSM.

Soft systems methodology follows two interacting streams of enquiry: a stream of cultural enquiry, and a stream of logic-based enquiry. Both streams may be regarded as stemming from the perception of purposeful actions (tasks) in the problem situation and various things about which there are disagreements (issues).

On the right-hand side of figure 26.1 is a stream of enquiry in which a number of models of human activity systems are used to illuminate the problem situation. This is accomplished by comparing the models with the perceptions of the real-world situation. These comparisons serve to structure a debate about change.

The left-hand side of figure 26.1 consists of three examinations of the problem situation. The first examines the intervention itself, since this will inevitably effect some change in the problem situation. The second examines the situation as a ‘social system’, the third as a ‘political system’.

The logic-driven stream and the cultural stream interact. Which selected human activity systems are found relevant to people in the problem situation will be determined by the culture in which it is immersed.

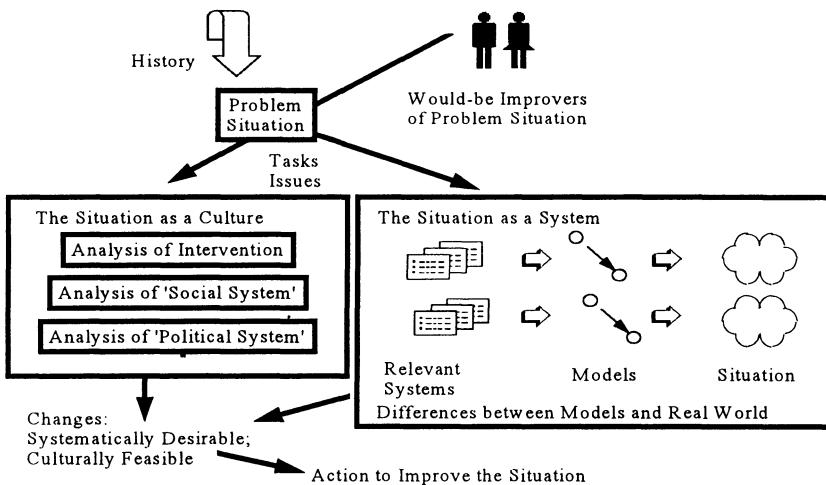


Figure 26.1: Soft Systems Methodology

### 26.3 The Logic-Driven Stream

The logic-driven stream is undertaken using two major techniques for defining relevant human activity systems: root definitions and conceptual models.

#### 26.3.1 Root Definitions

A root definition expresses the core purpose of a human activity system, in terms of input-process-output. Checkland has suggested that most useful root definitions be made out of six elements making up the acronym CATWOE:

- C. customers – the victims or beneficiaries of T.
- A. actors – those who would do T.
- T. transformation – the conversion of input to output.
- W. weltanschauung – the worldview which makes T meaningful.
- O. owners – those that could stop T.
- E. environmental constraints – elements outside the system which it takes as given.

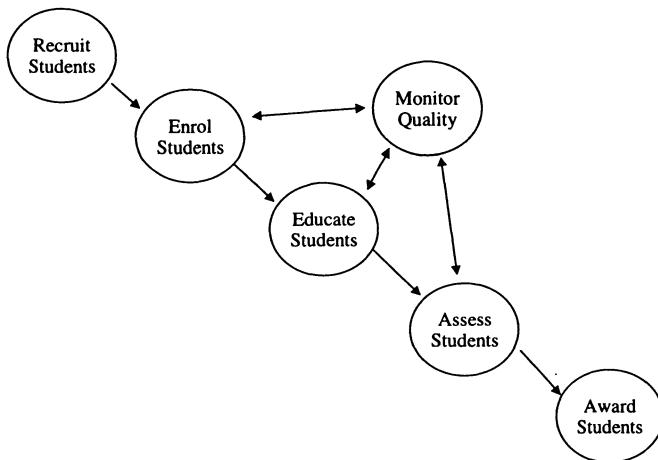
The core of CATWOE is the pairing of transformation with the worldview, which makes it meaningful. For any human activity system there will always be a number of different transformations by means of which it can be expressed, these deriving from different interpretations or worldviews of its purpose. The other elements of the mnemonic add the ideas that someone must undertake the purposeful activity, someone could stop it, someone will be its victim or

beneficiary, and that the system will take some environmental constraints as a given.

Consider a university as an example. A number of stakeholders exist in this organisation, each with a different worldview as to its purpose. For example, a student's worldview might be characterised as follows:

- C. Myself.
- A. Other students, lecturers and administrators.
- T. The process of attending modules, achieving satisfactory assessments and getting a degree.
- W. That higher education is a passage to better job prospects.
- O. The lecturers and administrators.
- E. The British Higher Education System.

Root definitions provide the material for constructing conceptual models. These represent pictorial representations of key relationships between the minimum necessary activities needed to support the key transformation process. The arrows represent dependent relationships between activities. Figure 26.2 illustrates a simple conceptual model drawn for a university context.



**Figure 26.2: An SSM Conceptual Model**

## 26.4 The Stream of Cultural Enquiry

The stream of cultural enquiry in SSM comprises three stages known generally as analysis one, analysis two and analysis three.

1. *Intervention analysis.* This involves analysing the intervention in terms of three roles: the client, the problem-solver and the problem owner. In terms of the client (the person or persons who caused the study to take place), we need

to analyse his or her intentions in causing the study to take place. In terms of the ‘would be problem solver’, we need to understand their perception of why they wish to do something about the situation in question. In terms of the problem owner we must decide whom to take the problem owners to be.

2. *Social system analysis*. This stage uses a simple model of a social system. It assumes a social system to be a continually changing interaction between three elements: roles, norms and values (chapter 2). Each continually defines and redefines itself in terms of the other two. By role is meant a social position that is recognised as being significant in the problem situation. A role is characterised by expected behaviours in it, or norms. Actual performance in a role will be judged in terms of local standards or values.
3. *Political systems analysis*. Processes by which differing interests reach accommodation. Political analysis is made practical by asking how power is expressed in the situation studied. What are the commodities or embodiments of power used in the situation?

#### 26.4.1 Rich Pictures

Each of these three stages of the cultural stream may exploit a diagramming technique known as rich pictures to help understand the particular situation under study. A rich picture is an attempt to loosely illustrate the cultural context of a problem situation. Figure 26.3 illustrates a rich picture drawn for a university.

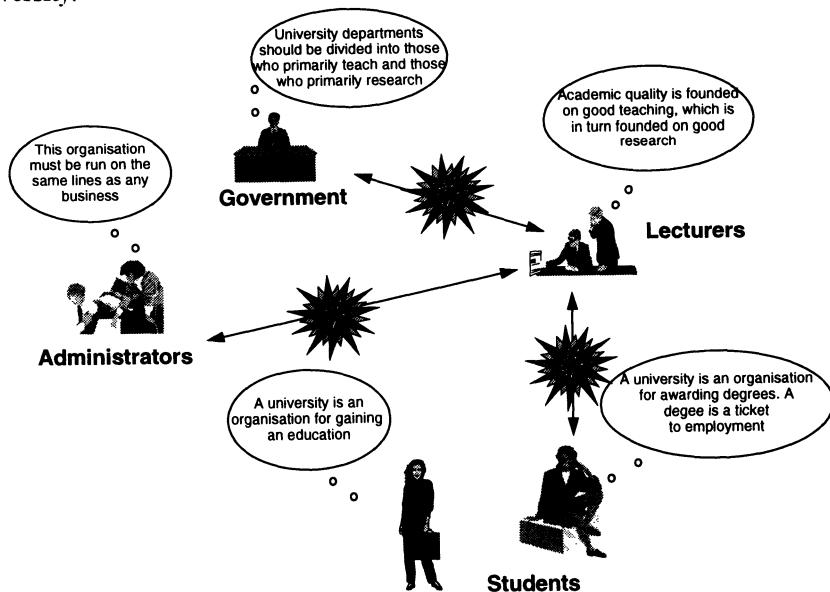


Figure 26.3: A Rich Picture

## **26.5 Action Research**

Checkland and others have come to see SSM as an example of what they have come to call action research. SSM is designed to help individuals do something about a situation that they regard as in some way unsatisfactory. SSM is therefore a research tool for action.

The two streams of SSM converge in a structured debate concerned with defining changes, which will help remove the dissatisfaction. Such changes must be systematically desirable and culturally feasible. The changes must reflect the idea of relevant systems within a cultural context.

Action research is frequently seen as an iterative process. A study is made, decisions about appropriate changes are founded in the study, changes are introduced, a further study is undertaken to evaluate the effects of such changes, and so on.

## **26.6 SSM in the Creation of Information Technology Systems**

Although SSM is portrayed as a technology-free method of systems thinking, Checkland and Scholes have described the method as particularly applicable to computerised information systems. They discuss how process-based and data-based models of information systems can lead on from the conceptual models of SSM. One methodology specifically designed for computerised information systems work, namely Multiview, explicitly uses this approach (Avison and Wood-Harper, 1990, 1997). Lewis (1994) discusses a more recent example of the application of SSM to the problems of data modelling.

## **26.7 Conclusion**

This chapter has concluded that an analysis of the informal system of some organisation is an important first step in any information systems development effort. Methods in this area have to be by their very nature ‘soft’ or interpretative. The most coherently organised example of methods available is Checkland’s Soft Systems Methodology. Some contingency-based methods such as Multiview have incorporated soft systems work. The focus of soft systems methods and the newer area of business process re-engineering are similar in the sense that they both propose conscious redesign of organisational activity to serve some declared objectives. Soft systems work however does not articulate the same profit-directed motives as the overt BPR literature. Also, the philosophical stance of SSM tends to emphasise the evolutionary rather than the revolutionary redesign of organisations.

## 26.8 References

- Avison D.E. and Wood-Harper A.T. (1990). *Multiview: An Exploration in Information Systems Development*. Blackwell Scientific Publications, Oxford.
- Avison D.E. and Wood-Harper A.T. (1997). *Multiview 2: An Exploration in Information Systems Development*. McGraw-Hill, Maidenhead.
- CCTA. (1990). *Managing Information as a Resource*. HMSO.
- Checkland P. (1987). *Systems Thinking, Systems Practice*. John Wiley, Chichester.
- Checkland P. and Scholes J. (1990). *Soft Systems Methodology in Action*. John Wiley, Chichester.
- Lewis Paul. (1994). *Information Systems Development*. Pitman, London.

## 26.9 Exercises

1. Identify two major stakeholder groups in an organisation known to you. Draw up a root definition for each group.
2. Draw a rich picture of the organisation in which you identify the stakeholder groups. Identify key issues of importance to such groups and any areas of conflict.
3. Draw up a conceptual model for the whole or part of an organisation known to you.

## ***Part Four Section Two***

### ***Development Methods***

In this section we address a number of distinct methodological approaches to developing information systems. In chapter 27 we consider the historical inheritance in this area. We review a number of the so-called structured methods, many of which are still in use today. Most of these methods adhere to a waterfall life-cycle, although contemporary versions of some structured methods such as SSADM have attempted to adapt themselves to other life-cycle models.

In chapter 28 and chapter 29 we consider two development approaches which explicitly utilise an iterative life-cycle model. Participatory design has been a movement which has had strong roots in academic work conducted in information systems. In contrast, rapid applications development has emerged from the commercial arena. At the micro level there are certain similarities between the two approaches, particularly in the focus on prototyping and user involvement. However, at the macro level there are key differences between the two approaches.

We conclude the section with a review of a number of extant object-oriented methods. We particularly discuss those OO methods which have been proven to have direct relevance to information systems development issues as compared to other application areas such as real-time systems.

# **27 Structured Methods**

## **27.1 Introduction**

Part 3 was devoted to describing a number of techniques popular in contemporary systems analysis and design. Presenting the techniques as a relatively discrete set of components has hopefully emphasised that the individual systems developer can frequently select appropriate techniques as and when they are needed during the life-cycle of a particular project. This means that because of the large number of choices available, a large number of possible development approaches can be created.

In this chapter we examine the need for and composition of a standard methodology. We concentrate here particularly on a number of what are frequently termed ‘traditional’ methodologies in the sense that they adhere to principles underlying the structured analysis and design movement. Although originating in the 1980s, many organisations still heavily rely on approaches similar to the ones discussed here. We describe the framework underlying three traditional methodologies: Structured Systems Analysis (STRADIS), Information Engineering, and SSADM. We conclude with a feature analysis of the three methodologies chosen.

## **27.2 What is a Methodology?**

Strictly speaking, the term methodology means ‘the study of method’. However, within the world of ISD the term methodology has been used as a synonym for method. An information systems development method or methodology might be defined as being made up of the following primary components:

1. A life-cycle model of the information systems development process.
2. A set of techniques.
3. A documentation method associated with these techniques.
4. Some indication of how the techniques chosen, along with the documentation method, fit into the model of the development process.

Given methodologies may also contain the following secondary components:

1. A strategy as to how development projects are to be managed.
2. A strategy for training new users of the methodology.
3. A strategy as to what support tools can be utilised and where.

All methodologies also work within some philosophy that might be defined

as the set of assumptions about what constitutes information, an information system, and the place of information systems within organisations. Some include a phase of business analysis (part 4 section 1).

### **27.3 Why Do We Need a Standard Development Methodology?**

Any organisation looking to set up a standard methodology for information systems development has a number of choices available:

1. It can develop its own.
2. It can purchase and use one of the many ‘off-the-shelf’ methods.
3. It can purchase an ‘off-the-shelf’ methodology but adapt it to its own purposes.

A standard development methodology is normally employed for a number of reasons. Some of the reasons are outlined below:

1. People employ a method because they believe that it is likely to produce a better end-product than one produced via an *ad hoc* approach. In other words, better information systems are held to result from method-driven work.
2. Developers do not need to spend valuable time creating a new method for each new project. Instead, they can become skilled in the application of techniques within one overarching framework.
3. New members of the development team are more easily trained in the process of systems development.
4. Some people have claimed that the use of a method reduces the level of skills required by developers. This improves development by reducing its cost.
5. The application of a standard method makes project management and the estimating of project time that much easier.
6. A standard method facilitates communication, not only between development staff, but also with end-users.
7. Standard methods encourage the automation of the development process.
8. The improved systems specifications and the production of detailed documentation associated with the use of a method makes the maintenance and enhancement of information systems that much easier.

### **27.4 The Growth of Method**

Methods have arisen from two sources: the commercial world and the academic world.

In the commercial world, methods arose, and continue to arise, out of

consultancy work in information systems development. In the early days, information systems consultancies sold solely the skills of individual developers to client organisations. Each consultant employed his own favourite techniques in his own preferred ways. This made communication between developers somewhat difficult, and the problem of controlling such *ad hoc* development even more difficult. Most consultancy organisations came to realise that the method of information systems development was one of their most important assets that had to be produced as a product, be made consistent, marketed, maintained and continually developed.

Academic methods started life as research vehicles in universities and other research centres. The main aim of such methods was to build techniques upon sound theoretical concepts. Two areas which have proven academic roots is the idea of soft systems analysis (chapter 26) and the important emphasis on user participation (chapter 28) in the development process.

In recent years there has been something of a decline of interest in the use of methods, at least amongst the practitioner community. Although alternative development methods to the structured methods have been proposed, particularly in relation to object-oriented approaches (chapter 30) and to more iterative forms of development such as rapid applications development (chapter 29), the belief in the ability of methods to solve problems with development appears to have suffered a setback.

## 27.5 A Typology of Methods

It is useful to classify the array of traditional methods under three major headings: data-driven methods; process-driven methods; and integrative methods. Although data-driven and process-driven methods may be seen as precursors of the integrative stream, many methods still fall very much into either the data-driven or process-driven camp.

Data-driven methods are those which have primarily arisen out of an overriding interest in database development. Process-driven methods have arisen out of an interest in modelling the functional transformation of data within organisations. As a general abstraction, it is probably true to say that the process-driven approach has had strong support from American software companies and their European associates, while the data-driven approach continues to have strong support in Western Europe.

In the last decade, most of the proffered methods have attempted to integrate the data and process elements of information systems development. Methods like SSADM are clearly in this camp, as are the recent attempts at object-oriented analysis and design (chapter 30).

A number of other features have been discussed in the context of comparing methods (Fitzgerald *et al.*, 1985) such as:

1. *General philosophy.* The set of principles underlying a method. For instance, whether the method takes an objectivist or subjectivist view of reality.
2. *Development process.* The general model of the development process supported. For instance, whether the method adheres to a waterfall, prototyping, evolutionary or hybrid model of information systems development (chapter 2).
3. *Techniques.* For example, E-R diagrams, DFDs, action diagrams, etc. (part 3).
4. *Automated support.* The place of automated support within the method. Whether both front-end and back-end CAISE tools exist to support the activities of the method (chapter 12).
5. *Scope.* Whether it incorporates business analysis (part 4 section 1) as well as information systems development. Whether the method expects the production of computerised systems. Whether the method is defined for use on large or small projects.
6. *Users.* The expected and actual users of the method. For instance, whether the method is designed solely for use by information system professionals or not. Whether user participation (chapter 28) is expected. How much actual practical experience the method has been exposed to.
7. *Product.* The major product produced by the method. For instance, whether the method stops at design, or whether a working system is expected as output.

In the sections which follow we briefly describe three traditional methods: STRADIS; Information Engineering; SSADM. Our aim is mainly to illustrate the distinction between process-driven, data-driven and integrative methods. Discussion of participative methods (chapter 28) and object-oriented approaches (chapter 30) and iterative approaches (chapter 29) is deferred to other chapters. We conclude with a discussion of where each method lies on the seven feature dimensions listed above.

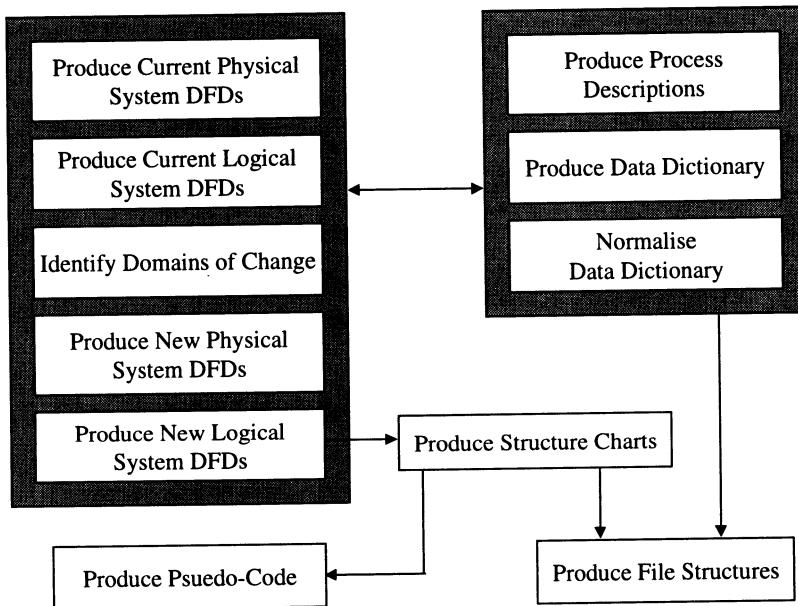
## 27.6 Structured Systems Analysis (STRADIS)

In this section we discuss the basis of a method primarily due to Gane and Sarson (1977) but also expounded in the work of Yourdon, Constantine, DeMarco and Myers (Yourdon and Constantine, 1979) (DeMarco, 1979) (Myers, 1979). The method STRADIS has been developed over a number of years on a practical basis in the United States. Very little has been written on the detailed steps of the method, although commercial variants such SADT (Structured Analysis and Design Technique) are commonplace. Structured Systems Analysis contains most of the classic elements of a process-driven approach.

### 27.6.1 The Structure of Structured Systems Analysis

Gane and Sarson distinguish between four major stages of information systems development: initial study, detailed study, defining and designing alternative solutions, and physical design. The initial study stage is a cut-down version of the feasibility study stage described in chapter 7. Detailed study is similar in objectives to analysis, stage 3 is similar to design and stage 4 is similar to implementation.

Figure 27.1 illustrates the basic elements of stages 2 to 4 of Structured Systems Analysis. We have also incorporated the contributions of Yourdon, Constantine, DeMarco and Myers into the discussion.



**Figure 27.1: Structured Systems Analysis**

The main technique used is data flow diagramming (chapter 17). We first document as much as we can about the existing system as a set of current physical data flow diagrams. We then abstract out the implementation detail from such diagrams to produce a set of current logical data flow diagrams. The next step is to identify significant areas on the DFDs that may be amenable to change with the introduction of computing technology. These domains of change

direct us to produce a set of new logical DFDs. Incorporating implementation detail we produce a set of new physical DFDs.

Alongside the data flow diagramming, developers will be producing and amending a data dictionary (chapter 19) and an associated set of process descriptions (chapter 18) in structured English. The data dictionary will be suitably normalised to design file structures for the new information system.

The eventual set of DFDs will be used to drive the production of structure charts (chapter 21) through transform or transactional analysis. Structure charts constitute the program designs for the new system and, along with process descriptions, are expected to further drive the production of pseudocode.

## **27.7 Information Engineering**

The concept of Information Engineering is generally attributed to the work of James Martin and Clive Finkelstein (Finkelstein, 1989). Information Engineering may be seen as an evolutionary development from an earlier method known as D2S2 (Macdonald and Palmer, 1982). In its present form information engineering is clearly a sophisticated method which displays many of the elements of integrative approaches. The general philosophy of the method however still displays its origins in a data-driven tendency.

Information engineering has been developed specifically with automation in mind. It is a method constructed from a set of integrated, graphical techniques which contribute to defining a model of a given information system. This model is held in an active encyclopaedia which is used to:

1. Store all the information relevant to a given development project.
2. Control consistency and mapping between the various techniques.
3. Produce, via a system generator, major parts of the working information system automatically.

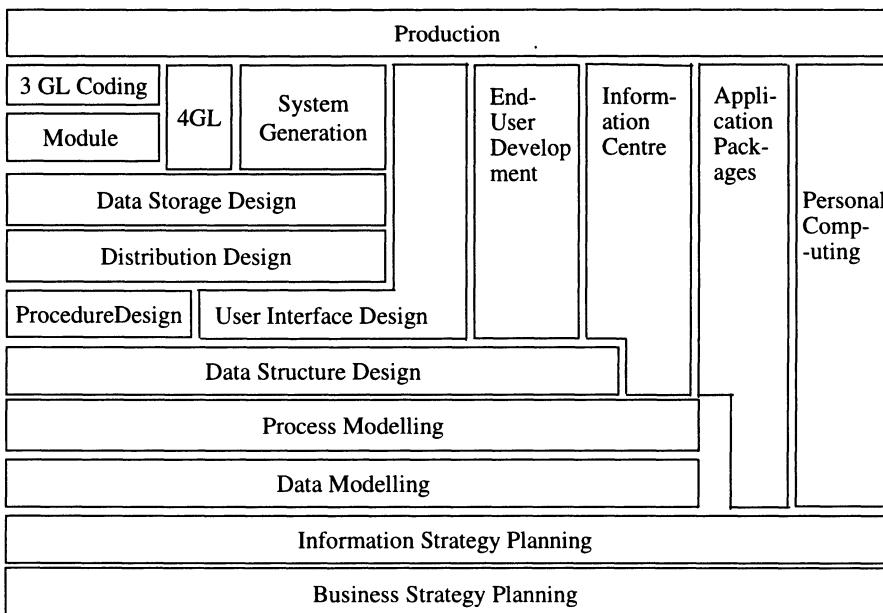
### ***27.7.1 The Structure of Information Engineering***

Figure 27.2 illustrates the concept of information engineering as a set of building blocks, each representing a technique employed.

The horizontal building blocks represent the fundamental analysis, design and implementation techniques of the method. The vertical blocks represent strategies for shortening the project life-cycle:

1. The blocks all rest on an enterprise or business model established by a process known as business strategy planning. This model establishes the mission, objectives and goals of the enterprise.
2. The next block represents information strategy planning. This constitutes a top-down analysis of business objectives, their information needs and

priorities, the types of data that must be kept, the functions making up the business, and how all these pieces relate.



**Figure 27.2: Information Engineering**

3. The third block consists of data modelling and process modelling. Data modelling creates a broad, overview E-R model (chapter 16). Process modelling breaks down broad business functions into specific business processes, and identifies interdependencies between processes and what data is needed to support processes (chapter 18).
4. The fourth block is data structure design that creates the preliminary database design and attempts to make it as stable as possible.
5. A technique known as action diagramming is used to specify the procedures that act on the database. These diagrams are drawn in such a way that they can be converted directly into templates for 4GL programs (chapter 11).
6. For large, heavy-duty applications the application might have to be designed with distribution in mind. Physical tuning of the database may also need to be considered.
7. Module design and 3GL coding (chapter 8) can be used particularly where performance and efficiency is critical.

Essential to the concept of information engineering is a set of alternative fast-development paths that exploit fourth generation technology. The diagram in

figure 27.2 shows the earliest points at which it is reasonable to break out of the standard development cycle and proceed directly to working systems.

## 27.8 SSADM

In 1980 a British information systems consultancy, Learmonth and Burchett Management Systems (LBMS) were invited to work with the Central Computer and Telecommunications Agency (CCTA) of the UK government in carrying out a joint project to develop a standard analysis and design method for public sector organisations (Cutts, 1990). This resulted in SSADM (Structured Systems Analysis and Design Method) which has now undergone a number of versions – Version 4.2 being the latest. SSADM is an open standard, i.e. it is freely available for use in public and private sector industry. The method is now mandatory procedure for most public sector computing projects. Many commercial organisations are now also established users of the method.

SSADM is a classic example of an integrative method. It attempts to encompass the use of process analysis techniques with data analysis techniques in one unified framework (Ashworth and Goodland, 1990).

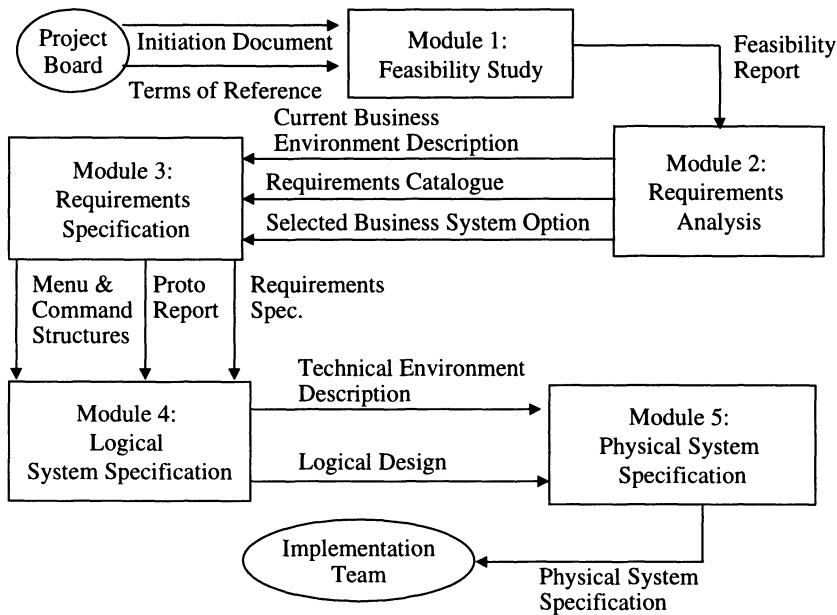
### 27.8.1 *The Structure of SSADM*

SSADM (version 4) is organised into five clearly defined modules (Eva, 1992). These modules are further subdivided into stages, steps and tasks:

1. *Feasibility study.* The feasibility module consists of a single stage – stage 0, feasibility. This stage constitutes a high-level analysis and design exercise. It is undertaken in order to determine whether or not a system can in fact meet business requirements.
2. *Requirements analysis.* This module consists of two stages. Stage 1, investigation of current environment is used to identify the current business environment and system in terms of major processes and data structures. Stage 2, business system options, is used to generate up to six possible logical models of systems.
3. *Requirements specification.* This module consists of a single stage – stage 3, definition of requirements. During this stage more detailed documentation of requirements takes place.
4. *Logical system specification.* This module consists of two stages. stage 4, technical system options, produces specifications pertaining to the technical and development environments for up to six options. One of the options is selected. Stage 5, logical design, represents the logical design of update processes, enquiry processes and dialogues.
5. *Physical design.* This module consists of a single stage: stage 6, physical design which takes the logical specifications produced from module 4 and

creates a physical database design and a set of program specifications.

Figure 27.3 summarises the interaction of these stages as an overview data flow diagram.



**Figure 27.3: Overview Diagram - SSADM (Version 4)**

Each stage can be broken down into more detail in terms of:

1. The set of tasks to be performed in each stage.
2. The set of techniques to be used in each stage.
3. The inputs to the stage, and the outputs from each stage.

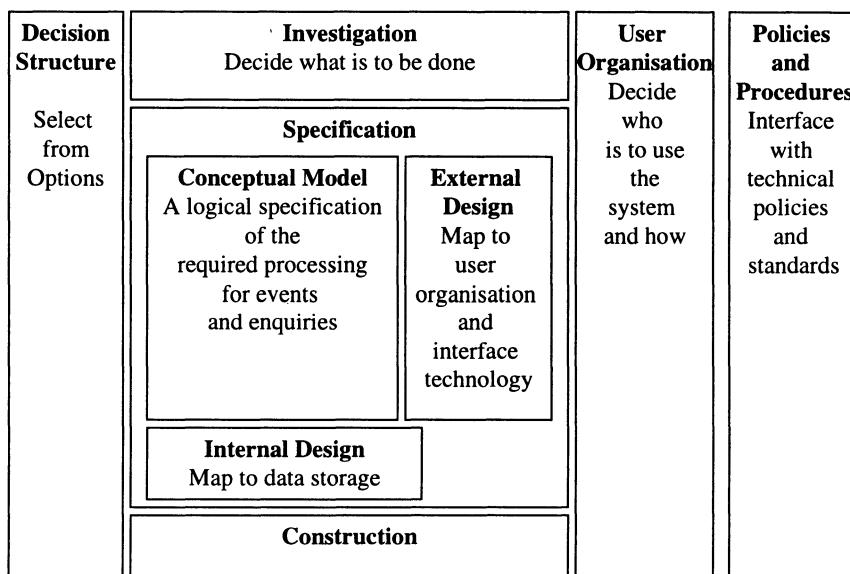
In 1996 a new version SSADM 4+ (or version 4.2) was introduced. This constitutes a repackaging of version 4 of the method rather than a radical departure with the previous version. It particularly renames the collection of stages described above as the default structural model. The idea is that an organisation can diverge from this default model to better support particular project contingencies. SSADM 4+ therefore describes itself as being customisable.

The key additions to SSADM 4+ are the inclusion of a system development template which defines a broad view of the structure of an information systems development project, and a three-schema architecture which separates the design

and specification of an information system into a number of areas, particularly designed to support client-server approaches.

The systems development template consists of the following components as illustrated in figure 27.4. Three components are concerned with interfaces between the project and its environment:

1. The decision structure shows the involvement of management in selecting from options at a number of critical points during the development.
2. The user organisation reflects the mapping of business activities and constraints on to the development process and emphasises the importance of involving the user throughout the development process.
3. The policy and procedures section defines the necessary interfaces between the development effort and the organisation's technical policies and standards.



**Figure 27.4: SSADM System Development Template**

The other three components break down a project into investigation, specification and construction:

1. Investigation constitutes the early part of a development project where a business activity model and a requirements catalogue is created.
2. Specification constitutes the major focus of SSADM 4+. Under this heading, the conceptual model for the information system is defined and mapped on to data storage technology on the one hand and interface technology on the other.

3. Construction is outside the scope of the core method defined above and includes the implementation and testing of the system.

The key techniques from the SSADM toolkit can be mapped on to this system development template.

The three-schema architecture separates the design and specification contained within the method into a number of areas, particularly designed to support client-server approaches:

1. *Conceptual model*. The underlying data and business logic which the system needs to contain
2. *External design*. The user interface which provides the means of accessing the system functions
3. *Internal design*. The physical implementation, management and access to the data

A simple mapping of this 3-schema architecture to a client-server system would mean that external design would concentrate on the client functionality and the conceptual model and internal design would be equated with server functionality.

#### **27.8.2 SSADM in the Development Life-cycle**

There are stages that need to come before and after an SSADM project. The ideal predecessor to the SSADM feasibility study is a business analysis stage (part 4 section 1) as well as the development of a corporate information systems strategy (chapter 33). This will identify and prioritise candidate information systems, each of which will move through the SSADM life-cycle in turn. The progression from physical design to implementation is deliberately left unspecified because of the variety of technical environments used. However, SSADM is increasingly being used in conjunction with RDBMS (chapter 10) and fourth generation tools (chapter 11). The controlling body for the method, the SSADM Management Board, has encouraged suppliers of such products to develop interface guidelines to SSADM, which describe how the step from physical design to implementation should be taken (Haigh, 1991).

The SSADM manuals do not cover all aspects of the work required to implement the five stages of the method. For instance, they do not include a project management method (chapter 31) or procedures for quality control (chapter 36). However, 'hooks' are provided to other methods, such as PRINCE, a standard project management method.

## 27.9 A Feature Analysis

We list some of the general comments we can make on the methods discussed above in terms of the features outlined in section 27.5:

1. *General philosophy.* All the methods considered in this chapter adhere to an objectivist view of reality. In other words, they assume that there is a reality that is the same for everyone. The task of the information systems developer is to abstract key elements of this reality in an information system. Two other methods – ETHICS (chapter 28) and SSM (chapter 26) – take a much more subjectivist view of reality. In other words, they make the assumption that reality is a social construction. That the task of the information systems developer is to negotiate an information system. That an information systems project is a reality constructing process.
2. *Model of development.* All three methods adhere to a waterfall-like model of information systems development. However, Information Engineering encourages the use of prototyping (chapter 24) as a parallel stream of development and SSADM in its current version acknowledges the benefits of prototyping as a requirements elicitation tool.
3. *Techniques.* Information Engineering (IE) and SSADM use a battery of similar techniques, although IE tends to emphasize the importance of data analysis. STRADIS tends to concentrate on the use of DFDs and data dictionaries.
4. *Tool support.* All three methods offer some form of automated support. Many integrated CAISE tools have been designed specifically for SSADM. IE is presently supported by two CAISE workbenches: Information Engineering Facility and Information Engineering Workbench.
5. *Scale of projects.* All three methods were originally designed for large-scale computer projects. However, IE specifically identifies personal computing as a fast strategy for producing systems. SSADM does not currently include a business analysis stage, although a method based on SSM can be used to front-end an SSADM project. IE specifically includes a strategic and business planning stage within its framework.
6. *Intended use.* All three methods are designed for use mainly by information systems developers. No precise figure is available for the actual users of SSADM, IE or STRADIS.
7. *Product.* IE specifically addresses working systems. SSADM and STRADIS stop at design.

## 27.10 Conclusion

Methods are frameworks for the application of techniques and the documentation methods associated with such techniques. Standard methods continue to

grow largely out of practitioner experience in the IS development domain. In this chapter we have largely described some of the features of some dominant structured methodologies.

There has been a growing emphasis on contingent methods in this area. That is, methods which include techniques for determining the best way to tailor a method to particular circumstances. Multiview, described in chapter 26, is perhaps the best known of such approaches. Current interest in the issue of rapid applications development can also be seen as a consequence of such an emphasis. It must be said that in recent years there appears to have been something of a decline of interest in both the development and the use of methods. Although alternative development methods to the structured methods have been proposed, particularly in relation to object-oriented approaches (chapter 30) and to more iterative forms of development, such as rapid applications development (chapter 29), the belief in the ability of methods to solve problems with the development process seems to have waned somewhat.

### 27.11 References

- Ashworth C. and Goodland M. (1990). *SSADM: a practical approach*. McGraw-Hill, London.
- Cutts G. (1990). *Structured Systems Analysis and Design Methodology*. 2nd Ed. Blackwells Scientific, Oxford.
- DeMarco T. (1979). *Structured Analysis and System Specification*. Prentice Hall, Englewood Cliffs, NJ.
- Eva M. (1992). *SSADM (Version 4) – A User's Guide*. McGraw-Hill, London.
- Finkelstein C. (1989). *An Introduction to Information Engineering: from strategic planning to information systems*. Addison-Wesley, Sydney.
- Gane C. and Sarson T. (1977). *Structured Systems Analysis: tools and techniques*. Prentice-Hall, Englewood-Cliffs, NJ.
- Fitzgerald G., Stokes N., Wood J.R.G. (1985). 'Feature Analysis of Contemporary Information Systems Methodologies'. *The Computer Journal*. (28). 223-30.
- Haigh J. (1991). 'Feats of Design'. *Government Computing*. Sept. 20-22.
- Macdonald I.G and Palmer I.R. (1982). 'System Development in a shared data environment – the D2S2 methodology'. In Olle T.W., Sol H.G. and Verryn-Stuart A.A. (Eds) *Information Systems Design Methodologies: a comparative review*. North-Holland, Amsterdam.
- Macdonald I.G. (1986). 'Information Engineering: an improved, automatable methodology for the design of data sharing systems'. In Olle T.W., Sol H.G. and Verryn-Stuart A.A. (Eds). *Information Systems Design Methodologies: improving the practice*. North-Holland, Amsterdam.
- Myers G. (1979). *Structured Design*. Van Nostrand, London.

- SSADM Management Board. (1996). *Introducing SSADM 4+*. NCC/Blackwell. Oxford.
- Yourdon E. and Constantine L. (1979). *Structured Design: fundamentals of a discipline of computer program and systems design*. Prentice Hall, Englewood Cliffs, NJ.

### 27.12 Exercises

1. List the three most important problems in information systems engineering that you believe methodologies address.
2. Methodologies constrain. Discuss the consequences of this statement.
3. Discuss the relationship between CAISE and the methodology movement.
4. Discuss whether a methodology such as SSADM is suitable for PC-based projects.
5. Is a large-scale methodology applicable to the Goronwy Galvanising project?

# **28 Participatory Design**

## **28.1 Introduction**

In recent years there has been something of a renaissance of interest in the idea of Participatory Design (PD) of information systems (IS). Participatory Design might be defined as any attempt to include significant *stakeholders*, particularly end-users in the process of developing an information system. Lytinnen and Hirschheim (1987) define stakeholders as being ‘all those claimants inside and outside the organisation who have a vested interest in the problem and its solution’. Friedman (1989) believes that this focus on user relations constitutes a distinct phase in the history of computing equivalent in stature to two previous phases focused on hardware and software constraints.

Clearly the study of user relations in IS development can be approached from a number of directions. In this chapter and the next we wish to highlight some of the differences and similarities between two approaches to the user relations problem that seem to have evolved from quite separate traditions: participatory design and rapid applications development.

## **28.2 Underlying Philosophy**

The basic tenet underlying the PD movement is that any development method is reliant on a bedrock of philosophical assumptions, and that these assumptions predetermine the way in which IS work is undertaken. Floyd *et al.* (1989) portray the difference between PD (particularly the Scandinavian variant) approaches and other approaches to systems development in terms of two pairs of alternatives, the second of each pair portraying the PD position:

1. The system should either reflect the interests of the system owners, or – as fairly as possible – the interests of all those affected. This has been described as the goal of democratisation.
2. The system is primarily designed to compensate for human weaknesses, or to support human strengths. This is related to the goal of humanisation.

Participatory design might be defined as any attempt to include significant stakeholders, particularly end-users in the process of developing an information system. Kuhn and Muller (1993) define the field of participatory design as a ‘rich diversity of theories, practices, analyses, and actions, with the goal of working directly with users (and other stakeholders) in the design of social systems including computer systems that are part of human work’. They characterise PD as first taking root, especially in the Scandinavian workplace

democracy movement (Bjerknes *et al.*, 1988), and also in the UK with socio-technical design (Mumford (1983), Checkland (1987) and Stamper (1989)). More recently PD has begun to have a certain influence in the US, particularly in relation to Computer Supported Cooperative Work (CSCW) (Muller (1992)), and Joint Application Development (JAD) (Carmel *et al.* (1993)).

### **28.3 Three Traditions**

It is useful to portray PD as emanating from three overlapping traditions that we now examine:

#### *28.3.1 The Scandinavian Tradition*

For a number of years, the Scandinavian countries of Norway, Sweden and Denmark have taken a particularly humanistic view of the development of IT systems. Trades Unions have long been seen to be central to the information systems development process. The strength of trades unions backed by national legislation has meant that employees have been able to negotiate satisfactory agreements with their employers over work practices. Unions have been able to fund their own independent research projects and educational programs in new technology. This has given workers increased understanding of the technology and enabled them to become actively involved in decisions relating to the design and implementation of technical information systems (Bjerknes *et al.*, 1988).

The Scandinavian tradition in participatory design is frequently accounted for in the UK and US as a uniform tradition. Bansler (1989) rightly discusses at least two schools: the socio-technical school and the critical school. Although both arose during the 1970s in response to industrial democracy movements in Norway, Sweden and Denmark, each takes a different approach to participation. The socio-technical school draws on the same tradition as Mumford (Olerup, 1989). The critical school draws more on the idea of action research or praxis and is represented in the work of Nygaard (Norway), Ehn (Sweden) and Kyng (Denmark) (Sandberg, 1985). Hirschheim and Klein (1989) refine this categorisation in terms of two paradigms of information systems development: radical structuralism and neohumanism. Radical structuralists are ‘warriors for social progress’; neohumanists are emancipators or social therapists.

#### *28.3.2 The UK Tradition*

In the UK, a number of suggestions have been made as to ways of improving the fit between technical and social systems. This tradition, which we will call for matter of convenience, the socio-technical tradition, emphasises a more evolutionary approach to systems work based on the establishment of a

consensus over work-technology relations. This is clearly in contrast to the more radical Scandinavian tradition evident in the work of Bjerknes *et al.* (1988). The socio-technical tradition is particularly evident in the work of Mumford (1983), Checkland (1987) and Stamper (1989).

Eason (1988) proposes that a socio-technical design approach is founded on a number of propositions:

... IT technical design is not enough because benefit can only come if these systems are effectively harnessed and exploited by their users. The achievement of this involves the creation of compatible social and technical systems to serve some important organisational purpose. This in turn means the design of a social system to serve this purpose and the creation of a technical system which will support the users in the social system. The design process by which this is achieved requires a process of planned change which not only creates the appropriate system but creates in the users a motivational and knowledge state where they are able and willing to exploit the technical capabilities. This involves the participation of the stakeholders in the design process and individual and collective learning processes. Since organisational change and human learning take time an important requirement is evolutionary development so that decisions are made on the basis of mature reflection. Since there will be existing procedures for system design the pursuit of these objectives is most likely to be successful if techniques are offered in the form of a flexible "toolbox" which can be matched to existing practice and local circumstances.'

### 28.3.3 The US Tradition

A number of workers in the US have been trying to adapt the tenets of the PD approach to the US context. Muller (1992), for instance, discusses how researchers at Bellcore have accounted for the Scandinavian design principles in terms of a North American corporate context:

'We believe that participatory design has specific advantages in terms of relatively delimited and product-oriented business motivations. Participatory design can help to leverage all the available expertise to improve product quality and market share. Participatory design can also help to increase the commitment to the product or project, through the involvement of the customers and the "downstream" software development staff.'

Carmel *et al.* (1993) see Joint Application Design (JAD) as the dominant strand of PD in the US. JAD is sometimes also taken to stand for Joint Application Definition or Joint Application Development. JAD was developed within IBM by Chuck Morris and Tony Crawford during the late 1970s. JAD is

a team-based information gathering and analysis technique in which business users and developers come together in one or more facilitated workshops to focus on a particular problem.

#### **28.4 Components of Participatory Design**

Because of the large diversity of material it is difficult to devise a set of common components for the area of PD. Therefore, the list below is particularly directed at the micro level; at the level of development techniques, where we believe there is most potential for practical application in the UK context:

1. *User participation.* Hirschheim (1983) maintains that PD is not the same as user involvement. All systems have some degree of user involvement, but only some are participatively developed. PD differs from user involvement in that PD deals with content as well as involvement issues. PD broadens the scope of what is being designed to include social and job considerations. Also, PD tends to include more user groups in design activities.
2. *Empowerment.* A related issue to participation is the degree to which information systems, both in development and use, ‘empower’ the workforce. Clement (1994) distinguishes between functional empowerment and democratic empowerment. The ‘concept of empowerment may be termed functional in the sense that it is oriented towards improving performance in the interest of organisational goals that are assumed to be shared unproblematically by all participants’. Functional empowerment contrasts with an older view of empowerment which Clement calls democratic empowerment. ‘Democratic empowerment ... emphasises the rights and abilities of people to participate as equals in decisions about the affairs that affect them’. Democratic empowerment, characterised by Muller (1992) in the *adage no mechanisation without representation*, is the type of empowerment frequently highlighted in the PD literature.
3. *Work analysis.* PD workers have particularly emphasised a close analysis of the context of the current work situation, as well as an awareness of the way in which the introduction of an IS inevitably changes work. This is seen as particularly important in light of the way in which tacit knowledge underlies work. Tacit knowledge is the name given by Michael Polanyi (1958) to the ability of human beings to perform skills without being able to articulate how they do them.
4. *Job design.* One of the key principles of PD is that developers must concern themselves not only with the development of technical systems but also with the design of work. For example, the aim of Mumford’s methodology ETHICS is to design a new form of work organisation with the dual objectives of improving job satisfaction (the social system) and work efficiency (technical system).

5. *Cooperative design and prototyping.* Clement and Van den Besselaer (1993) note that there seems to be a general shift in orientation of PD projects over a decade or so. Early projects focused on worker democratisation. More recent work has focused on developing cooperative methods for development. Cooperative design has been implemented by the Aarhus school of PD particularly in relation to cooperative prototyping. Bødker and Grønbæk (1991) define cooperative prototyping as 'a design process where both users and designers are participating actively and creatively, drawing on their different qualifications.'
6. *Mutual learning.* Users and developers alike are reliant on a mutual process of learning and communication. Users and developers are seen as being equal partners in the development process. Users gradually learn about the technical possibilities and developers gradually learn about the work of organisations.
7. *Designing by doing.* Early experimentation and testing is encouraged, particularly using cooperative and low-tech prototyping and also by 'playing' envisioning games. All these techniques aim to promote communication and learning processes. One particularly important design technique used in association with cooperative prototyping is the search for design breakdowns. Bødker (1991) characterises breakdowns as situations in which unforeseen changes in the material conditions of some human activity causes a reflection on everyday actions which would not normally occur. Breakdowns are observed in and reflected upon in the work-like evaluation of prototypes.
8. *Low-technology prototyping.* One emphasis in the PD area is on the use of non-computer prototypes such as paper and pencil or cardboard mockups, sometimes even board games, as a means of helping both developers and users envision future uses of technology. This is based on the idea that frequently computer-based (high technology) prototypes get in the way of effective design in that they are predicated on a level of understanding which the user community may not currently have.

## 28.5 ETHICS

In Britain, probably the most influential of the attempts to introduce user participation into information systems development is the work of Enid Mumford (1983) at the Manchester Business School.

Mumford distinguishes between three different levels of user participation:

1. *Consultative participation.* Decision-making is still in the hands of systems analysts, but there is a great deal of staff at every level.
2. *Representative participation.* A design group is formed made up of representatives of all grades of staff with systems analysts. The representatives however are selected by management.

3. *Consensus participation.* A design group is formed as in 2, but representatives are elected by staff and given the responsibility to communicate group decisions back to staff.

Mumford has developed a method for participative development out of her work on socio-technical systems. The method is called ETHICS (Effective Technical and Human Implementation of Computer Systems) (Mumford, 1996). The objective of ETHICS is to design a new form of work organisation with the dual objectives of improving job satisfaction (the social system) and work efficiency (technical system). The steps of the method are laid out below and illustrated in figure 28.1.

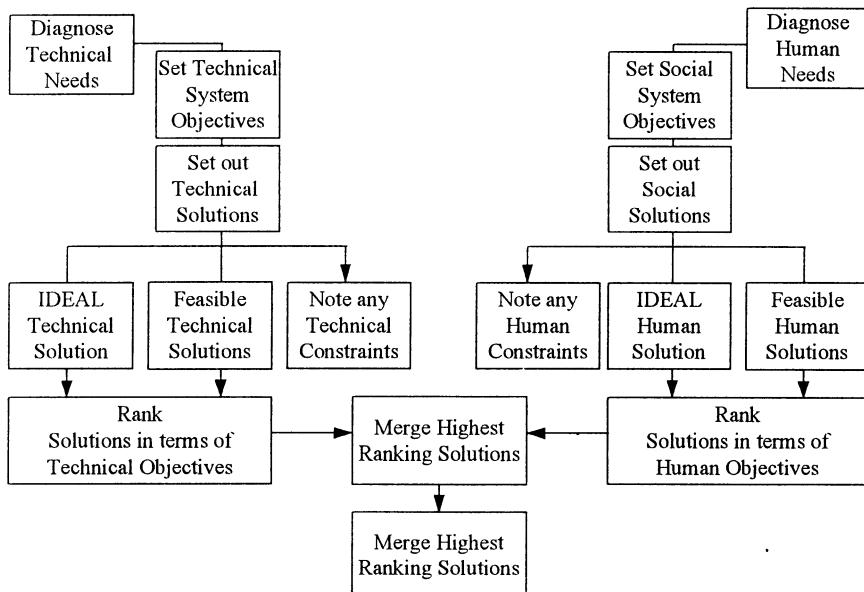


Figure 28.1: The ETHICS Method

1. The diagnosis of technical and human needs is undertaken primarily by collecting and analysing questionnaire data. This data is used to determine the fit between the present work situation and a valued work situation in terms of a typology of needs. Both job satisfaction objectives (such as provide work variety, and performance incentives) and efficiency objectives (such as improve level of customer performance) are then constructed.
2. In step 2 we set out a range of technical and social alternatives. We then assess each alternative in terms of human advantages and disadvantages and technical advantages and disadvantages.
3. In step 3 we rank each technical and human alternative in terms of their

ability to meet both human and technical objectives. We choose the highest ranking human and technical alternatives that are compatible.

#### 4. Develop detailed socio-technical design.

Mumford has recently developed a cut-down version of ETHICS designed for use in business analysis exercises called QUICKETHICS.

##### *28.5.1 An Example of the Application of ETHICS*

Mumford provides a case study of a customer orders and accounts system which was analysed by the ETHICS approach, and from which a new social and technical system emerged. Although this was conducted in the 1980s, it illustrates the essential principles of the approach.

The basic technical system involved orders clerks filling out order forms, accounts clerks updating customer ledgers and any problems being resolved by lots of paper-passing. The social system represented a clear distinction between orders clerks and accounts clerks, each accounts clerk working across a number of different customer accounts, and problems being resolved by two senior clerks.

The motivations for change were both technical and social. Examples of technical problems were orders incorrectly filled out, customer ledgers incorrectly updated, slow methods of resolving problems. Examples of social problems included high absenteeism, high staff turnover, and some 'industrial vandalism'.

Results from the job satisfaction questionnaire identified low job satisfaction in the present work situation as a result of piece-work, lack of overall picture, individual isolation, low status, and poor prospects for advancement. The valued work situation was characterised in terms of more responsibility, group working, more important work and better opportunities for advancement.

After the study was complete a new work and technical system was put into place. The technical system constituted terminal input of orders, batch update of a central database and regular reports output from the system. The social system was changed to small work groups of 5 clerks, each work group handling orders and accounts for a group of customers. The group handled customer problems thrown up by printouts.

## 28.6 Conclusion

Participatory design is defined as any attempt to include significant stakeholders, particularly end-users in the process of developing an information system. However, participatory design does not constitute a unified tradition. Instead, it is composed of a number of overlapping areas generated by different schools of thought geographically dispersed around the world. Having said this,

all of the schools of participatory design take a generally emancipatory stance to the introduction of IS/IT into organisations. Greater democratisation, increasing job satisfaction and generally improving cooperation and coordination within the workplace through the introduction of IS/IT are all key themes of this movement.

## 28.6 References

- Bjerknes G., Ehn P. and Kyng M. (1988). (Eds) *Computers and Democracy: A Scandinavian Challenge*. Gower, Oxford.
- Bødker S. (1991). *Through the interface: a human activity approach to user interface design*. Lawrence Erlbaum, New Jersey.
- Bødker S. and Grønbæk K. (1991). 'Design in Action: from prototyping by demonstration to cooperative prototyping'. In *Design at Work: cooperative design of computer systems*. Greenbaum J. and Kyng M. (Eds). Lawrence Erlbaum, NJ.
- Checkland P. (1987). *Systems Thinking, Systems Practice*. John Wiley, Chichester.
- Clement A. and Van den Besselaer P. (1993). 'A Retrospective Look at Participatory Development Projects'. *CACM*. 36(4). June.
- Floyd C., Mehl W-M, Reisin F-M, Schmidt G., Wolf G. (1989). 'Out of Scandinavia: alternative approaches to software design and system development'. *Human-Computer Interaction*. 4. 253-350.
- Friedman A.L. with Cornford D.S. (1989). *Computer Systems Development: history, organisation and Implementation*. John Wiley, Chichester.
- Hirschheim R. A. (1983). 'Assessing Participatory Systems Design: some conclusions from an exploratory study'. *Information and Management*. 6. 317-328.
- Kuhn S. and Muller M.J. (1993). 'Participatory Design'. *CACM*. 36(4). 26-28.
- Muller M.J. (1992). 'Retrospective on a year of participatory design using the PICTIVE technique'. *CHI'92*. May 3-7.
- Mumford E. (1983). *Designing Participatively*. Manchester Business School Press.
- Mumford E. (1995). *Systems Design – Ethical Tools for Ethical Change*. Macmillan Press, Basingstoke.
- Mumford E. (1996). *Effective Systems Design and Requirements Analysis – The ETHICS Approach*. Macmillan Press, Basingstoke.
- Polanyi M. (1958). *Personal Knowledge*. Macmillan, London.

### 28.8 Exercises

1. Discuss some of the reasons why PD has taken off in Scandinavia, but has yet to make an impact in the UK or the US?
2. Describe some of the advantages you feel might accrue from involving users more directly in development projects as Mumford suggests.
3. To what extent do you think that socio-technical approaches to design are similar to or different from BPR?

# **29 Rapid Applications Development**

## **29.1 Introduction**

Rapid Applications Development (RAD) seems to be a term that is currently used in association with a whole range of tools, techniques, methods, and indeed even information systems management styles. In this chapter we describe some of the key features of RAD approaches.

## **29.2 The Context for RAD**

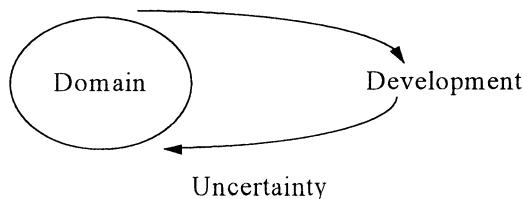
Giddings (1984) characterises information systems as domain-dependent software. Information systems are domain-dependent because there is a necessary interdependence between the software system and its domain or universe of discourse. Such systems are characterised by an intrinsic uncertainty about the universe of discourse. In particular, the use of the software may change the nature of the universe of discourse and hence the nature of the problem being solved. For instance, the development and introduction of an IS to support the domain of corporate sales may cause concomitant changes to the way in which sales work is organised. Hence, what constitutes IS ‘support’ for sales staff may also change, leading to a redefinition of the problem that an IS is designed to solve (see figure 29.1).

RAD can be seen as a response to two types of uncertainty evident in this proposition (Beynon-Davies *et al.*, 1996):

1. *Business uncertainty*. A necessary uncertainty characteristic of the domain itself (the business environment). It has become almost accepted wisdom that the modern business environment is characterised by large rates of change. Companies must react more quickly to changes in the business environment to survive (Scott Morton, 1991). Features of the current business environment include globalisation of markets and production, virtual organisations, and customisable products (Drucker, 1988). In response to business uncertainty there is some evidence of dissatisfaction on the part of business with the service provided by IS developers. As evidence of this, a number of articles have been published on the so-called productivity paradox of information technology (Brynjolfson, 1993). There is also some evidence of an increasing concern amongst organisations in the UK and elsewhere with the large amounts of money that appears to have been devoted to software projects with little apparent organisational benefit. Available literature suggests that failure is a ubiquitous feature of both the UK and International experience of IS engineering (Coopers and Lybrand, 1996). It is therefore perhaps not

surprising to find that many companies are now actively pursuing outsourcing strategies in the face of dissatisfaction with internal IS performance (Lacity and Hirschheim, 1993).

2. *Development uncertainty.* Uncertainty that is introduced by the process of developing information systems. The modern IS development environment is also subject to large rates of change. At the level of technology, for instance, developers now face a bewildering array of tools including DBMS, 4GLs, GUI builders, CASE tools, middleware etc. As a result, the IS development task no longer involves programming within the domain of a single language such as COBOL or C. Instead IS development involves a hybrid of code production, re-use of established software components, and distribution of software modules across local and wide area networks. Development uncertainty is also characterised by a lack of a current dominant paradigm in the IS methods area. There appears to be something of a mismatch between established methods such as SSADM and the current development domain. Fitzgerald (1996), for instance has found a low take up of large-scale methods in the current development domain in Eire. This is confirmed by research conducted by Moynihan and Taylor (1995) in the UK. A frequent reason cited in both studies is the perceived cumbersome nature of methods such as SSADM in relation to current development work.



<u>Business Uncertainty</u>	<u>Development Uncertainty</u>
globalisation	hybridisation of development
customisation	methodological ‘chaos’
virtual organisations	

Figure 29.1: Business and Development Uncertainty

### 29.3 Objectives and Assumptions

RAD first appears to have become topical with the publication of a text by James Martin with the same title. Martin defines the key objectives of RAD as high quality systems, fast development and delivery and low costs. These objectives can be summed up in one sentence: *the commercial need to deliver working business applications in shorter timescales and for less investment*.

A number of assumptions are seen as underlying the RAD approach:

1. That it is impossible to specify requirements accurately.
2. That the application is its own model.
3. That design and testing are iterative processes.
4. That the application is always complete, but never finished.
5. That 'empowerment' of developers and users to make decisions on behalf of the business is crucial to the effective development of information systems.

#### **29.4 Methods**

A number of people see RAD as a complete approach to information systems development in that it covers the entire life-cycle, from initiation through to delivery. Not surprisingly there are a number of methods available for RAD – such as Martin and more recently in the UK, DSDM. The Dynamic Systems Development Method (DSDM) consortium have produced a number of versions of a public domain RAD method (DSDM, 1995). This method seems particularly directed at melding standard development issues such as project management, quality assurance and software testing with the exigencies of rapid development. The expressed aim of the consortium is to remove the 'hacker' connotations associated with what many people refer to as first generation RAD.

However, RAD does not appear to be an information systems methodology in the sense of the term used in chapter 27. For instance, although there are a number of methods available for RAD, such as Martin and DSDM, most of these initiatives emphasise a tool-kit or contingent approach to systems building rather than a prescriptive one. Hence, for instance, the DSDM manual is deliberately a slim, one-volume work as compared, for instance, to the eight or so volumes documenting Structured Systems Analysis and Design Method (SSADM). Also, RAD approaches such as DSDM have not proposed any radically new development techniques. Instead, they have indicated how standard systems analysis and design techniques (such as E-R Diagrams, State Transition Diagrams, etc) can be fitted into a rapid development framework.

#### **29.5 Components**

While acknowledging the contingent nature of RAD, the following appear to be the common components of RAD approaches discussed in the literature:

1. *Joint application design (JAD)*. RAD seems to be characterised by small development teams of typically 4-8 persons. Such teams are made up of both developers and users who are empowered to make design decisions. This means that all RAD team members must be skilled both socially and in terms of business. Users must possess detailed knowledge of the application area; developers must be skilled in the use of advanced tools. Hence, 'team-

building' activities such as team dinners are seen as an important part of a RAD project. Most approaches to RAD seem to use joint application development (JAD) workshops at various points in the development process, particularly to elicit requirements. In such workshops, key users, the client, some developers and a 'scribe' produce system scope and business requirements under the direction of a 'facilitator'. Development teams are usually expected to come up with fully documented business requirements in 3/5 days. Such requirements may specify a series of phased deliverables over a given time-span. Further development workshops may be scheduled during the life of a project to develop jointly each deliverable.

2. *Rapidity of development.* RAD projects seem to be typically of relatively small-scale and of short duration. Also, 2-6 months is frequently discussed as being a normal project length. The main rationale for this is that any project taking more than six months to complete is likely to be overtaken by business developments. In total, it has been suggested that no more than 6 man-years of development effort should be devoted to any particular RAD project.
3. *Clean rooms.* JAD workshops are usually expected to take place away from the business and developer environments in 'clean' rooms – that is, places free from everyday work interruptions and full of requisite support facilities such as flip charts, post-its, coffee, computers, etc. The emphasis is on highly focused problem solving.
4. *Timeboxing.* Project control in RAD is seen to involve scoping the project by prioritising development and defining delivery deadlines or 'timeboxes'. If projects start to slip, the emphasis in RAD projects is on reducing the requirements to fit the timebox, not in increasing the deadline. See figure 29.2.
5. *Incremental prototyping.* RAD is frequently discussed in terms of incremental prototyping and phased deliverables. Prototyping is essentially the process of building a system in an iterative way. The developers, after some initial investigation, construct a working model that they demonstrate to a representative user group. The developers and the users then discuss the prototype, agreeing on enhancements and amendments. This cycle of inspection-discussion-amendment is usually repeated at least three times in RAD projects, until the user is satisfied with the system. In RAD, prototyping may be used at any stage of development: requirements gathering; application design; application build; testing; delivery.
6. *Rapid development tools.* It is not surprising to find that modern approaches to RAD demand good support from tools for rapid developmental change. This normally means some combination of fourth generation languages (4GLs), graphical user interface (GUI) builders, database management systems (DBMS) and computer-aided software engineering (CASE) tools. Using such tools some changes to prototypes can be made *in situ* at user-developer meetings.
7. *Highly interactive, low complexity projects.* Most RAD projects seem to be

conducted on applications that are highly interactive, have a clearly defined user group and are not computationally complex. The tendency is to rule out the applicability of RAD for large-scale, infrastructure projects, particularly the construction of large, distributed information systems such as corporate-wide databases. Evidence suggests that such infrastructure is best put in place before undertaking RAD projects. Such an infrastructure can then act as a feeder to systems developed using RAD. This is perhaps not surprising when one considers that most RAD tools work off a database in some way. Therefore, the database needs to be created before application development begins.

8. *Types of RAD project.* There generally appear to be two types of RAD project: the intensive and the phased RAD project. In the highly intensive type of project, a team of developers and users are closeted away in a clean room for some weeks, and are expected to produce a working deliverable at the end of that time. A phased project is one spread over a number of months. Such projects are normally initiated by a JAD or JRP (Joint Requirements Planning) workshop. The subsequent phases of the project are then normally organised in terms of the delivery and demonstration of three incremental prototypes. The aim is to continually refine the prototype into something that is deliverable at the end of timebox.

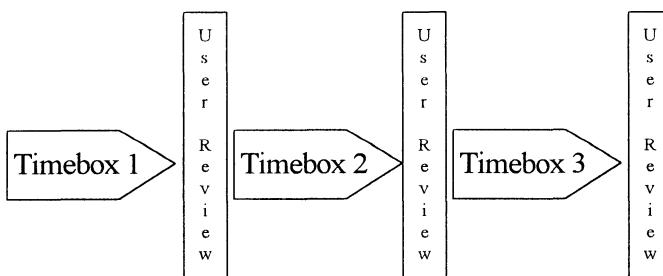


Figure 29.2: Timeboxes and User Reviews

## 29.6 Dynamic Systems Development Method

Dynamic Systems Development Method (DSDM) is a non-proprietary RAD method. It is produced by the DSDM consortium: a non-profit-making organisation of vendors, users and individual associates of RAD. In early December, 1995, the consortium had almost 100 member companies (Stapleton, 1997). Its intention is to become the UK and international standard for RAD work. Many CASE and application building tool vendors are committed to it. Many companies have now adopted it as their preferred framework for RAD projects.

### 29.6.1 DSDM Principles

The Dynamic Systems Development Method Consortium maintains that DSDM is based on nine fundamental principles:

1. *Active user involvement is imperative.* DSDM sees itself as a user-centred approach. Active involvement by the user community throughout the development project is therefore seen as crucial.
2. *DSDM teams must be empowered to make decisions.* DSDM project teams consist of both developers and users. Both groups must be given the power to make key decisions. The developers need to be able to rapidly decide on technical solutions. The business users need to be able to decide upon key requirements for the application.
3. *The focus is on frequent delivery of products.* The work of a DSDM project is focused on application products that can be delivered within agreed periods of time. This enables the project team to define quickly the optimal approach to achieving the products required in the time available.
4. *Fitness for business purpose is the essential criterion for acceptance of deliverables.* The focus of a DSDM project is on delivering business functionality in the required time. This means that a system may be rigorously engineered later if this is felt fit. Traditionally, the focus has been on rigorously engineering systems to satisfy a requirements document, whilst ignoring the fact that documented requirements may be inaccurate.
5. *Iterative and incremental development is necessary to converge on an accurate business solution.* The key emphasis in DSDM is on evolving a system by incremental steps. Partial solutions may be delivered to fulfil immediate business need. Later versions of a system are built on the basis of lessons learned in the feedback process from users. Only by explicitly reworking a system in this way can an accurate business solution be produced.
6. *All changes during development are reversible.* The ability to backtrack to a previous version of a system is seen to be an inherent and important feature of a DSDM development.
7. *Requirements are baselined at a high level.* Requirements are ‘frozen’ at a high-level by agreeing the purpose and scope of the system without presupposing that a more detailed investigation of requirements is needed.
8. *Testing is integrated throughout the life-cycle.* Testing is not treated as a separate phase or activity within DSDM. As the system is developed incrementally, so is it tested incrementally. Testing is conducted both to ensure that it is fulfilling business need as well as being technically sound.
9. *A collaborative and co-operative approach between all stakeholders is essential.* The nature of DSDM projects in which low-level requirements are not fixed at the outset demands continuous Cupertino and collaboration between sponsors, developers and users throughout the life of a project.

### 29.6.2 The DSDM Life-cycle

There are five phases of development within a DSDM project (see figure 29.3):

1. *Feasibility study*. This phase considers the feasibility of the project in business and technical terms as well as the suitability of the project for a RAD approach.
2. *Business study*. This phase defines the high level functionality and the major business entities affected.
3. *Functional model iteration*. This phase is used to construct and demonstrate the required functionality using a working prototype.
4. *System design and build iteration*. This phase is used to refine the functional prototype, particularly to meet non-functional requirements.
5. *Implementation*. The implementation phase includes the handover to users followed by a review of the project's success.

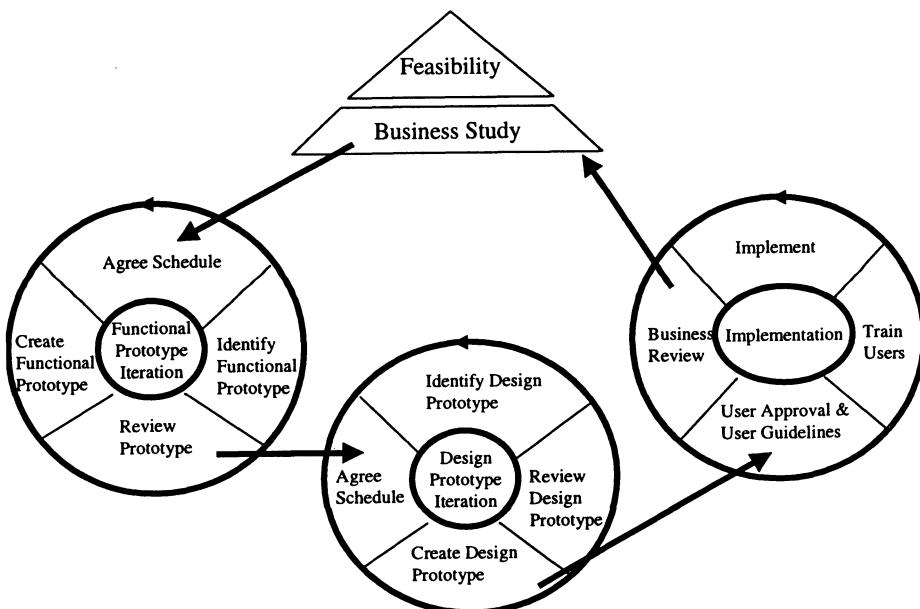


Figure 29.3: The DSDM Life-Cycle

### 29.7 Conclusion

RAD and PD (chapter 28) both have user involvement in the development of information systems as their primary focus. Rapid applications development (RAD) is a contingent method that has developed in response to business and development uncertainty in commercial information systems development.

Participatory design (PD) is a style of development that has developed primarily in the academic arena as a response to dissatisfaction with the underlying philosophy of dominant development approaches. At the macro level, the two development styles appear divergent in the sense that they adhere to radically different philosophies. PD emphasises worker emancipation while RAD emphasises performance and efficiency concerns. This may account for the current degree to which the RAD community is unaware of the PD area and vice versa. However, at the micro level, there are many potential points of synergy. For instance, there is much potential for exploiting some of the techniques or approaches developed within PD (such as low-technology prototyping, designing by doing etc.) within the context of RAD.

## 29.8 References

- Beynon-Davies P., Mackay H., Slack R., Tudhope D. (1996). 'Rapid Application Development: the future for business systems development?' *Proc. BIT'96*. Manchester.
- Brynjolfsson E. (1993). 'The Productivity Paradox of Information Technology'. *CACM*, 36(12), 67-77.
- Coopers and Lybrand. (1996). *Managing Information and Systems Risks: results of an international survey of large organisations*.
- Drucker P.F. (1988). 'The Coming of the New Organisation'. *Harvard Business Review*, 66(1), 45.
- Fitzgerald B. (1996). 'An Investigation into the Use of Systems Development Methods in Practice'. *Proc. ECIS'96*, Lisbon.
- Giddings R.V. (1984). 'Accommodating Uncertainty in Software Design'. *CACM*, 27(5), 428-434.
- Lytinnen K. and Hirschheim R. (1987). 'Information Systems Failures: a survey and classification of the empirical literature'. *Oxford Surveys in Information Technology*, 4, 257-309.
- Lacity M.C. and Hirschheim R. (1993). *Information Systems Outsourcing: myths, metaphors and realities*. John Wiley, Chichester.
- Moynihan E. and Taylor M. (1995). 'The Evolution of Systems Methods and why they are sometimes not used'. *Proc. BCS Information Systems Methods Conf*, Wrexham.
- Scott Morton M.S. (1991). *The Corporation of the 1990s*. Oxford University Press, New York.
- Stapleton J. (1997). *DSDM – Dynamic Systems Development Method: the method in practice*. Addison Wesley Longman. Harlow, England.

**29.9 Exercises**

1. Consider an organisation known to you. To what extent do you believe that the RAD approach is applicable?
2. Generate a list of factors that might be important in selecting a project for RAD.
3. To what extent is RAD a replacement for other methods such as SSADM?
4. Is the formalisation process being undertaken by DSDM a good or bad thing for RAD? Discuss.

# **30 Object-Oriented Methods**

## **30.1 Introduction**

Graham (1994) discusses the various approaches to object-oriented analysis and design in terms of three dimensions: process, data, and dynamics or control. Methods fall into two basic types: ternary and unary. Ternary methods (such as Rumbaugh's OMT and Martin and Odell's Ptech) apply different techniques (frequently adaptations from the structured approaches) to each of these dimensions of information systems modelling. Unary methods (such as Coad and Yourdon and Wirsfs-Brock) attempt to incorporate all three dimensions in one consistent O-O approach. Graham defines his own method, called SOMA, which he describes as a hybrid approach in that it attempts to utilise the strengths of various approaches.

This chapter serves as a review of OO approaches to analysis and design work. For more detail on object modelling readers are referred to chapter 22. Because of the vast array of different terminology current amongst the OO methods, we have tended to utilise the terms discussed in chapter 22 as a standard across the distinctions.

## **30.2 A Unary Method: Coad and Yourdon**

This method is interesting for being the first relatively complete OO methodology available for commercial software projects. Coad and Yourdon published two books in quick succession: one on OO analysis (Coad and Yourdon, 1991), the other on OO design (Coad and Yourdon, 1991a). In these works they emphasise the similarity of notations between analysis and design in the OO tradition as a strength of their approach.

Coad and Yourdon suggest that OO analysis proceeds in five stages associated with the five layers of an object model:

1. *Subjects*. The problem area is decomposed into a number of layers of some five to nine objects known as subjects.
2. *Objects*. The objects are then identified in greater detail by performing data analysis.
3. *Structures*. Here, generalisation and aggregation hierarchies (chapter 22) are developed.
4. *Attributes*. Attributes are detailed against objects and cardinality and optionality conditions specified for relationships of association.
5. *Services*. This is the term used by Coad and Yourdon for operations or methods. In terms of each object the available range of services is specified.

Figure 30.1 illustrates a class diagram, a subject area diagram, a generalisation and aggregation hierarchy and an association relationship in the Coad and Yourdon notation. Note that the class diagram is divided into three areas, corresponding to the class name, the list of attributes and the list of methods. The framed outlines around the classes are meant to indicate classes that can have instances as compared to abstract classes.

On the generalisation and aggregation hierarchies in figure 30.1 AKO: and APO: stand for a-kind-of and a-part-of respectively. The notation used for the generalisation relationship here indicates that there may be other subclasses of a vehicle not indicated. On the aggregation and association relationships, crow's-feet indicate the many end of relationships, while circles indicate optionality.

The methodology does not suggest in any great detail how the behavioural part of an object model should be specified. Instead, Coad and Yourdon suggest that available techniques from the armoury of structured techniques such as data flow diagrams and state transition diagrams be used for this purpose.

Coad and Yourdon's conception of OO design includes four additional components. OO design principally involves refining the products of OO analysis into something they refer to as the Problem Domain Component. To this is then added a human interaction component, a task management component and a data management component. These components essentially address design issues such threads of control, specific processors, package software, etc.

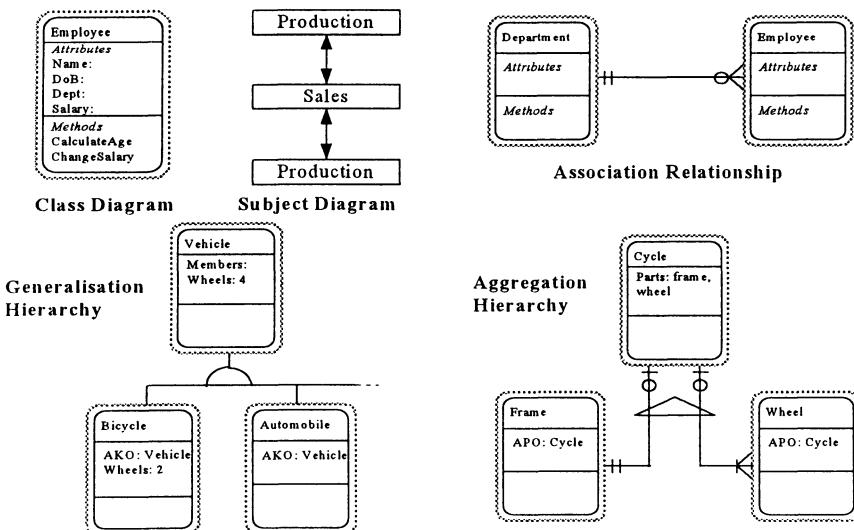


Figure 30.1: Coad and Yourdon Notation

### 30.3 A Ternary Method: Rumbaugh's OMT

The Object Modelling Technique (OMT) is widely regarded as one of the most complete OO methods so far published. Originating from the work of James Rumbaugh (Rumbaugh *et al.*, 1991) and his colleagues at General Electric (Blaha *et al.*, 1988) it constitutes a methodology with strong roots in structured approaches as well as a rich and detailed set of analysis and design notations.

OMT breaks down into three main phases or activities: analysis, system design and object design. The analysis phase assumes that a requirements specification exists and involves producing three separate models with different notations. The first model is the object model, which consists fundamentally of an E-R diagram (chapter 16) annotated with additional information. The second model is the dynamic model built using state transition diagrams (chapter 20). The third model is the functional model, which for all intents and purposes can be considered as essentially a data flow diagram set. Operations discovered in the process of generating the dynamic and functional models are added to the object model.

Figure 30.2 illustrates the basic object-modelling notation of OMT. The filled circles on the aggregation and association relationships represent the many ends of relationships. The open circle represents optionality (zero or one). State transition diagrams in OMT are generally drawn as fence diagrams and are known as event trace diagrams. For data flow diagramming conventions, see chapter 17.

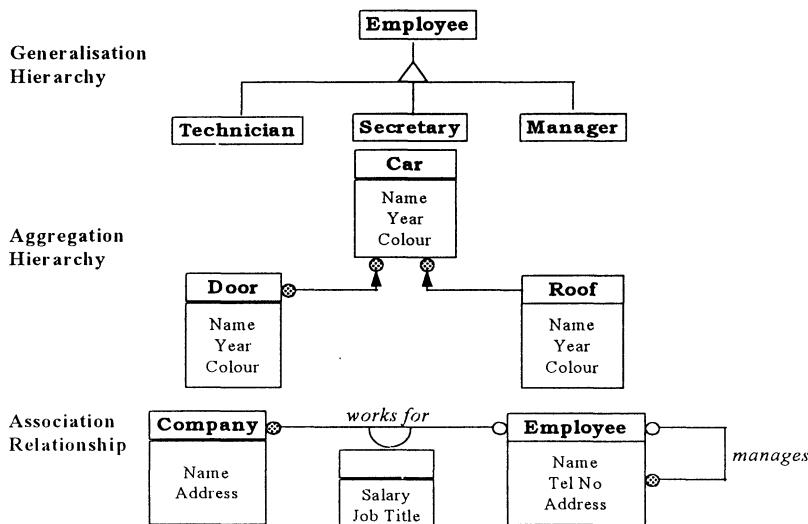


Figure 30.2: OMT Notation

Having completed the two to three models, system and object design can be conducted. System design proceeds, amongst other things, by organising objects into subsystems, identifying concurrency, allocating subsystems to processors or tasks, deciding on data storage strategies, the use of peripherals and global resources, and deciding on control structures. Object design involves converting the information on the dynamic model and functional model into object model operations. All of this information is then documented in a design specification.

### 30.4 A Hybrid Approach: SOMA

Graham (1994) describes a hybrid method which has been influenced not only by methods like Coad and Yourdon and OMT but also by work in artificial intelligence (chapter 14) and semantic data modelling (chapter 16). Figure 30.3 illustrates some of the notation suggested in SOMA; a notation that Graham maintains is less ambiguous than that available in most other OO methods.

Seven activities are described as comprising SOMA:

1. Identifying layers
2. Identifying objects
3. Identifying usage, generalisation and aggregation structures
4. Defining data semantics and associations
5. Adding attributes to objects
6. Adding methods to objects
7. Adding the declarative semantics to objects (rule-sets)

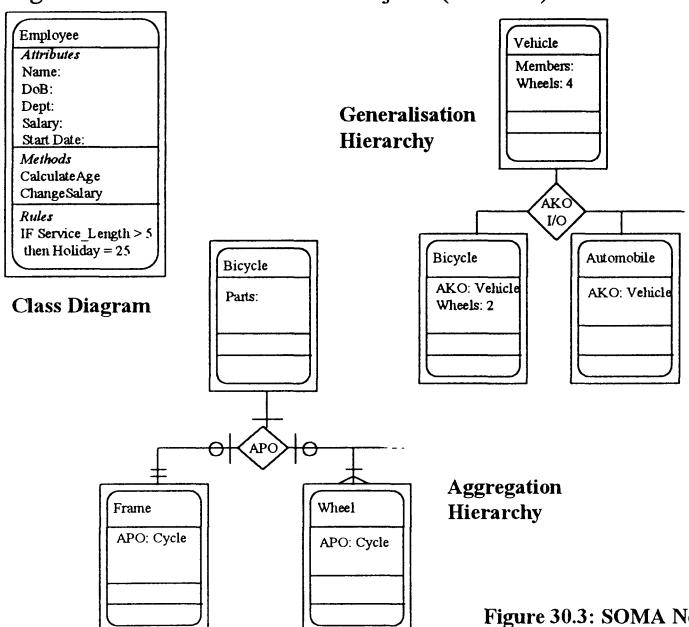


Figure 30.3: SOMA Notation

Figure 30.3 illustrates some of the elements of the SOMA notation. Note that the rounded rectangle used in the COAD and Yourdon notation is replaced by a normal rectangle to distinguish between a SOMA class and a SOMA class with instances. Note also that a section for rules is included along with attributes and methods on the class diagram. Here, the modeller can particularly include declarative integrity rules (in an expert system type of syntax; see chapter 14) to more fully specify an application.

The AKO I/O label in the diamond on the generalisation hierarchy indicates this is an a-kind-of relationship and that it is inclusive/optional. Inclusive means that the subclasses in this hierarchy may overlap. Optional indicates that there may be more, as yet unidentified subclasses for this particular superclass.

The APO label on the aggregation hierarchy clearly labels this as an a-part-of relationship. Note the way in which for each participating class in this aggregation relationship appropriate data semantics can be defined for optionality and cardinality.

### 30.5 Conclusion

In this chapter we have particularly concentrated on three OO methods that have been designed with the information systems development problem in mind – Coad and Yourdon, OMT and Soma. We are aware that there are a number of other OO methods available that have been designed to handle specification in areas such as real-time simulation. The main distinction between the methods described here and the other methods are in their emphasis on the problems of handling complex data as opposed to complex processing. In very general terms most commercial information systems are characterised by the need to accommodate complex data structures, but rely on comparatively simple processing. In contrast, areas such as real-time systems normally demonstrate very complex processing functionality with comparatively simple data structures. Therefore, it is perhaps no surprise to find that most of the methods described here can be seen as natural extensions to conventional data modelling.

One recent development may have a significant impact on the area of OO analysis and design. This is the attempt by three ‘gurus’ of the field, Grady Booch (Booch), James Rumbaugh (OMT) and Ivar Jakobson (OOSE), to develop a Unified Modelling Language for OO analysis and design. Readers are referred to Booch *et al.*, 1998 for detail on this development.

### 30.6 References

- Blaha M., Premerlani W., Rumbaugh J. (1989). ‘Relational Database Design Using an Object-Oriented Methodology’. *CACM*. 31(4). 414-427.

- Booch G., Rumbaugh J. and Jakobson I. (1998). *Unified Modelling Language User Guide*. Addison Wesley Longman, Reading, Mass.
- Coad P. and Yourdon E. (1991). *Object-Oriented Analysis*. 2nd Ed. Prentice Hall, Englewood Cliffs, NJ.
- Coad P. and Yourdon E. (1991a). *Object-Oriented Design*. 2nd Ed. Prentice Hall, Englewood Cliffs, NJ.
- Graham I. (1994). *Object-Oriented Methods*. 2nd Ed. Addison Wesley Longman, Harlow, England.
- Rumbaugh J., Blaha M., Premerlani W. Eddy F., Lorensen W. (1991). *Object-Oriented Analysis and Design*. Prentice Hall, Englewood Cliffs, NJ.

### 30.7 Exercises

1. Draw the University Object model provided in chapter 16 in each of the notations described in this chapter.
2. To what extent do you feel OO methods constitute process extensions to data modelling?
3. What do OO methods offer over and above the structured methods?
4. Are OO methods applicable to all types of information systems projects?

## ***Part Five***

# ***Management***

This part of the book is devoted to a discussion of a number of critical areas surrounding the management of information systems development as an activity within organisations. It is important to emphasise that management issues have a critical bearing on the shape of information systems development, and it is for this reason that we consider them here.

We distinguish first between the management of individual development projects as compared to the general management associated with the information systems service. Then we take a look at issues of planning, particularly in the light of attempts to identify more 'strategic' uses for information systems.

Of late, the information systems service has come under fire from general management in terms of its perceived failure to deliver satisfactory levels of performance in maintaining the IS/IT infrastructure. Therefore, in chapter 34 we discuss the important issue of how one goes about evaluating the contribution that information systems make to organisations.

One side effect of the criticism of the IS service referred to above has been the growing trend to outsource either the whole or part of the IS function to outside suppliers. In chapter 35 we discuss some of the implications of this process for the management of IS.

Finally, in chapter 36 we look at some of the project-based as well as the more organisationally based ways in which the quality of information systems can be assured.

# **31 Project Management**

## **31.1 Introduction**

In this chapter we address the issues of managing information systems projects. We define a project here as being any concerted effort to develop an information system.

The issue of project management can be divided into three interrelated areas: project planning, project organisation and project control. Project planning involves determining as clearly as possible the likely parameters associated with a particular project. Project organisation concerns how to structure staff activities to ensure maximum effectiveness. Project control is concerned with ensuring that a project remains on schedule, within budget and produces the desired output.

## **31.2 Project Failure**

The business of information systems development is at least 40 years old, but managers of development departments still have to face many project failures (chapter 37). One of the biggest causes of systems failures is the project end-date cast in stone. The problem is that a software project is frequently and mistakenly seen as analogous to a traveller getting from one part of the country to another. However, the traveller has three advantages over the information systems developer:

1. The distance between geographical points is known.
2. The best route between points is easily worked out using readily available maps.
3. The traveller may have done the journey, and many like it, before.

In the systems development world:

1. The distance to be travelled is seldom known.
2. The route used is often untried and there is little basis for assessing whether it is optimal.
3. The project manager has never trodden the path before.

For these reasons there is a modern tendency to encourage systems development organisations to keep detailed records of past projects. In a sense, a map of the geography of an organisation's systems development needs to be built up. No two projects will be entirely the same. But keeping track of how much time,

effort and resources have gone into projects is an essential base from which to plan future projects.

### 31.3 Project Planning

The classic questions of project planning are: what, who, when, how, and progress? (O'Connell, 1996).

1. *What.* The product or output must be defined and the project must be broken down into a series of tasks. Adherence to a standard model of information systems development, as described in chapters 7 and 27, clearly aids this process. Standards to be used in the project, such as appropriate notations, must also be identified.
2. *Who.* Staff must be assigned to the project and responsibilities identified. The most popular method of estimating the amount of staff needed for a project is to use experienced people who have conducted similar projects in the past. Another approach is to estimate the size of the proposed product and derive a staff estimate from this figure by the application of an appropriate formula (Boehm, 1981).
3. *When.* Milestones must be identified and schedules established. Many experienced project managers recommend that a software project be divided into sequential phases and a milestone or control point established at the end of each phase.
4. *How.* A budget for the project must be constructed and resources must be allocated to the project. The likely cost of the project must be calculated and a case made for a budget for the project (chapter 34).
5. *Progress.* An effective mechanism for monitoring the progress of projects must be established. Milestones can be used as points of audit to ensure that standards are being adhered to and the project is on schedule.

### 31.4 Project Estimation

The conventional way of planning a project is to segment it into a number of stages, each of which can be managed independently. Each of these stages is further broken down into a series of tasks or activities.

One popular method of representation is to lay out a project in diagrammatic form as a network. Developed in the late 1950s, such an approach is known as the critical path or PERT (Programme Evaluation Review Technique) method (Cotterell and Hughes, 1995).

Figure 31.1 illustrates how a small project such as that involved in building a system for Goronwy Galvanising might be laid out. The nodes represent stages, the arrows represent predecessor/successor relations between stages. Each stage

can be further expanded as a diagram indicating the dependencies between activities.

An estimate is then made of the resources required to achieve each task – usually expressed in the number of person-days required. The sum of these person-days, plus a contingency factor for emergencies, is the estimated time required for each stage. Conducting this calculation for each stage will give the manager an idea of the overall person-days required. Since person-time is also the most significant cost-factor, this calculation will give the manager an idea of total approximate cost for a project.

For each activity the earliest possible start date is calculated on the basis of a schedule assigned to predecessors. A latest completion date is also calculated for each activity on the basis of the scheduled start dates for each of the activity's successors and the target completion date for the overall project. The difference between the calculated time available to complete an activity and the estimated time required to complete it is known as an activity's float. If the float is zero, the activity is said to be critical, since any delay in completing it will cause a delay in completing the final project.

This estimating approach described above is usually complicated by the fact that since no two information systems projects are ever the same there is an inherent uncertainty associated with the time required for each activity. Brooks (1997) has also discussed how using person-days, person-weeks or person-months as the central unit of estimating and scheduling can be misleading. It is tempting to infer that the progress of a project improves with the number of people assigned to it. Because of the increased communication between team-members, Brooks makes a cogent case for reversing this inference. He summarises his argument in the statement: 'adding manpower to a late software project makes it later'. The reasons for this are manyfold: people take a time to settle into a project, new personnel need to be trained, the amount of communication between team members increases the greater the size of the group, and so on.

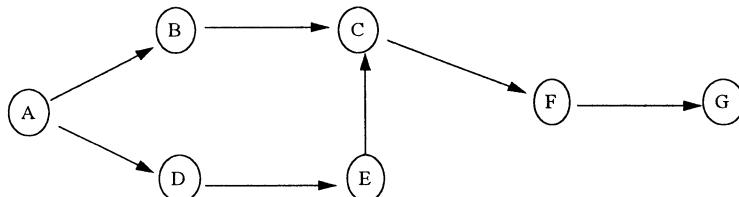


Figure 31.1: A Simple PERT

### 31.5 The Organisation of Projects

Project organisation is the issue of how to organise staff so that they produce the desired output. Essentially there are three alternatives in organising staff (Daly, 1979) (see figure 31.2):

1. *Project organisation.* Here staff are organised within project boundaries. This form of organisation encourages quick decision-making, minimises interfaces between staff and generates high identification with projects among staff. This is the style of project organisation promoted in RAD approaches (chapter 29). The disadvantages are that it works well only for small projects, the economies of scale are low, and the sharing of expertise across projects is minimal.
2. *Functional organisation.* Here staff are organised according to functional responsibilities, each function supporting a number of different projects. This form of organisation generates economies of scale, promotes the growth of specialists, and reduces the effects of staff turnover. It is hence probably the most common type of project organisation used in large development centres. The disadvantages are that it generates lots of communication across projects, it decreases the number of people with a general feel for projects, and reduces the cohesion of given projects.
3. *Matrix organisation.* Here staff are mixed across project and functional divisions. The basic organisation is functional, but a project organisation is imposed under a series of project managers. The advantages of this approach are that short-term objectives (the success of a project) are maximised via the project organisation whereas long-term objectives (such as promoting specialism) are maximised via the functional division. The major disadvantage is that project and functional needs may conflict in some organisations.

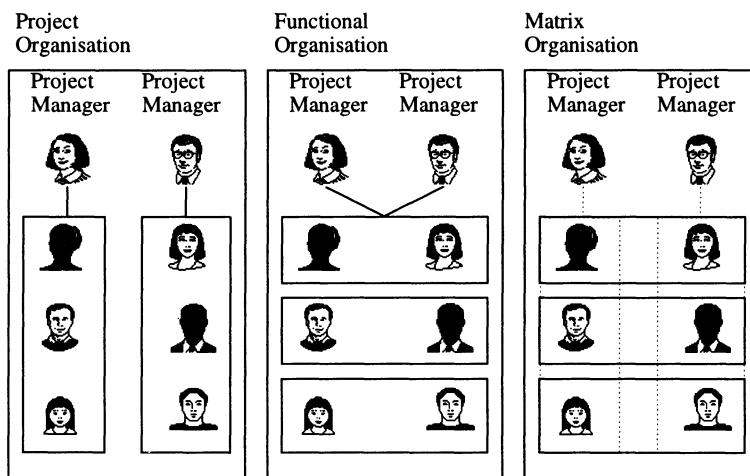


Figure 31.2: Forms of Project Organisation

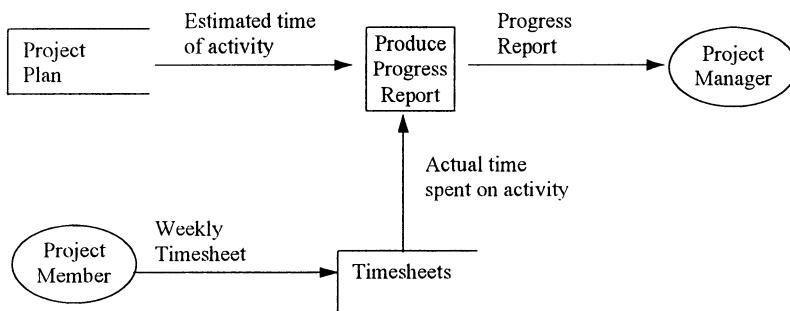
### 31.6 Project Control

The aim of project control is to ensure that schedules are being met, that the project is staying within budget and that appropriate standards are being maintained. The most important objective of project control is to focus attention on problems in sufficient time for something to be done about them. This means that continual monitoring of progress must take place.

The primary document used for the evaluation of progress is the progress report. This contains information on time estimated for each activity plotted against actual time spent. Another useful measure is an estimate as to the percentage of completeness of a project.

Time actually spent on a project is usually collected via weekly time-sheets. Such time-sheets normally indicate the tasks performed by development staff and their duration. They are also useful in highlighting time spent on unplanned work.

Many automated tools are now available to aid the project manager. Plans can be created using a graphics approach as described in section 31.4, and estimates can be associated with each activity. Time-sheet data can then be fed into the system and progress reports automatically generated. Some packages even allow the manager to perform ‘what-if’ reasoning on the project model. A schematic data flow diagram for this process is presented in figure 31.3.



**Figure 31.3: Project Control**

Progress reports can be used either by management reviews or project audits. Management reviews are scheduled opportunities for project managers to apprise themselves of the accomplishments and problems associated with a given project. Project audits are formal events scheduled into the life-cycle of a given project in which an independent audit team examines the documentation associated with a given development effort and interviews key team members.

Large-scale projects in particular may institute a change control mechanism. This involves setting up a systematic mechanism (sometimes known as configuration management) for handling all changes to a developing piece of

software. Normally a change management committee is instituted for decision-making and a change management procedure is established. The essence of this procedure is to ensure that each version of a product can be uniquely identified and that time and money is not wasted on unimportant work.

### **31.7 PRINCE**

The acronym PRINCE stands for PRojects IN Controlled Environments. PRINCE is a structured method for project management originally developed from a government-sponsored initiative in the 1970s which resulted in the method known as PROMPT (Bentley, 1992). A new version of PRINCE, PRINCE II (Bentley, 1997) was introduced in 1996.

PRINCE offers a method for defining the organisation structure for a project as well as a definition of the host company. This ensures that business interests and technical concerns are both involved. It provides the following components:

1. It defines the structure and content of project planning.
2. It defines a set of controls and reports that can be used to monitor whether a project is proceeding to plan. It also defines procedures for dealing with exceptions to the plan.
3. It actively encourages the monitoring of quality (chapter 36).

The application of PRINCE revolves around identification of the products that are to be produced by a project rather than the more usual concentration on the tasks to be completed. Identification of products leads to identification of activities needed to produce them. Such activities then trigger off the planning stage and are used to define the mechanisms for ensuring quality.

PRINCE II differs from the original prince in being process-based. It proposes a number of generic activities to be carried out during a project:

1. Starting up a project
2. Initiating a project
3. Controlling a stage
4. Managing stage boundaries
5. Closing a project
6. Directing a project
7. Managing project delivery
8. Project planning

In addition, PRINCE II defines a number of components which are applied within the appropriate activities: issues of organisation, plans, controls, products, quality, risk management, the control of change and configuration management. PRINCE II is written from the standpoint that it is most likely that there will be

a customer and supplier working together to complete a project. These two groups will frequently come from separately managed areas and often from separate organisations. Customers specify the desired outcomes, make use of the final product and in most cases fund the project. Suppliers provide resources to create the intended outcome.

Both PRINCE and PRINCE II have been designed to be used particularly with the information systems development methodology SSADM (chapter 27). It also has clearly specified relationships to information systems strategy formulation and business analysis (chapter 33).

### **31.8 Conclusion**

Even though many systematic techniques have been suggested for the process of project management, this activity is still very much an art rather than a science. Projects are about people, and people management is still the critical element in most information system projects (DeMarco and Lister, 1987). In the next two chapters we raise the level of focus from that of the management of individual projects to the management of the IS service and the planning involved in constructing a portfolio of development projects for the organisation.

### **31.9 References**

- Bentley C. (1992). *Introducing PRINCE: The Structured Project Management Method*. NCC Blackwell, Oxford.
- Bentley C. (1997). *Introducing Prince II*. NCC Blackwell, Oxford.
- Boehm B.W. (1987). Software Engineering Economics. Prentice Hall, Englewood Cliffs, NJ.
- Brooks F.P. (1997). *The Mythical Man-Month*. Revised Edition. Addison Wesley Longman. Reading, Mass.
- Cotterell M. and Hughes R. (1995). *Software Project Management*. Thomson Publishing.
- Daly E.B. (1979). 'Organising for Successful Software Development'. *Datamation*. December. 107-116.
- DeMarco T. and Lister T. (1987). *Peopleware: productive projects and teams*. Dorset House.
- Keen J.S. (1987). *Managing Systems Development*. (2nd Ed.). John Wiley, Chichester.
- O'Connell F. (1996). *How to Run Successful Projects II: the silver bullet*. Prentice Hall, Hemel Hempstead.

**31.10 Exercises**

1. At what point do you think a critical mass is reached in terms of project group size?
2. Discuss three of the main problems arising in the management of large project groups?
3. Which form of organisation do you think is most prevalent in information systems departments: project organisation; functional organisation or matrix organisation?
4. What sort of timings, effort and resource information should be kept in an organisation's project experience-base?
5. Even the most carefully planned of projects fail. Suggest some reasons.

# **32 Information Systems Management**

## **32.1 Introduction**

Planning is the process of determining what to do over a given time-period. Managing is the process of executing, evaluating and adapting plans in the face of contingencies. The issue of management and how it relates to IS development is a large one. In this chapter we limit ourselves to discussing the following issues: the perceived role of the IS services function, the importance of IS/IT to the organisation and the role of risk in managing the applications development portfolio.

## **32.2 Levels of Management**

Michael Earl has distinguished between three forms of management relevant to IS and IT: information systems management, information technology management and information management. These three forms are distinguished on the basis of the basis of the primary objective, the basis of management and the primary focus and responsibility:

<b>Domain</b>	<b>Primary Question</b>	<b>Management basis</b>	<b>Focus and responsibility</b>
<b>Information Systems</b>	What to do?  Information handling to support organisational activities	Demand driven with a business focus	Information handling applications  Concern of managers of business units
<b>Information Technology</b>	How to do it?  Providing the mechanisms to implement desired information handling	Supply of technology and support services	Maintenance of infrastructure  Concern of technical specialists
<b>Information Management</b>	Why do it?  Overall development of the organisation	Strategic direction of the organisation	Planning, regulation and co-ordination  Concern of senior management

This framework is interesting in that it suggests that a range of competencies is required in the management of information, information systems and inform-

ation technology, and that the locus of such forms of management should logically be sited at various levels within organisations. It also suggests that organisations need to have three levels of planning in place: an information strategy, an information systems strategy and an information technology strategy (chapter 33).

### **32.2.1 Information Management**

One consequence of the information society is the increasing importance of information to modern organisations. Not many years ago, the information systems was seen largely as a service activity that facilitated, but did not greatly influence, the operation of the majority of companies. With the exception of companies that were in the information systems business themselves, companies tended to treat computing as a necessary expense item to be controlled in a similar manner to water and electricity.

Over the past few years this conception of information systems as service tools has begun to change. Businesses have begun to consider how information technology can enhance the strategic competitiveness of companies in a rapidly changing market (chapter 33).

Earl (1989) has plotted some of the micro-revolutions making up the global information revolution in companies. He characterises the history of business computing into two eras: what he calls the era of data processing (DP) and the era of information technology (IT). Each era is distinguished by its position on nine dimensions.

1. *Financial attitude to IT.* In the DP era the financial attitude towards computing was one of cost; in the IT era it is one of investment.
2. *Business role of IT.* In the DP era computing was seen as mainly a support activity for company business; in the IT era it is seen as critical to company performance.
3. *Applications orientation of IT.* In the DP era the applications orientation was primarily tactical; in the IT era it is strategic.
4. *Economic context for IT.* In the DP era computing had a neutral role in changes in the business sectors of national and international economies; in the IT era computing and communications form one of the most important enabling technologies for economic change.
5. *Social impact of IT.* In the DP era the social impact of IT was limited; in the IT era it is all pervasive.
6. *MIS thinking on IT.* In the DP era management information was seen as secondary to the major information areas of business; in the IT era, management information systems are seen as central.
7. *Stakeholders concerned with IT.* In the DP era few people were involved in, or wished to be involved in, deciding information systems strategy; in the IT era the stakeholders are many and various.

8. *Technologies involved in IT.* In the DP era IT meant computing; in the IT era it means any technology concerned with information.
9. *Management posture to IT.* In the DP era the management posture was one of delegation; in the IT era it is one of leadership.

Earl and many others have been particularly interested in the last change. The discipline of information management directs itself to answering the question: what knowledge will enable management to take an effective role in determining the strategic employment of IT?

### **32.3 The Role of the IS Service**

As was mentioned in chapter 6, many organisations are rethinking the role of the IS service. In very general terms, the IS service has to adapt to, or deal with, a number of areas of tension within modern IS and its iteration with organisations. In this section we consider three:

1. Innovation and control.
2. IS services dominance and user dominance
3. Development maturity

#### ***32.3.1 Innovation and Control***

It is important that organisations maintain an effective balance between the control of information technology and the expanding need for innovation in the use of such technology.

It is useful to conceive of the process of applying IT within organisations as an assimilation process. Nolan originally distinguished four phases or stages of IT assimilation within organisations. In a later paper, Nolan (1990) expanded this idea into a six-stage model:

1. *Initiation.* This phase is characterised by the introduction of IT for cost savings, with IT belonging to business units. There is a lack of management and user interest in IT, but a steady expenditure growth on IT.
2. *Contagion.* This phase is characterised by a blossoming of IT assimilation into new areas. However, this growth is unmanaged and management and control of IT is devolve to technologists. Consequently there is a steep rise in expenditure on IT.
3. *Control.* This phase is characterised by senior management concern over the growth in IT spending. As a consequence, formalised control is introduced, and a concentration on information systems to save money not to make money is introduced. However, IT expenditure continues to rise.

4. *Integration*. This phase is characterised by a gradual lowering of controls to encourage innovation, reorganisation of IS/IT to bring it closer to the business, and a steep rise in expenditure as the organisation attempts to implement infrastructure projects.
5. *Data administration*. This phase is characterised by shared data and common systems across the organisation, business driven development, and a steady rise in expenditure
6. *Maturity*. This phase is characterised by the introduction of strategic planning of IS/IT, data resource management, and appropriate IS/IT expenditure.

Phases 1 and 2 of this model involve innovation, phases 3 and 4 involve control, and phases 5 and 6 involve strategic direction (illustrated in figure 32.1). A given organisation may be at different levels in terms of different information technologies on this sequence. For instance, the organisation may be in phase 4 so far as relational database technology is concerned but only at phase 1 as far as object-oriented systems are concerned. It is therefore important that the IS planning process (chapter 33) take these possible differentials into account.

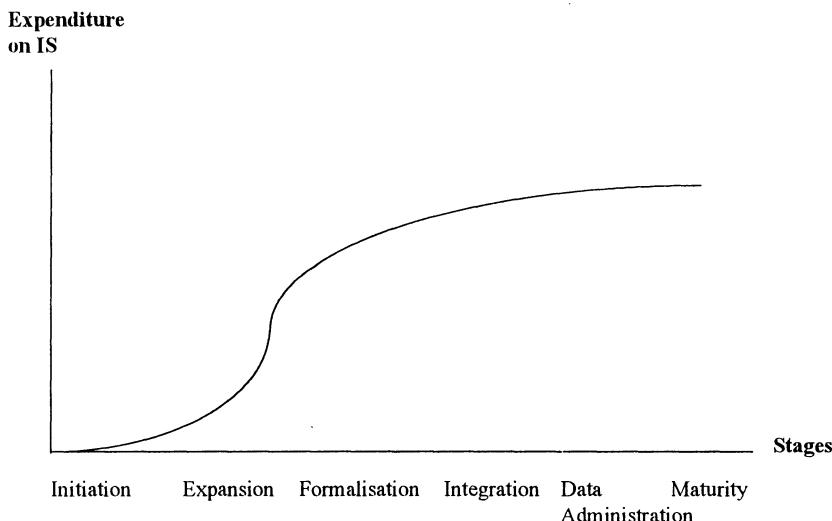


Figure 32.1: Nolan's Stages of Growth Model

#### *The New Technology Group*

The management of innovation and the management of control demand two different sets of competencies. Cash *et al.* (1992) suggest setting up a specific group to address innovation issues. This should be separate from the normal development group. The main task of such a group is to be exploratory and experimental in the investigation of new technology applications within the

organisation. Because of its very nature, the management of such a group must be more informal than development or maintenance groups.

### **32.3.2 IS Services Dominance and User Dominance**

In many organisations there exists an inherent tension between the IS service and user dominance in the areas of technology acquisition and IS development.

In some organisations, user departments have been given a large amount of effective control over acquisition and development issues. A number of pressures have contributed to this situation:

1. User demand for information systems has frequently been far greater than the capability of a centralised IS service to deliver.
2. IS staff sited at user sites are held to be more in touch with the particular needs of such sites. IS staff can develop particular experience of the IS needs of user sites.
3. User domination of IS can help address elements of the perceived communications problem between technical and non-technical staff.

The possible problem with user dominance is that short-term need fulfilment on the part of user departments may actually be made at the expense of the long-term orderly management and development of IS. This may be particularly important to ensure the integration and interoperability of information systems.

In terms of IS services dominance some of the pressures are as follows:

1. A centralised IS services function enhances the capability of the organisation to recruit and retrain highly skilled and highly motivated IS staff.
2. A centralised IS service can effectively consider integration and interoperability issues across the organisation.
3. Centralised IS development and delivery may be more cost effective in, for instance, being able to negotiate discounted purchases with hardware and software suppliers through economies of scale.

The main danger is that a centralised IS service can become too preoccupied with technological and orderly development issues at the risk of slow response to user requests.

### **32.4 Five Levels of Maturity**

From the standpoint of software engineering a framework has been proposed for assessing the maturity of an organisation in terms of its information systems function. An organisation is classified on one of five levels of 'maturity':

1. *Initial level.* There is no formal method, no consistency, no standards on how information systems should be built. Every software developer considers himself or herself to be an artist or craftsman.
2. *Repeatable level.* There is some consensus within the organisation as to the appropriate way to develop information systems. No attempt has been made however at formalisation or documentation of procedure. Success largely depends on the intuitive skills of project managers.
3. *Managed level.* There is a formal, documented process for developing information systems and managing their development. The organisation continually refines and updates its methods.
4. *Measured level.* The organisation has instituted software metrics for measuring the process of developing information systems and its associated products.
5. *Optimising level.* The organisation uses the measurements from the previous level to improve the development process.

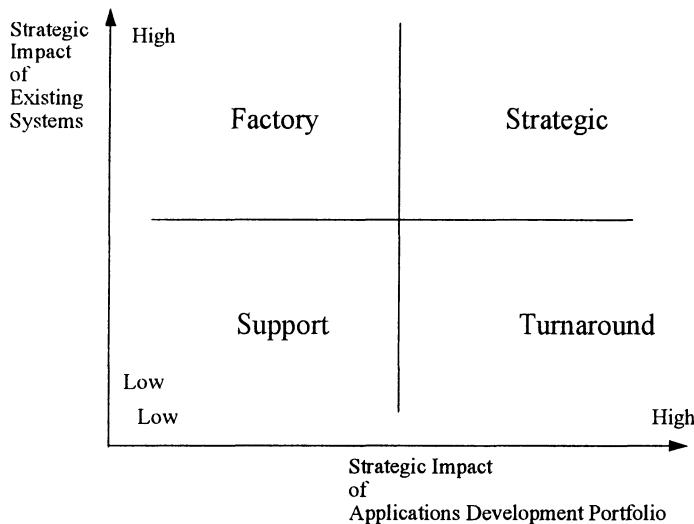
The first three levels are obviously designed to encourage the use of systematic approaches to information systems development with associated project management and quality assurance. The last two are obviously designed to foster the exploitation of software metrics (chapter 36).

### 32.5 The Importance of IS to Organisations

In chapter 3 we described a number of models of organisation that have a crucial bearing on the way in which we approach IS/IT. Another way of looking at organisations is in terms of the relative importance of IS/IT to the activity of the organisation. Cash *et al.* (1992) provide a useful way of conceptualising organisations in terms of:

1. The importance of existing IS to the strategy of organisations.
2. The importance of the IS development portfolio to the strategy of organisations.

Considering these as two dimensions, four organisational possibilities are illustrated in figure 32.2:



**Figure 32.2: Cash et al's Taxonomy of Organisations**

### ***Strategic***

Strategic organisations are those in which smooth functioning of existing IT activity is critical to their daily operation. Also, applications under development are critical to the organisation's future competitive success. Many organisations in the financial industry are clearly in this sector. For instance, IT is at the heart of modern banking. Future developments in the area of internet banking are likely to be critical to the industry.

### ***Turnaround***

Such organisations require considerable amounts of IT support, but the organisation's activities are not absolutely dependent on the uninterrupted, cost-effective functioning of IT support to achieve its short-term or long-term objectives. Applications under development, however, are absolutely vital to the long-term health of the organisation. Many organisations in the retail industry have traditionally been in this camp. However, for the large supermarket chains IT and the development of applications has become critical to competitive success.

### ***Factory***

For smooth operation, the organisation is heavily dependent on cost-effective, reliable IT. The IS development portfolio however is heavily dominated by maintenance work, and the applications under development, though important, are not fundamental to the ability of the organisation to compete. The manufacturing sector has implemented many systems such as just-in-time manufacturing systems that are heavily reliant on IT. However, it is currently unclear in many organisations how IT can stimulate further improvements in manufacturing processes in the future.

### **Support**

Some organisations are not dependent on the smooth functioning of IT; nor are their applications portfolios critical to future effectiveness. Organisations in agriculture have historically not been heavy investors in IT and also are unlikely to be large investors in this technology in the future.

One should emphasise that this taxonomy is probably equally relevant to parts of organisations as well as the whole organisation. Different divisions and departments may be in different quadrants of the innovation matrix at different times. For instance, manufacturing operations may be in factory or support whereas the marketing division may be in the strategic or turnaround quadrant.

## **32.6 Managing the Applications Development Portfolio**

Cash *et al.* (1992) develop the idea of managing the applications development portfolio as being a process of risk assessment and management. The key idea is that risk is involved in all IS projects. Risk might be defined as a negative outcome that has a known or estimated probability of occurring based on some experience or theory. The idea of IS failure is clearly the negative outcome most prominent in most people's minds (chapter 37).

Risk assessment is clearly the process involved in estimating the degree of risk associated with a given project, usually at the feasibility stage of development. A number of frameworks have been generated which suggest a number of characteristics indicative of risky IT projects. Cash *et al.* (1992) suggest that there are at least three important dimensions that influence the risk of a project: project size, experience with the technology and project structure. In general, the smaller, more experienced and more highly structured the project, the less risk is likely to be associated with it. It is also easier to manage such projects using classic project management techniques.

However, risk is not inherently bad. Some projects may be naturally risky, but may have a high strategic payoff in relation to areas of innovation. In 'strategic' companies the balance of high risk to low risk projects may be relatively high because of the importance of IS as a competitive weapon to such companies.

## **32.7 Concerns of IS Managers**

Over a number of years a group of researchers in the US have surveyed Chief Information Officers (CIO) of American companies to document their key concerns. In 1991 (Neiderman *et al.*, 1991), the following list of issues were ranked in order of perceived importance:

1	Developing an information architecture
2	Making effective use of the data resource
3	Improving IS strategic planning
4	Specifying, recruiting and developing human resources
5	Facilitating organisational learning and use of IS technologies
6	Building a responsive IT infrastructure
7	Aligning the IS organisation with that of the enterprise
8	Using IS for competitive advantage
9	Improving the quality of software development
10	Planning and implementing a telecommunications system

In 1996 (Bracheau *et al.*, 1996), the perceived ranking of issues had subtly changed:

1	Building a responsive information architecture
2	Facilitating and managing business process redesign
3	Developing and managing distributed systems
4	Developing and implementing an information architecture
5	Planning and managing communication networks
6	Improving the effectiveness of software development
7	Making effective use of the data resource
8	Recruiting and developing IS human resources
9	Aligning the IS within the enterprise
10	Improving IS strategic planning

However, note that the focus on an effective information architecture has remained the number one concern for CIOs over a volatile period for the IS/IT industry.

### 32.8 Conclusion

In this chapter we have introduced the topic of IS management and considered a number of issues which affect the shape of IS management within organisations. It is something of a truism that management works more effectively if it has a series of plans by which it can calibrate its performance. This probably explains why the development of an effective information architecture remains the number one concern for IS managers. This topic of IS planning we cover in the next chapter.

### 32.9 References

- Bracheau J.C., Janz B.D., Wetherbe J.C. (1996). 'Key issues in information systems management: 1994-1995 SIM Delphi Results'. *MIS Quarterly*. 20(2). June.
- Cash J.I, Mcfarlan F.W., McKeney J.L. (1992). *Corporate Information Systems Management*. 3rd Ed. Richard Irwin.
- Earl M.J. (1989). *Management Strategies for Information Technology*. Prentice Hall, Hemel Hempstead, UK.
- Neiderman F., Bracheau J.C., Wetherbe J.C. (1991). 'Key issues facing IS Managers'. *MIS Quarterly*. 17(4).
- Nolan R.L. (1990) 'Managing the Crisis in Data Processing'. Reprinted in *The Information Infrastructure*. Harvard Business Review.

### 32.10 Exercises

1. Determine the perceived role of the IS services function within some organisation known to you. How does it conceive of its role in relation to the issues of innovation/control and IS/User dominance of IT?
2. In terms of a new technology group, why do you think the management of such a group should be more informal than say an IS development or maintenance group?
3. Consider some organisation known to you. Find out: Does it have a specialist IT department? What is its make-up? Does it conform to a traditional IT department or does it have a different structure? How would you rate it on the maturity levels and why? Is the IT department seen purely as a service function or in a more strategic sense?
4. Nolan's six stage model was developed a number of years ago. Discuss to what degree it is still relevant today.
5. Choose a market sector such as food retail, banking or manufacturing. Use Cash *et al.*'s framework to characterise the importance of IS/IT to the sector.

# **33 Information Systems Planning**

## **33.1 Introduction**

Information systems planning is the process of deciding upon the optimal information, information systems, and information technology architecture for some organisation.

1. *Information architecture*. This consists of activities involved in the collecting, storage, dissemination and use of information within the organisation.
2. *Information systems architecture*. This consists of the information systems needed to support organisational activity in the areas of collection, storage, dissemination and use.
3. *Information technology architecture*. This consists of the hardware, software, communication facilities and IT knowledge and skills available to the organisation.

The objective of IS planning is to develop strategy in each of these three areas. The practical output of IS planning is a document or documents which describes strategy in these three areas. Whilst acknowledging the distinctions raised above, we shall use the term information systems strategy to include issues relating to information and information technology planning. An information systems strategy can be described as being the structure within which information, information systems and information technology are applied within the organisation. Such a strategy should establish an organisation's long-term infrastructure which will allow information systems to be designed and implemented efficiently and effectively. An information systems strategy is particularly directed at avoiding fragmentation, redundancy and inconsistency amongst information systems in the organisation. The strategy should also be directed at ensuring an effective 'fit' between an organisation and its information systems (chapter 38).

### ***Example***

A university can be considered as an organisation to which IS planning may be applied:

1. *Information architecture*. A university needs to collect and store information on information pertaining to teaching, research information and consultancy information.
2. *Information systems architecture*. On a very broad level a university needs information systems to support the teaching, research and consultancy

processes. In terms of teaching, for instance, a university needs to record information about its student population, the courses and modules that students are currently taking and the grades that students have achieved.

3. *Information technology architecture.* A university needs an integrated communications infrastructure enabling technology such as e-mail and the internet. It also needs standards established in terms of hardware and software for both administrative and teaching purposes.

### 33.2 Top-down and Bottom-up IS Planning

Most IS planning within organisations has been conducted using a project-oriented approach. The emphasis has been technical, rather than organisational. The main aim has been ensuring the success of a particular IS development project. A related approach seeks to blend organisational and technical objectives by considering IS planning on a needs basis. This is a bottom-up approach in the sense that when a specific organisational need calls for a new information system, some form of formal project planning is set in motion.

These two approaches are generally contrasted with more top-down approaches in which an information systems strategy is either produced in parallel with a business strategy, or becomes an inherent part of the business planning process. It is this form of IS planning which we primarily consider in this chapter.

### 33.3 The Advantages and Value of IS Planning

One of the inherent messages from work in this area is that IS planning must take place within the context of general business planning. In this sense:

1. IS planning is important in ensuring that there is a close match between the proposed direction of an organisation and its information services.
2. IS planning can be used to ensure that IS projects more clearly focus on business rather than purely technical results.
3. IS planning should lead to greater integration of both current and future information systems.
4. Planning facilitates effective IS resource allocation. It becomes easier to estimate the effect (risk) of proposed IS projects in terms of business objectives and evaluate current systems in terms of effectiveness.
5. Good planning is a necessary part of good management of information systems.
6. IS planning may improve the ability of an organisation to react better to unforeseen circumstances.

### ***33.3.1 The Value of IS Planning***

Cash *et al.* (1992) make the important point that the value of IS Planning varies with the type of organisation (chapter 32):

1. Since *strategic* companies are critically dependent on smooth IS functioning both now and in the future, such organisations benefit from considerable amounts of IS planning.
2. *Turnaround* companies also need a substantial IS planning effort since although current organisational effectiveness is not critically dependent on IS, future performance is critically dependent on future IS.
3. In *factory* organisations mass effort in IS planning is probably not needed, although year by year operational planning is still essential.
4. In *support* organisations there is a tendency to assume that such organisations need little if any IS planning. The danger, however, is that opportunities may arise in evolving technology that may be missed by such organisations.

## **33.4 The IS Planning Process**

To develop an IS strategy an organisation usually first evaluates a number of existing planning methodologies such as IBM's Business Systems Planning and James Martin's Information Engineering. The organisation then generally selects a methodology for IS planning or customises its own.

The organisation then usually sets up a committee of users and IS specialists. It relies on the training provided by the methodology to guide the planning study.

Next the committee carries out the multiple phases of the study, generally lasting several months, sometimes years. It uses existing documentation and interviews with business executives to define its current business processes and data. It also studies how the current information systems support these processes and data. In this area, the use of business process re-engineering approaches may be appropriate (chapter 25).

Using its documented understanding, the committee then identifies and prioritises its key IS for the future together with an implementation schedule. It prepares a report which includes a long-range plan with recommendations for hardware, software, computers and personnel support.

### ***33.4.1 Stages of IS Planning***

From commonalities amongst a number of IS planning methodologies, a series of broad stages of IS planning can be identified:

1. Assessment of information use and management.
2. Establishing a vision of how the organisation should use information.
3. Developing the information systems architecture.
4. Developing the information services strategic plan.
5. Developing information services operational plans and budgets.

In this chapter we focus on stages 1 through 3. Stages 4 and 5 are features of the IS service which has been discussed in chapters 6 and 32.

### 33.5 Information Systems Assessment

Any planning process must begin with assessment of the current situation. Current performance is compared with some set of objectives. Business and IS objectives would be expected to result from a business analysis exercise using techniques such as those described in chapters 25 and 26.

The IS planning process should begin with an assessment of the use of information, information systems and information technology in the entire organisation as well as an assessment of the work of the IS service. Such an assessment may be conducted by a committee composed of both IS professionals and user-managers. Some representation may be provided by some outside organisation, particularly IS consultancies.

An IS assessment will probably document current levels of information, IS and IT use and compare them against some set of standards, perhaps produced as benchmarks of past performance or by analysing industry norms. A technical assessment of current information systems and technology infrastructure will also form part of the picture as well as an assessment of current attitudes to information systems and IT within the organisation.

Another important part of the assessment will be a review of the mission of the IS service. This will address the important issue of what are the reasons for having an IS service. Sinclair (1986) suggests that such reasons can be classified under three major headings: efficiency, effectiveness and competitiveness:

1. *Efficiency*. Are IS services helping the organisation to remain active with the minimum of resources?
2. *Effectiveness*. Are IS services helping the organisation to spend its time doing the right things?
3. *Competitiveness*. Are IS services engaged in projects that will improve the position of the organisation in its environment?

Many organisations are clearly answering *no* to questions such as these and hence considering other options to an internal IS service such as facilities management and outsourcing (chapter 35). Outsourcing is the idea of transferring the whole or part of the IS service, particularly support activities, to outside

suppliers. Facilities management is the idea of also transferring the management of the IT function to outside contractors.

### **33.6 Establishing a Vision**

The second step in any IS planning process is to envisage an ideal state for information systems within some organisation at some future moment in time. Much of the contemporary literature on IS planning seems to be taken up with considering the competitive advantage afforded by information systems.

#### ***33.6.1 Competitive Advantage***

Competitive advantage (Porter, 1985) in this context may be defined as the important market leverage afforded by information systems. Competitive advantage comes in three forms:

1. *Cost advantage*. Establish itself as a low-cost leader in the market. This normally means concentrating on areas involved in improving overall efficiency via better planning.
2. *Differentiation*. Differentiate its product so that the marketplace perceives a product to be superior. This may be through improving quality and reliability relative to price, better market understanding, image promotion etc.
3. *Location*. Find a niche to service. This may be by placing distribution, marketing and information outlets where required.

### **33.7 Methods for Assessing Competitive Advantage**

A number of frameworks have been proposed for assessing competitive advantage afforded by information systems. We consider four:

1. Critical Success Factors
2. Five Forces Model
3. Customer Life Cycle
4. Value-Chain

#### ***33.7.1 Critical Success Factors***

Any organisation needs to identify areas in which it has relative superiority, and to use that superiority both to create barriers to entry as well as to launch strategic offensives. One popular method of doing this is the critical success

factor concept or CSF. A CSF is a factor which is deemed crucial to the success of a business. Consequently, CSFs are those areas that must be given special attention by management. They also represent critical points of leverage for achieving competitive advantage. There are normally only a few CSFs – perhaps between 3 and 8 – for each organisation. CSFs follow the 80/20 rule in that only a few issues really count in terms of organisational effectiveness.

A CSF for a chain of high street jewellers is likely to be the location of its outlets; a CSF for a health authority is likely to be the quality or standard of service that it gives to its customers – patients.

CSFs are usually contrasted with CFFs or critical failure factors. A CFF is an aspect of the organisation, the poor management of which is likely to precipitate organisational failure. A CFF for the high street jeweller chain is likely to be a high amount of shrinkage in consumer demand. A CFF for a health authority might be poor co-ordination of its staff, particularly subcontracted staff.

CSFs and CFFs are useful ways of identifying areas for the maximal application of information systems. A high street jeweller chain, for instance, would benefit from an information system which enabled managers to select optimal locations for their stores based on factors such as population density, state of local economy etc. In contrast, a health authority would benefit from an information system which ensured the efficient work scheduling of nurses.

### **33.7.2 Five Forces**

Another framework for assessing competitive advantage is based on the work of Porter and Millar (1985). They argue that a successful firm shapes the structure of competition by influencing five primary forces:

1. Industrial rivalry, the competitive position of rival organisations
2. Customer bargaining power
3. Supplier bargaining power
4. Barriers to entry, threat of new entrants
5. Threat of substitutes

These five forces are illustrated in figure 33.1. Many combinations of these factors, such as low industrial rivalry, high barriers to entry, and low buyer bargaining power, can lead to sustainable, above-average, long-term profits. Information systems can be used to achieve such goals by:

1. Changing the basis of competition
2. Strengthening customer relationships
3. Overcoming supplier problems
4. Building barriers against new entrants
5. Generating new products

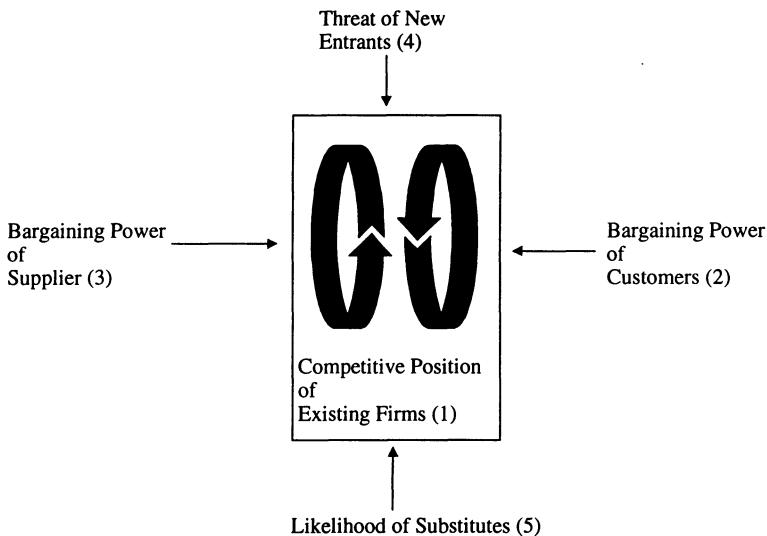


Figure 33.1: Porter's Competitive Forces

For instance, Shaw and Stone (1988) illustrate how the use of customer databases for marketing and sales can have a strategic impact:

1. Win customers from competition by using large customer database
2. Use of market research databases to target customer 'need'
3. Use of telemarketing, direct mailing to improve sales
4. Cost of setting up and maintaining accurate customer database as a barrier to entry
5. Electronic shopping as a new product

### 33.7.3 The Customer Resource Life-cycle

Ives and Learmonth (1984) define an application as being strategic if it changes a company's product or the way a firm competes in its industry. They use the idea of a customer resource life-cycle to identify potential strategic applications. This model considers a firm's relationship with its customers and how this relationship can be changed or enhanced by the strategic application of IT.

The customer resource life-cycle is discussed as a cycle of 13 stages:

1. *Establish requirements*. To determine how much of a resource is required.
2. *Specify*. To detail the attributes of the required resource.
3. *Select a source*. Locate an appropriate source for the resource.
4. *Order*. To order a quantity of the resource from the supplier.

5. *Authorise and pay.* Before a resource can be acquired, authority for the expenditure must be obtained and payment made.
6. *Acquire.* To take possession of the resource.
7. *Test and accept.* The customer must verify the acceptability of the resource before putting it to use.
8. *Integrate.* The resource must be added to an existing inventory.
9. *Monitor.* Ensure that the resource remains acceptable while in inventory.
10. *Upgrade.* If requirements change, it may be necessary to upgrade resources.
11. *Maintain.* To repair a resource, if necessary.
12. *Transfer or dispose.* Customers will eventually dispose of a resource.
13. *Account for.* Customers must monitor where and how money is spent on resources.

Ives and Learmonth give examples of the strategic application of information technology at each of these stages. For instance, a Swedish dairy co-operative produces for their retailers pro-forma orders detailing estimated requirements. The customer then has the option of using the prepared orders or changing them. Such orders assisted the retailer at both the specification and requirements stage of the customer life-cycle.

The idea of a customer resource life-cycle is equally relevant to public-sector organisations such as the NHS. In this setting, the patient is the primary health service customer. The patient consumes a health service resource. A possible resource life-cycle might then be:

1. *Requirements specification.* IT can be used as a means to aid the GP and the patient in establishing what health resource is required and what quantity of the resource is required.
2. *Selection.* A series of options can be presented to the patient regarding, for instance, a number of hospitals able to offer a given treatment.
3. *Order.* The GP would be able to query a hospital's elective admission system to find out when a stay is feasible at the hospital. If it suits the patient a provisional booking could be made on the system.
4. *Authorise and payment.* Appropriate routines would update the treatment accounts of the hospital and the budget of the general practice.
5. *Acquire.* When the patient arrives for his stay in hospital his details would be transferred from the GP's system to the patient administration system of the hospital.
6. *Test.* The patient would be given a listing of the proposed treatment and be requested to sign it off.
7. *Integrate.* The consumption of the health care resource would be added to the patient history held at the centralised register of the region.
8. *Monitor and upgrade.* Preliminary investigation may change the initial prognosis leading to modification of the original health care plan. Any changes or additions to the plan would be recorded by hospital systems.

9. *Maintain.* Regular checkups will be made of patients to ensure effective functioning. Such checkups will be notified to patients by the GP system, and the results recorded by the system.
10. *Accountability.* The patient receives a report of every treatment concluded with the cost it has associated with the health service budget.

### 33.7.4 The Value Chain

The customer resource life-cycle has much in common with an idea generally attributed to Porter of an organisation's value chain. An organisation's value chain is a series of interdependent activities that brings a product or service to a customer. Such activities are organised in two major types: primary and secondary activities. This is illustrated in figure 33.2.

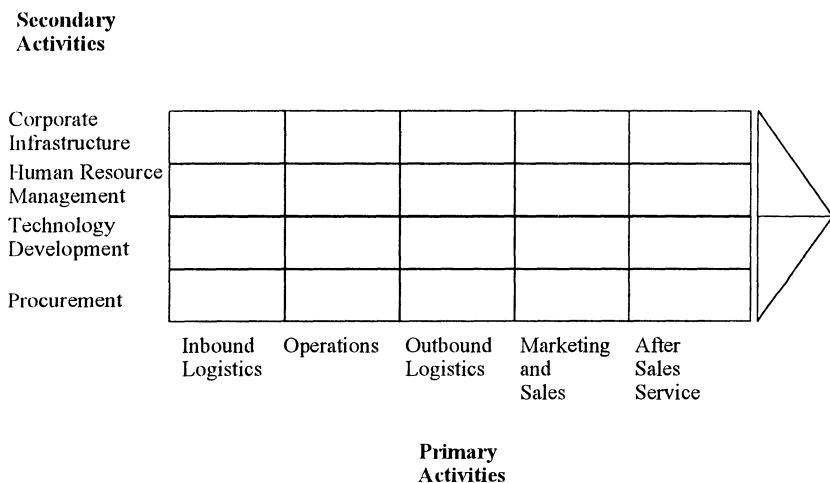


Figure 33.2: The Value Chain

Primary activities consist of the following:

1. *Inbound logistics.* Materials receiving, storing and distribution to manufacturing premises.
2. *Operations.* Transferring inputs into finished products.
3. *Outbound logistics.* Storing and distributing products.
4. *Marketing and sales.* Promotions and sales force.
5. *Service.* Services that maintain or enhance product value.

Secondary activities consist of the following:

1. *Corporate infrastructure.* Support of entire value chain such as general management, planning, finance, accounting, legal services, quality management.
2. *Human resource management.* Recruiting, hiring, training and development.
3. *Technology development.* Improving the product and manufacturing process.
4. *Procurement.* Function or purchasing input.

IT can affect any of the activities on the value chain:

#### *Primary Activities*

1. *Inbound logistics.* Placing order-entry links to suppliers so that they make stock levels available to purchasing.
2. *Operations.* Computer-controlled manufacturing.
3. *Outbound logistics.* Airline reservation systems.
4. *Marketing and sales.* Pharmaceutical company placing order-entry terminals in pharmacies.
5. *After-sales service.* Elevator company maintaining a maintenance recording system.

#### *Secondary Activities*

1. *Corporate infrastructure.* E-mail for corporate co-ordination.
2. *Human resource management.* On-line human resources database.
3. *Technology development.* CAD/CAM.
4. *Procurement.* Electronic bulletin board of parts.

### **33.8 Strategic Information Systems**

A strategic information system might be defined as any IT application which directly assists an organisation in achieving its corporate strategy (Jackson, 1997). Corporate strategy involves the relationship of an organisation with its environment. Strategic decision-making involves determining what business a firm is in and what kinds of business it will seek to enter. When an information system is used to achieve competitive advantage it may be referred to as an SIS.

#### *Case*

Thomson Travel developed a communications network compatible with the British Videotex network (Prestel) for the purpose of providing direct reservations to its customers (travel agencies). As a direct result, Thomson's sales in a given product segment doubled in one year.

**Case**

The supermarket chain Tesco introduced a customer loyalty card that awarded discounts to customers on the basis of how much the customer purchased from the retail chain. Tesco is now attempting to place a customer database in each store such that discounts can be awarded at the time of purchase.

### 33.9 The Changing Nature of Strategic IS

An IS is only likely to remain strategic for a limited period of time. Usually the competitive advantage afforded by a given information system will soon be eroded because of emulation by competitors. For example, most travel agents have links to a range of travel companies through information technology. Also, most of the supermarket chains in the UK have now emulated Tesco's loyalty card.

Therefore, Ciborra (1994) has argued that this means that SIS are unlikely to be created using top-down approaches. Effective SIS must continually filter up from the bottom to the top layers of the organisation through a continuous process of 'bricolage' – creative experimentation. This is similar to the conclusion of Currie (1994) who contrasts the formal models of the IS strategy planning process with what she calls the 'adhocracy' of actual IS planning.

### 33.10 IS Services Strategic Plan

An IS services strategic plan sets goals for this organisational function within some future timeframe. An IS services strategic plan will normally comprise:

1. A statement concerning the organisation of the IS services function. For instance, prior planning processes should have already have established the basis for the financial control of the IS service. Two main options are normally considered: that of an unallocated cost centre in which a budget is delivered directly to the IS service and consequently user departments do not pay directly for IS work; that of an allocated cost centre and charge-out, in which IS budgets are delivered to user departments and the IS service charges user departments for services.
2. An identification of how IS can align itself with business strategy. Cash *et al.* (1992) discuss the important difficulty of aligning IS strategy with business strategy in that business strategy generally considers a 1 year frame for reference whereas IS strategy must consider a 3-5 year frame of reference. One attempt to improve the coupling between business and IS is to create a seat on the board for a chief information officer or CIO.
3. A portfolio of development plans for IS.
4. A portfolio of maintenance plans for IS.
5. A portfolio of operational and support plans for IS.

### 33.11 IS Services Operational Plans

Operational plans relate the activity of the IS service with goals and budgets. Operational plans would normally include:

1. A description of development projects with associated resource implications and costings.
2. A description of maintenance activity associated resource implications and costings.
3. A description of operational effort associated resource implications and costings.
4. A description of IS planning and management activity associated resource implications and costings.

### 33.12 Conclusion

In summary, information systems planning is the process of deciding upon the ideal information systems architecture for some organisation. An information systems architecture (ISA) can be described as being the structure within which information, information systems, and information technology are applied within the organisation.

IS planning has been discussed in this chapter as a top-down planning process involving the following stages: assessment of information use and management; establishing a vision of how the organisation should use information; developing the information systems architecture; developing the information services strategic plan; developing information services operational plans and budgets.

Establishing a vision involves comparing current IS performance against some set of objectives. Developing an architecture is normally discussed in terms of assessing the use of information systems for competitive advantage. Four frameworks were discussed in this context: Critical Success Factors; Five Forces Model; Customer Life Cycle; the Value Chain.

Recent work has suggested that although such frameworks are valuable, they fail to take account of the informal/*ad hoc* nature of much IS planning work.

### 33.13 References

- Ciborra C. (1994) 'The Grassroots of IT and Strategy'. In Ciborra C. and Jelassi T. *Strategic Information Systems: a European Perspective*. John Wiley, Chichester, UK.
- Currie W. (1994) 'The Strategic Management of a Large-Scale IT Project in the Financial Services Sector'. *New Technology, Work and Employment*. 9(1).

- Ives B. and Learmonth G.P. (1984). 'The Information System as a Competitive Weapon'. *CACM* 27(12). 1193-1201.
- Jackson Ivan F. (1997). *Information Systems – The Customer Service Focus*. Macmillan press, Basingstoke.
- Porter M.E. (1985). *Competitive Advantage: creating and sustaining superior performance*. Free Press.
- Porter M.E. and Millar V.E. (1985). 'How Information Gives you Competitive Advantage'. *Harvard Business Review*. 63(4). 149-160.
- Remenyi D. (1991). *Strategic Information Systems Planning*. NCC/Blackwell.
- Shaw R. and Stone M. (1988). *Database Marketing*. Gower, London.
- Wainwright M.E., DeHayes D.W., Hoffer J.A., Perkins W.C. (1994). *Managing Information Technology*. Macmillan, New York. 2nd Ed.

### **33.15 Exercises**

1. Assess the current state of information systems in some organisation known to you.
2. Develop a small analysis of how information systems might be used to improve the strategic position of some organisation using one or more of the frameworks discussed in the unit.
3. From the frameworks discussed, choose what you feel to be the most effective framework and briefly justify your choice.

# **34 Information Systems Evaluation**

## **34.1 Introduction**

In recent years a number of questions have been frequently voiced by general management within organisations in relation to information systems and information technology:

1. Do we know how much is currently spent on IS?
2. What value results from this spending?
3. How should IS alternatives be justified/ prioritised/ financed?
4. Why do IS budgets continue to rise while IT unit costs continue to fall?
5. How can we regain our belief in IS returns?

In this chapter we look at the evaluation of information systems investments by organisations, and in particular discuss a number of techniques utilised in the context of evaluating investments in IS/IT.

## **34.2 The Costs and Benefits of IS**

A distinction is frequently made between formative evaluation and summative evaluation. Formative evaluation involves assessing the shape of an IS whilst in the development process itself. Formative evaluation may be used to make crucial changes to the design of an information system. Such a form of evaluation is particularly relevant in the context of iterative approaches to development such as RAD (chapter 29). Summative evaluation occurs after an IS has been implemented. For this reason it is sometimes referred to as post-implementation evaluation.

Clearly evaluation can take place on a number of levels:

1. *Functionality*. Does the information system do what is required?
2. *Usability*. Is the information system usable by its intended population?
3. *Utility*. Does the information deliver key business benefit for the organisation?

In this chapter we concentrate on summative evaluation of the utility of an IS. This form of summative evaluation is fundamentally concerned with assessing the costs and benefits of introducing an IS. In a sense, IS evaluation is a technique used by organisations to continuously identify and maintain the balance between IS costs and benefits.

### 34.3 IS Costs

It is useful to make the distinction between two types of costs associated with IS projects: tangible or visible costs and intangible or invisible costs. Tangible costs are frequently referred to as visible costs because they are reasonably straightforward to measure. Intangible costs are frequently referred to as invisible costs because most organisations experience difficulty in assigning actual measurable quantities to such costs. Keen (1990) in his study of a number of large-scale IS projects illustrates the way in which tangible and intangible costs are distributed in relation to typical phases in a bespoke as compared to a packaged software development effort:

#### In-House Software Development Project

Phase	Visible (%)	Hidden (%)
Planning and Design	40	0
Programming	10	0
Testing	10	20
Installation	10	10
<b>Total</b>	<b>70</b>	<b>30</b>

#### Purchased Software Package

Phase	Visible (%)	Hidden (%)
Planning and Design	0	40
Programming	25	0
Testing	5	10
Installation	0	20
<b>Total</b>	<b>30</b>	<b>70</b>

The interesting point here is that far more of the costs are likely to be intangible in a packaged software development than in a bespoke development project.

A typical list of cost-types associated with IS projects is given below:

1. *Hardware costs*: computers, printers, storage, accessories etc.
2. *Software costs*: off-the-shelf packages, bespoke software, development tools etc.
3. *Installation costs*: data entry, data conversion etc.
4. *Environmental costs*: wiring, furniture, air-conditioning etc.
5. *Running costs*: electricity, communication costs etc.
6. *Maintenance costs*: on hardware and software.
7. *Security costs*: risk management and disaster recovery mechanisms.
8. *Networking costs*: network hardware, software and maintenance.
9. *Training costs*: frequently underestimated.

#### 10. *Wider organisational costs*: new salary structures, management etc.

The important point to make in relation to this list is that the costs of an IS must be taken into account over its entire life, not solely in terms of its development cost. Frequently, organisations forget that the initiation of an IS means a permanent commitment to costs in terms of continuing resources being committed to the running and maintenance of the IS.

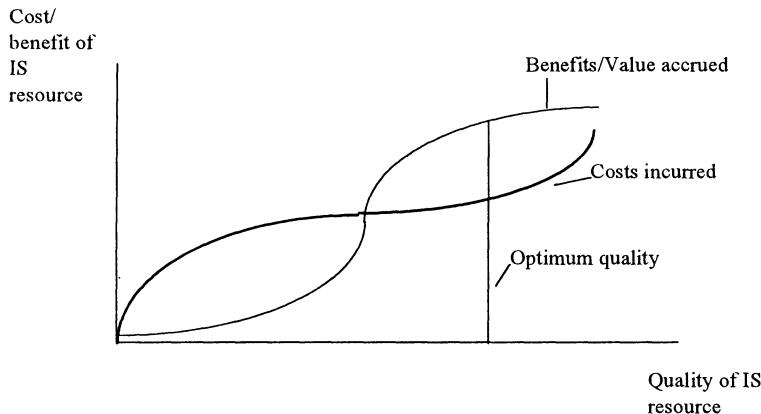
### 34.4 IS Benefits

IS benefits concern the value that the organisation gains from having an IS. Again we may distinguish between tangible and intangible benefits. IS have frequently been initiated with the objective of gaining tangible benefits such as reducing staff count or increasing productivity. Intangible benefits are more generally associated with issues of organisational efficiency. More recently, people have started to argue that intangible benefits such as increasing customer satisfaction or building better links with suppliers have equal relevance in investment decisions. Intangible benefits are more generally concerned with organisational effectiveness. Below we list some of the classic benefits associated with information systems:

1. *Accuracy*: increased accuracy of information
2. *Quality*: better quality information
3. *Usability*: more useful information
4. *Flexibility*: ability to use information more flexibly
5. *User satisfaction*: increased work satisfaction
6. *Functionality*: more effective working
7. *Reliability*: more reliable service
8. *Utilisation*: more information used
9. *Relevance*: more focused information
10. *Productivity*: increased levels of working
11. *Security*: more secure information
12. *Profitability*: more money
13. *Speed*: getting information quicker
14. *Volume*: more information

### 34.5 Information Value

It is useful to conceive of the balancing of IS costs and benefits as being a process of determining optimal information value. Figure 34.1 illustrates this process.



**Figure 34.1: Balancing IS Costs and Benefits**

On this graph we plot the costs and benefits of having an IS resource against the quality of that IS resource. In these terms, initially costs will far outweigh benefits. Also, there will usually be a rise in costs when quality is increased. However, at some point benefits will start to exceed costs. The assumption is that there is some optimum point for quality in relation to information value defined by that point on the graph where the difference between value accrued and costs incurred is at its greatest.

### 34.6 Techniques for IS Evaluation

Most of the established techniques for evaluating information system investments focus on tangible costs and benefits. Two of the most popular are return on investment and payback period. It must be said that most IS practitioners still seem to rely mainly on one or other of these 'hard' evaluation techniques.

In recent years, a number of techniques have been suggested for focusing on intangible as well as tangible benefits. One of the most popular of such approaches is that known as Information Economics.

#### 34.6.1 Return on Investment

The return on investment (RoI) associated with an IS project is calculated using the following equation:

$$\text{RoI} = \text{average (annual net income / annual investment amount)}$$

Hence, to calculate an RoI one must be able to estimate the income accruing from the introduction of an IS and the cost associated with an IS for a period into

the future. The average of this ratio of annual costs to benefits is then taken to indicate the value of the IS to the organisation.

Assume, for instance, that we need to calculate the ROI for the Goronwy Galvanising information system. The table below contains estimates as to the amount of extra income generated by the introduction of the system plotted against the costs associated with the development and maintenance of the system. We have assumed here that the system will take two years to complete and that the development costs will be of the order of £30,000. We also assume that the life of the project will be ten years.

Year	Income	Investment	Income/ Investment
1	£0	£200,000	0
2	£0	£100,000	0
3	£50,000	£10,000	5
4	£300,000	£10,000	30
5	£500,000	£10,000	50
6	£600,000	£11,000	55
7	£600,000	£11,000	55
8	£600,000	£12,000	50
9	£600,000	£12,000	50
10	£500,000	£13,000	38
11	£400,000	£13,000	31
12	£300,000	£14,000	21

$$\text{ROI} = \text{average (annual net income / annual investment amount)} = 32$$

### 34.6.2 Payback Period

Payback period still assumes that one is able to estimate the benefit of the introduction of an IS to the organisation for a number of years ahead. Benefit is measured in terms of the amount of cash inflow resulting from the IS. Payback is then calculated on the basis of:

$$\text{Payback} = \text{Investment} - \text{cumulative benefit (cash inflow)}$$

The payback period is equal to the number of months or years for this payback figure to reach zero. Clearly the assumption here is that those systems that accrue financial benefits the quickest are the most successful. In terms of the example above, if we assume that the cumulative benefit is the same as the income generated (which may not always be the case), then the payback period is four years.

### **34.6.3 Information Economics**

Information Economics (Parker *et al.*, 1988) attempts to include the evaluation of intangible as well as tangible benefits into the process of IS evaluation. It does this by assessing feasibility in the business domain and viability in the technological domain. Information Economics uses an extended form of ROI:

traditional cost/benefit analysis	+ value linking	+ value accelerating	+ value restructuring	innovation valuation
---	--------------------	-------------------------	--------------------------	-------------------------

Value linking and accelerating are attempts to estimate the ripple-effect of technology change on the organisation. Value restructuring is an attempt to assess the increases in productivity that arise from the introduction of IS/IT systems.

At a practical level Information Economics involves completing a scorecard for each IS and computing the weighted score as an indication of the value of the system to the organisation.

Evaluator	Technology Domain										
	Business Domain					Technology Domain					
	RoI (+)	SM (+)	CA (+)	MI (+)	CR (+)	OR (-)	SA (+)	DU (-)	TU (-)	IR (-)	
Business Domain											
Technology Domain											Weighted Score
Weighted Value											

Each of the columns on the scorecard stands for the following:

1. RoI = return on investment
2. SM = strategic match, i.e. the degree to which the system matches the strategy of the organisation
3. CA = competitive advantage, i.e. the degree to which the system will afford competitive advantage
4. MI = management information support
5. CR = competitive response, i.e. the degree to which the system will enable the organisation to react quickly to its environment
6. OR = organisational risk
7. SA = strategic IS architecture, i.e. the degree to which it matches the architecture for IS in the organisation
8. DU = definitional uncertainty, i.e. the degree to which requirements for the system remain uncertain
9. TU = technical uncertainty, i.e. the number of technical imponderables in the project

10. IR = IS infrastructure risk, i.e. the degree to which the system can adversely affect the IS infrastructure

Note that some of the columns are labelled as '+', in which they are taken as positively contributing to value and hence should be added to the score. Other columns are labelled '−', in which case they are seen as negatively contributing to value and hence should be subtracted from the score. Note also that each of the factors on the scorecard can be weighted. This allows the evaluator to indicate the importance of each factor to the particular project under consideration. Hence, in one project, strategic match may be a critically important factor and hence weighted as 10. In terms of another project, perhaps to produce a more operationally based or support system this factor may be rated as of low importance and perhaps weighted as 4.

### 34.7 Conclusion

The general conclusion is that IS managers and developers need to get more involved in IS investment work of the form discussed in this chapter because of increasing perception that the IS service is not delivering expected value to organisations. Investment in IS involves 3 components: costs, benefits, balance between costs and benefits.

Costs may be tangible (visible) or intangible (hidden). However, IS costs are compounded over time. Organisations can rarely choose to withdraw from an initial IS investment. Perhaps the term cost is inappropriate in the modern context of IS/IT. Rather than portraying IS/IT as an inherent cost, IS/IT should be seen as an investment. This may lead to a change of emphasis from one of cost savings in IS/IT to one of asset management.

IS benefits can be classified from tangible (cost savings) to intangible (greater effectiveness). Traditional techniques for investment appraisal such as payback period assume tangible benefits. The latest techniques such as Information Economics attempt to include intangible benefits into the evaluation process.

### 34.8 References

- Keen P.G.W. (1991). *Shaping the future: business design through IT*. Harvard Business School Press.
- Parker M., Benson R., Trainor H. (1988). *Information Economics*. Prentice Hall, London.

**34.9 Exercises**

1. Choose an information system known to you. Attempt to estimate the benefits associated with the IS and the costs associated with its development.
2. Choose an intended information system. Try to assess its utility using one of the standard techniques such as ROI.
3. In terms of some known information system, apply the information economics scorecard to the project.

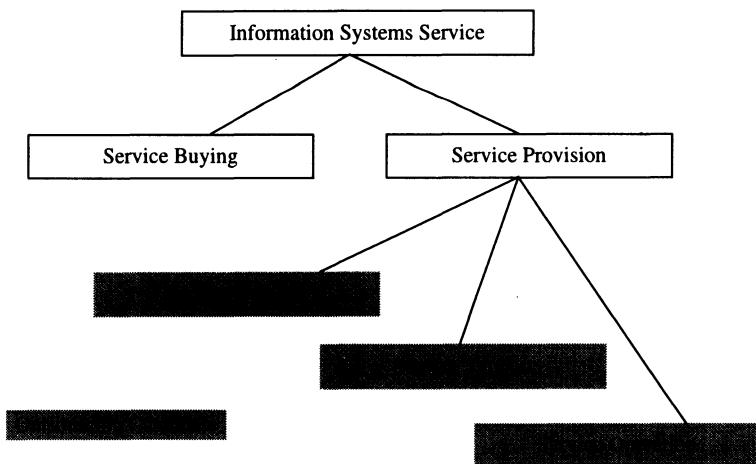
# 35 Information Systems Outsourcing

## 35.1 What is Outsourcing?

Outsourcing might be defined as the use of external agents to perform one or more organisational activities. In the last decade or so there has been a trend, particularly among large-scale companies, to hand over either the whole or part of the IS function to external agents.

One should emphasise that outsourcing is an issue that is not specific to IS. Many other organisational functions, such as building maintenance, car fleet management, etc., have been outsourced for many years in organisations. In general terms we may distinguish between a number of distinct types of IS outsourcing:

1. *Body shop* – contract programmers.
2. *Project* – use of vendors to develop a new system.
3. *Support* – use of vendors to maintain and support a system.
4. *Hardware* – outsourcing of hardware operations, disaster recovery, network management.
5. *Total IS* – ‘keys to the kingdom’ outsourcing: develop, operate, manage and control IS operations.



35.1: Types of Outsourcing

The central focus of outsourcing is the contract negotiated with an external supplier. This contract is normally executed as a service-level agreement – a

fixed fee for a specified number of services for a given contract duration. An excess fee is usually negotiated for additional services.

Figure 35.1 indicates the types of services that might be outsourced. ‘Keys to the kingdom’ implies outsourcing the entire IS service. However, most companies would wish to keep some element of service buying in-house and perhaps outsource one or more of the functions associated with information service provision.

### **35.2 Claimed Benefits of Outsourcing the IS Service**

There are a number of claimed benefits for outsourcing either the whole or a part of the IS services function:

1. *Economy*. Scale and specialisation enables vendors to deliver the same value for less money than insourcing (20-40% reduction in costs is frequently claimed).
2. *Quality*. More effective control of vendors leads to a better quality of service.
3. *Predictability*. Fixed-price contracts and service-level guarantees eliminates uncertainty.
4. *Flexibility*. Business growth can be accommodated without quantum changes in infrastructure.
5. *Making fixed costs variable*. Some agreements, such as running payroll may be based on price per unit work done.
6. *Freeing up human capital*. Scarce and costly IS talent can be refocused on higher value activity.
7. *Freeing up financial capital*. Some agreements include the sale for cash of technology assets to the vendor.

Clermont (1991) argues that for companies to derive value from outsourcing they need to do the following things well:

1. Segment the range of IT activities into pieces that potentially can be outsourced.
2. Use sound business analysis to identify those segments it makes most sense to outsource.
3. Treat the outsourcing relationship as a partnership, using the procurement process as a test of the vendor/user working relationship.

Clermont (1991) identifies the following ways in which outsourcing makes sense:

1. When there is little opportunity for a company to distinguish itself competitively through its use of IT either in terms of its processing or its applications.

2. When the predictability of an uninterrupted IS service is not required.
3. When it does not strip the company of critical technical know-how that is the key to technical innovation.
4. When existing IS/IT capabilities are limited or ineffective.

### 35.3 Myths of Outsourcing

Lacity and Hirschheim conducted a major research of outsourcing strategies in the US. They argue that much of the discussion of outsourcing in the literature has mythical qualities. In particular, they review and criticise three dominant assumptions in this literature:

1. Organisations initiate outsourcing for reasons of efficiency.
2. An outsourcing vendor is inherently more efficient than an internal IS dept through economies of scale.
3. Vendors are partners.

#### 35.3.1 Reasons for Outsourcing

There appears to be a distinction between the overt and covert reasons for organisations outsourcing their IS service. So far as overt reasons are concerned, the main rationale seems to be that of cost efficiency and eliminating a burdensome resource since IS is seen to be non-core competency for the organisation. Research suggests that many organisations consider outsourcing for a number of covert reasons such as to acquire or justify additional resources, to react to positive media reports of outsourcing, to reduce personal risk (management) associated with uncertainty, to enhance personal credibility of IS managers.

#### 35.3.2 Economies of Scale

The assumption that outsourcing generates inherent economies of scale can be broken up into two parts: assumptions associated with efficiencies generated by mass production (such as lower execution, hardware and software costs) and assumptions associated with labour specialisation (such as greater access to technical and business talent). Each of these assumptions can be criticised.

1. *Cost per million of instructions per second (MIP)*. There appears to be no correlation between size of data centre and efficiency.
2. *Hardware costs*. Smaller companies may be able to negotiate good discounts through use of older hardware or maintaining more difficult configurations of hardware and software.

3. *Software costs.* Change from company to group (size of hardware) licenses may benefit smaller companies.
4. *Access to technical talent.* Many outsourcing companies employ existing IS staff.
5. *Access to business talent.* Talented employees frequently get siphoned off to woo new accounts.

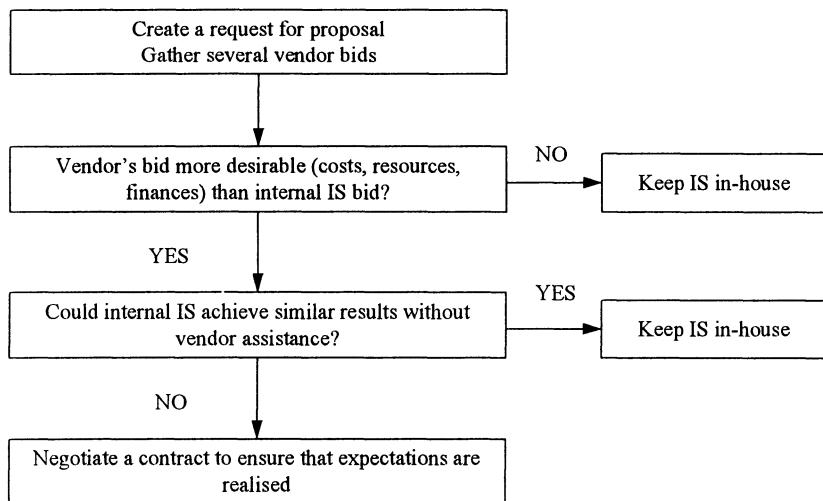
### **35.4 Lessons of Outsourcing**

In their study of outsourcing Lacity and Hirschheim report on a number of lessons of outsourcing:

1. Public information sources tend to portray an overly optimistic view of outsourcing.
2. Outsourcing appears to be a symptom of the problem of demonstrating the value of IS to organisations.
3. Organisational members may initiate outsourcing for reasons other than cost efficiency.
4. An outsourcing vendor may not necessarily be more efficient than an internal IS department.
5. The internal IS department may be able to achieve similar results without vendor assistance.
6. If a company decides to outsource, then the contract is the only mechanism to ensure that expectations are realised.
7. The metaphor that IS is merely a utility is misguided. Based on the assumption that one unit of IS resource is equal to any other.

### **35.5 An Outsourcing Selection Process**

Given the lessons above, Lacity and Hirschheim suggest that outsourcing of the IS service be approached with more caution than perhaps it has been in the past. They suggest a selection process be constructed using that illustrated in figure 35.2 as a basis.



**Figure 35.2: Outsourcing Evaluation Process**

### 35.6 Conclusion

Outsourcing is the use of external agents to perform one or more organisational activities. In this chapter we have considered some of the issues surrounding the issue of outsourcing either the whole or part of the IS service in an organisation. The large amount of attention devoted to outsourcing by organisations since the mid-1980s appears to be a reflection of a certain dissatisfaction on the part of businesses with the levels of service they are getting from their IS/IT arm. However, the assumption that information and information systems are analogous to physical materials and hence to utilities is misguided. Information and information systems are necessarily intertwined with human activity. It therefore becomes more difficult to separate out IS from the organisation than it is to separate out other services.

### 35.7 References

- Clermont P. (1991). 'Outsourcing without guilt'. *Computerworld*. Sept 9.  
 Lacity M.C. and Hirschheim R. (1993). *Information Systems Outsourcing: myths, metaphors and realities*. John Wiley, Chichester.

**35.8 Exercises**

1. The IS outsourcing market is currently dominated by three or four major IS/IT suppliers. What effect do you think this level of domination may have on the IS/IT industry if the outsourcing trend continues?
2. Many software development houses have subcontracted functions such as programming to third world development shops. What effect do you think this will have on the Western software industry over the long term?

# **36 Quality Assurance**

## **36.1 Introduction**

Quality assurance or software quality assurance has been defined as a planned and systematic pattern of all the actions necessary to provide adequate confidence that software conforms to established technical requirements (Chow, 1985). The key ideas of quality assurance are:

1. *Comprehensiveness.* Quality assurance is not restricted to the function of a software quality group or phase. It includes all the necessary activities that contribute to the quality of software throughout the entire life cycle of a project.
2. *Planning.* The emphasis is on a systematic plan to achieve the objectives of software quality. The quality of a piece of software is not left to the efforts of individuals.
3. *Relativity.* The notion of quality is relative to some requirements. There is no absolute sense of quality. The purpose of quality assurance is not to guarantee 100% reliability or zero defect software. It is rather to increase confidence that every reasonable effort has been made to ensure the quality of the end product. Quality therefore equals conformance to requirements, not excellence.
4. *Cost.* When purchasing any product, the level of quality of a product is usually reflected in its price. Hence, software quality involves increased cost. Product quality is therefore wholly a matter of customer choice. It is essential at the requirements analysis stage for the analyst to identify appropriate customer quality needs.

## **36.2 Information Systems Areas**

Quality assurance is therefore a topic that involves many different information systems areas. First, although a definition is provided above, there is no ready agreement on a definition of software quality, not to mention how to measure it. Second, since quality must be built into the product from day one of a project, quality assurance must include a consideration of sound techniques of construction. Third, in most large-scale projects, quality cannot be assured without effective management. Fourth, the availability of tools for information systems development and project management is critical to the success of quality assurance. Fifth, quality is a concept that has organisation-wide implications. Quality assurance has to be planned for and implemented within the environment of a given organisation.

In this chapter we limit ourselves to considering how quality can be effected in two ways: by the use of software metrics; through the adoption of peer group review.

### 36.3 Software Metrics

A software metric is a number extracted from a software product. Metrics are used in a number of ways: to provide feedback to staff on the quality of their work; to monitor the structural degradation of a system during maintenance; to aid costing and estimating software projects.

Some of the main classes of software metrics are given below:

1. *Function-based metrics.* These are metrics used primarily for project estimation. Probably the best-known function-based metric is associated with an estimating technique known as function-point analysis. Project managers conduct function-point analysis by counting the features of some functional specification such as the number of entities or processes. These counts are then inserted into an algebraic expression that produces what is called a function point count. Function point counts and actual costs of projects are stored in a historical database. These figures are used to produce a statistical estimate for the current project.
2. *System design metrics.* These metrics are extracted from design notations such as structure charts (chapter 21). They are usually calculated by counting features such as the size of a module's interface or the number of calls that a module makes. These metrics are used as a measure of the amount of coupling between the modules of a system (chapter 21). Good designs exhibit a low measure of coupling.
3. *Control-flow metrics.* These metrics are usually extracted from detailed program designs or program code. They are based on counting features in the control flow of a program or module, such as the number of decision statements. Control-flow metrics are particularly useful for measuring the readability and testability of modules.
4. *Source-code metrics.* These metrics are based on counting features in source code such as the number of identifiers and operands. They have been used to measure the readability of programs, as an estimate of the number of residual errors, and as an estimate of the amount of debugging that a module requires.

Metrics do not need to be as sophisticated as some of the ones indicated above. In one company known to the author, the most useful metrics collected were the number of errors identified in events known as structured walk-throughs.

### 36.3.1 Structured Walkthroughs

A structured walkthrough is a group review of any product of software development prior to its release into the project life cycle. It is also referred to as a peer group review, team debugging or ego-less programming.

Walkthroughs can take place at various stages in a project. During analysis, such items as E-R diagrams and DFDs might be subject to review. During the design phase a review of structure charts might be considered. At the implementation phase, walkthroughs of programs and the associated documentation are essential.

Yourdon (1978) has maintained that structured walkthroughs have a number of tangible and intangible benefits:

1. *Correctness.* Organisations that subject all systems to stringent walkthroughs reportedly reduce the program error-rate from an average of three to five defects per hundred lines of code to a more manageable average of three to five defects per thousand lines of code.
2. *Standards.* Walkthroughs are a useful way of establishing and enforcing standards for the analysis, design, coding, testing and documentation of computer systems.
3. *Readability.* Walkthroughs improve the readability of programs and systems documentation, thereby aiding the process of software maintenance.
4. *Training.* Walkthroughs have the effect of passing on good organisational practices to new company personnel.
5. *Insurance.* Walkthroughs disseminate information about systems throughout the organisation, thus offsetting the problems of high staff turnover.

A structured walkthrough is usually a group effort, with several people serving different roles within the team. A team might consist of a presenter, a co-ordinator, a scribe, a maintenance critic, a standards critic, and one or more user representatives.

1. The presenter is normally the creator of the product being reviewed. His or her role is to lead the team through an examination of the product.
2. The co-ordinator organises all the activities that should occur prior to a walkthrough. For example, the co-ordinator must ensure that each team member receives a copy of the product to be reviewed prior to the walkthrough.
3. The scribe records the proceedings, and eventually forwards a report to management summarising the team's findings.
4. The maintenance critic inspects the product for any source of future maintenance problems.
5. The standards critic ascertains that the product adheres to organisational standards.

## 6. User representatives verify that the product performs as requested.

Even though each team member has a primary role, all team members are equally accountable in terms of examining the quality of the system and offering comments and criticisms. The responsibility of the team is to give an accurate appraisal of the product being reviewed. The participants should identify defects, but they should not attempt to correct them. Correction is the sole responsibility of the presenter.

During the walkthrough, participants are required to remain as objective as possible. In particular, they should remember to evaluate only the product itself, and not the person presenting it. A walkthrough can easily degenerate into heated and destructive argument if criticism becomes personal. Moreover, the participants should remember that in system development there are many 'correct' solutions to the same problem. Participants should therefore not try to impose their own preferred solution on the problem under consideration.

The outcome of the walkthrough is the walkthrough report prepared by the scribe. This report comes in two parts: a summary and an issues list.

On the summary, all participants sign the report to indicate that they are in agreement with the decisions made. At the bottom of the summary the final verdict is given. The product is accepted as is with minor revisions or it is rejected. Rejection can be broken down into three categories:

1. The product has so many serious flaws that it must be completely rebuilt.
2. The product needs major revisions.
3. The review was incomplete and must continue.

The second page of the report, the issues list, details all the problems that need attention. As the problems are corrected, they are checked off this list by the team member assigned to monitor progress.

Managers are expected to receive at least a copy of the summary. In contrast, the issues list is primarily for the benefit of the presenter.

### **36.3.2 Function Points**

Probably the most popular group of metrics are the function-based metrics. These are metrics used primarily for project estimation. Probably the best-known function-based metric is associated with an estimating technique known as function-point analysis. Project managers conduct function-point analysis by counting the features of some functional specification such as the number of entities or processes in a system. These counts are then inserted into an algebraic expression that produces what is called a function point count. The function point count of a system is used as a measure of the complexity of a system. Function point counts are normally related against three other forms of project-based metric: actual costs of development projects; number of person days

required to complete a project; total elapsed time for a project. All of these metrics are ideally stored in a historical database. These figures are then used to produce a series of estimates for the current project. In very general terms one would expect systems with high function points to take longer and involve more staff than those with low function points.

Albrecht and Gaffney (1983) originally proposed function point analysis. In this approach, each component part of the system is classified in terms of five function point types:

1. External input types such as transactions from the user or other applications.
2. External output types such as transactions to the user or to other applications.
3. Logical internal file types, i.e. entities within the systems data model.
4. External file types, i.e. entities (files) to which the system will interface.
5. Query types such as those from users or the application.

Each component then has to be, classified as of low, average or high complexity. The estimator then has to calculate an aggregate complexity factor for each function type by multiplying the number of low complexity types, the number of average complexity types and the number of high complexity types against a weighted factor (in the range 1-10). The sum these results is then calculated to provide the total number of unadjusted function points for the system. A suggestive table of weightings is given in table 36.1.

	<b>Table 36.1</b>			
<b>Description</b>	<b>Function Complexity</b>			<b>Total</b>
	<b>Low</b>	<b>Medium</b>	<b>High</b>	
External Input	$\dots \times 3 =$	$\dots \times 4 =$	$\dots \times 6 =$	
External Output	$\dots \times 4 =$	$\dots \times 5 =$	$\dots \times 7 =$	
Internal File	$\dots \times 7 =$	$\dots \times 10 =$	$\dots \times 15 =$	
External File	$\dots \times 5 =$	$\dots \times 7 =$	$\dots \times 10 =$	
Query	$\dots \times 3 =$	$\dots \times 4 =$	$\dots \times 6 =$	
<b>Total Unadjusted Function Points</b>				

The next step is to apply an adjustment factor for particular application characteristics. For instance, the estimator might calculate the weight of influence (in terms of 0 for the factor not being present through to 5 for strong influence throughout the development) of the following factors: data communication; distributed functions; performance; heavily used configuration; transaction rate; on-line data entry; end-user efficiency; on-line update capability; complex

processing; reusability; ease of installation; ease of operation; multiple sites; facility for change. These influence factors are then summed and an overall adjustment figure produced using an expression such as, adjustment =  $0.65 + (0.01 \times \text{sum of influence factors})$ . This means that the sum of the influence factors can range from 0-35, meaning that the adjustment factor ranges between -35% to +35%. The last step is to multiply the function points total by the adjustment figure.

***Example***

Assume that we have analysed the Goronwy Galvanising application in terms of the following makeup of function points:

Two simple and one complex input transactions	$(2 \times 3) + 6$	= 12
Three average complexity external outputs	$3 \times 5$	= 15
Five simple tables	$5 \times 7$	= 35
One simple interface to another computer	$1 \times 5$	= 5
One simple and one complex query	$(1 \times 3) + (1 \times 6)$	= 9
Total unadjusted function points for system		76

The technical complexity adjustment included within the process described above has been the subject of a number of criticisms. In particular, it has been criticised for seemingly reflecting the concerns of computer systems constructed in the 1970s and early 1980s. The shape of modern-day computer systems is seen to be subtly different.

Charles Symons (1988, 1991) has proposed a number of improvements to the function point analysis of Albrecht and Gaffney. This form of function point analysis has become known as mark 2 function point analysis. Symons maintains that rather than categorise each transaction or process and then draw the relevant function points in terms of referencing a table, the estimator need only address three function types and multiply these by a defined constant for each process in a system.

$$\begin{aligned}
 & 0.58 \times (\text{number of input data element types}) + \\
 & 1.66 \times (\text{number of entity types referenced}) + \\
 & 0.26 \times (\text{number of output data elements})
 \end{aligned}$$

These three constants can be varied depending on whether the project team feels they are putting in more effort in to one element than another. However, the sum total of the three constants should always add up to 2.5.

***Example***

Assume that one of the processes within the Goronwy Galvanising application has the following profile.

The user keys in four data elements	$0.58 \times 4$	= 2.32
The process accesses two tables (entities)	$1.66 \times 2$	= 3.32
The process signals acceptance or rejection	$0.26 \times 2$	= 0.52
Unadjusted function points of this process		6.16

Estimators will need to calculate the function points for each process then sum them to obtain the overall unadjusted function points for the system. As in Albrecht's version, this unadjusted figure is then modified by a technical complexity adjustment. Symons proposes a larger set of factors to consider than Albrecht and also encourages the addition of factors to suit local conditions.

### 36.4 Quality Assurance and the Organisation

The quality ethic has become influential in business circles ever since the Japanese demonstrated the marketability of the concept. In organisations whose primary business is the production of information systems the question of quality is influenced by a number of issues:

1. *Project management.* It is argued that an effective project manager will build into his monitoring process clear checkpoints for assessing the quality of a developing information system (chapter 28).
2. *CAISE.* Many argue that CAISE tools offer the developer a route to better quality systems because fewer errors are likely to occur between stages such as design and implementation (chapter 12).
3. *The application of methods.* The advocates of development methods would claim that the application of such approaches is a key to quality in the sense that they particularly improve the communication process and co-ordination between large teams of developers (chapter 27).
4. *Object-Orientation.* Object-oriented languages foster the reuse of code. Building an information system from well-proven parts is likely to lead to increased quality (chapter 9).

### 36.5 Conclusion

Quality assurance is a series of activities, standards and procedures that ensure that software is developed that meets user requirements. The concept of quality and the process of assuring quality are many-faceted beasts that affect all aspects

of information systems development, from systems analysis and design through programming to IS management and planning. In a sense, the issue of quality is central to the professionalisation of information systems engineering which we discuss in chapter 39. Some notion of the distinction between a good and bad system is intrinsic to the rationale behind engineering disciplines.

One of the main lessons being learnt is that quality cannot be grafted on to a system at the end of its development. Quality is, above all, about people. A continuing commitment to produce quality information systems and provide a quality service is needed from people at all levels in information systems engineering.

### 36.6 References

- Albrecht A. J. and Gaffney J.E. (1983). 'Software Function, Source Lines of Code and Development Effort Prediction: a software science validation'. *IEEE Transactions on Software Engineering*. 9(6). 639-647.
- Chow T.S. (Ed.). (1985). *Software Quality Assurance: A Practical Approach*. IEEE Computer Society Press, Silver Spring, MD.
- Ince D. (Ed.). *Software Quality and Reliability: Tools and Methods*. Chapman and Hall, London. 1991.
- Symons C.R. (1988). 'Function Point Analysis: difficulties and Improvements'. *IEEE Transactions on Software Engineering*. 14(1). 2-11.
- Symons C.R. (1991). *Software Sizing and Estimating – Mark II FPA*. John Wiley.
- Yourdon E. (1988). *Structured Walkthroughs*. Yourdon Press, New York.

### 36.7 Exercises

1. One man's idea of quality is another man's idea of shoddy workmanship. Discuss.
2. Discuss whether a metric of quality is feasible?
3. Discuss whether a metric of quality is desirable?
4. In what ways do you think structured walkthroughs engender a quality ethic in project teams?
5. What disadvantages might be associated with structured walkthroughs?
6. How do you think standards affect quality and quality assurance?
7. Discuss the need for a quality control department in large information systems development organisations.
8. Discuss how the issue of quality might affect the Goronwy Galvanising project.

## ***Part 6***

# ***The Discipline of Information Systems Development***

To conclude this work we look at three inter-dependent issues that have an important bearing on the future of information systems development, both as a theoretical discipline and as a practical, professional activity.

Despite a vast array of tools, techniques, methods and management approaches available to workers in IS, many information systems continue to ‘fail’. In chapter 37 we consider this important issue and emphasise the important need to consider the interdependencies between information systems and organisations.

This issue is taken up again in chapter 38. Here we discuss the numerous factors which affect the ‘fit’ between information systems and organisations. This is the central focus of the discipline of information systems and therefore we use this chapter to help summarise much of the material covered within the body of the text.

In chapter 39 we consider the way in which bodies have attempted to define information systems as a true professional activity. We consider the need for such professionalisation as well as some of the practical constraints facing bodies wishing to promote this process.

Finally, in chapter 40 we return to the issue of what constitutes information systems in general and information systems development in particular as a discipline. We consider what is distinctive about information systems as compared to software engineering, computer science and management science. This leads us to focus more closely on information systems development as an important area of study within information systems and conclude with some of the ways in which ISD should be subject to more empirical investigation.

# **37 Information Systems Failure**

## **37.1 Introduction**

Information Systems (IS) failure is a topic which has become extremely newsworthy both in the general and the computing press in recent years. As a consequence, there is some evidence of an increasing concern amongst organisations in the UK and elsewhere with the large amounts of money which appears to have been devoted to software projects with little apparent organisational benefit. Available literature suggests that failure is a ubiquitous feature of both the UK and International experience of IS engineering (Coopers and Lybrand, 1996).

Given its prominence it is perhaps not surprising to find that the topic of IS failure has been a fervent area of debate for academics in the information systems, software engineering and computer science areas for a number of years. A considerable amount of published theoretical work has helped academics and practitioners achieve a better understanding of IS failure, particularly by demonstrating the multi-faceted nature of the concept. However, what empirical data we have on this topic is limited to either anecdotal evidence, case studies of different orders and indeed quality, and a limited amount of survey work (Beynon-Davies, 1997).

## **37.2 Reasons for Studying IS Failure**

There are a number of reasons why IS failure is a topic worthy of more detailed study:

1. *The ubiquity of failure.* A survey conducted by the US Government's Accounting Agency in 1979 (US, 1979) found that less than 3% of the software that the US government had paid for was actually used as delivered. More than half of the software was never used at all. Gladden (1983) reports in a similar survey that 75% of all system development undertaken is either never completed or the resulting systems are not used. In a recent international survey conducted by Coopers and Lybrand (1996), 60% of organisations internationally and 67% of organisations in the UK had suffered at least one systems project that had failed to deliver planned business benefits or had experienced significant cost and time overruns.
2. *Validation of IS engineering practice.* IS failures are also significant because they act as an important resource for validating development practice. The information technology industry is very good at suggesting 'appropriate' tools, techniques, methods and frameworks for constructing and managing

- information systems (Harel, 1980). It is generally poor at supplying empirical validation for many of its proposals (Galliers, 1992).
3. *The importance of breakdowns.* In other disciplines, the detailed study of 'breakdowns' has proven significant in changing established practice. Boddie (1987) cogently captures this idea in the following quote: 'We talk about software engineering but reject one of the most basic engineering practices: identifying and learning from our mistakes. Errors made while building one system appear in the next one. What we need to remember is the attention given to failures in more established branches of engineering.' In structural engineering, for instance, the study of bridge failures has revealed important information leading to the redesign of bridges, and the study of aircraft disasters has contributed greatly to the redesign of aircraft (Petrowski, 1985).
  4. *Evaluation and risk.* IS failure can be considered as an important part of the related areas of IS evaluation and risk assessment/risk management. In other words, whatever can be said about understanding IS 'failure' can also be used for evaluating IS 'success' (Lyytinen and Hirschheim, 1987). The concept of failure is also inherently associated with the concept of risk. Risk might be defined as a negative outcome that has a known or estimated probability of occurring based on some experience or theory. The idea of IS failure is clearly the negative outcome most prominent in most developers', managers' and indeed users' minds (Wilcocks and Griffiths, 1994).
  5. *Professionalism.* IS failure is important as a topic of study because of the crucial bearing it has on the issue of professionalism in IS engineering. IS failure, or the association with such 'failure', seems to come close to a form of 'social leprosy' or stigma (Goffman, 1990). Developers seem to become tarnished by association with information systems projects that are deemed to have failed. Therefore Abdel-Hamid and Madnick (1990) not surprisingly make the point that, although it is important to learn from IS project failures, people tend to hide mistakes rather than report and evaluate them. Also, Ewusi-Mensah and Przasnyski (1995) found amongst a sample of US companies that most organisations do not keep records of their failed IS projects, and do not make any formal attempt to understand what went wrong or attempt to learn from their failed projects. There is also some evidence to suggest that prominent IS failures such as LASCAD, TAURUS and WESSEX (Beynon-Davies, 1995) have the potential to affect the public perception of IS and IS engineering and consequently the professional relationship between IS engineers and their clients.

### 37.3 Existing Work on IS Failure

It is useful to distinguish between the extant theoretical and empirical work on IS failure. Most of the theoretical work has tended to concentrate on defining the concept of IS failure and in extending its connotations to include social and

organisational concerns. The current empirical work can be considered under three headings: collections of anecdotal material, case studies and survey work.

### **37.3.1 Theoretical Work**

Since the 1970s, a number of frameworks have been proposed for understanding the concept of IS failure, for example: Lucas (1975), Bignell and Fortune (1984), Lyytinen and Hirschheim (1987), Sauer (1993). Two recent approaches to IS failure seem particularly important because of the way in which they relate IS failure to social and organisational context: Lyytinen and Hirschheim's concept of expectation failure and Sauer's concept of termination failure.

Lyytinen and Hirschheim (1987) in conducting a survey of the literature on IS failure identify four major theoretical categories of such phenomena:

1. *Correspondence failure*. This is the most common form of IS failure discussed in the literature and typically reflects a management perspective on failure. It is based on the idea that design objectives are first specified in detail. An evaluation is conducted of the information system in terms of these objectives. If there is a lack of correspondence between objectives and evaluation the IS is regarded as a failure.
2. *Process failure*. This type of failure is characterised by unsatisfactory development performance. It usually refers to one of two types of failure. First, when the IS development process cannot produce a workable system. Second, the development process produces an IS but the project runs over budget in terms of cost, time, etc.
3. *Interaction failure*. Here, the emphasis shifts from a mismatch of requirements and system or poor development performance to a consideration of usage of a system. The argument is that if a system is heavily used it constitutes a success; if it is hardly ever used, or there are major problems involved in using a system then it constitutes a failure. Lucas (1975) clearly adheres to this idea of failure.
4. *Expectation failure*. Lyytinen and Hirschheim describe this as a superset of the three other types of failure. They also describe their idea of expectation failure to be a more encompassing, politically and pluralistically informed view of IS failure than the other forms. This is because they characterise correspondence, process and interaction failure as having one major theme in common: the three notions of failure portray a highly rational image of IS development; each views an IS as mainly a neutral technical artefact (Klein and Hirschheim, 1987). In contrast, they define expectation failure as the inability of an IS to meet a specific stakeholder group's expectations. IS failures signify a gap between some existing situation and a desired situation for members of a particular stakeholder group. Stakeholders are any group of people who share a pool of values that define what the desirable features of an IS are, and how they should be obtained.

Lyytinen (1988) broadens this analysis by making the useful distinction between development failure and use failure. Stakeholder groups can face problems in IS either in development terms or in terms of use. In the former case, a stakeholder's main concern is to mould the future IS to fit its interests. In the latter case, the main concern is to align the IS with the stakeholder's going concerns. In terms of development failure Lyytinen lists a number of categories of common problems: goals, technology, economy, organisational impact, participation and control of development, perception of development. In terms of use failures the following categories are held to be important: technical solutions, data problems, conceptual, people's reactions, complexity problems.

Sauer (1993) has recently criticised the model proposed by Lyytinen and Hirschheim for its plurality. Sauer's model posits a more conservative definition of information systems failure. According to his account an information system should only be deemed a failure when development or operation ceases, leaving supporters dissatisfied with the extent to which the system has served their interests. This means that a system should not be considered a failure until all interest in progressing an IS project has ceased. This definition of *termination failure* is hence stricter than Lyytinen and Hirschheim's concept of *expectation failure*.

Sauer develops a model of IS failure based on exchange relations. He portrays the development of information systems as an innovation process based on three components: the project organisation, the information system, and its supporters. Each of these components is arranged in a triangle of dependencies. The information system depends on the project organisation, the project organisation depends on its supporters, and the supporters depend on the information system. The information system requires the efforts and expertise of the project organisation to sustain it; the project organisation is heavily dependent on the provision of support in the form of material resources and help in coping with contingencies; supporters require benefits from the information system.

One of the key ways in which Sauer distinguishes termination failure from expectation failure is in terms of the concept of a flaw. Information systems are the product of a process which is open to flaws. Every information system is flawed in some way. However, flaws are different from failures. Flaws may be corrected within any innovation process at a cost, or accepted at a cost. Flaws describe the perception of stakeholders that they face undesired situations which constitute problems to be solved. Examples of flaws are program bugs, hardware performance, organisational changes, etc. Unless there is support available to deal with flaws they will have the effect of reducing the capacity of some information system to serve its supporters and may result in introducing further flaws into the innovation process. At some stage, the volume of flaws may trigger a decision to remove support and hence to terminate a project.

### ***37.3.2 Empirical Studies***

For convenience it is useful to divide empirical investigations of IS failure into three categories: anecdotal evidence, case studies, survey research.

#### *Anecdotal Evidence*

For a number of years a collection of anecdotal descriptions of IS failures has been accumulating in the ACM's Software Engineering Notes. McKenzie (1994) has analysed this material and found that of computer-related accidents (examples mainly of use failures) reported, 92% involved failures in what McKenzie calls Human-Computer Interaction: 'More computer-related accidental deaths seem to be caused by interactions of technical and cognitive/organisational factors than by technical factors alone.'

#### *Case Studies*

Benbasat *et al.* (1987) describe case research as being particularly important for those types of problems where research and theory are still at their early, formative stages. They see a key use of case studies in the generation of theory from practice. The topic of IS failure has been seen to be a particularly formative research area and therefore one particularly amenable to the case study approach.

The London Ambulance Computer Aided Despatch System project has been one of the most frequently quoted UK examples of information systems failure in recent times. Beynon-Davies (1995) makes the case that the prominence of this particular example is probably due more to the 'safety-critical' nature of this system and the claim that 20-30 people may have lost their lives as a result of this failure, than the scale of the project. Indeed, LASCAD (£1.1-£1.5m) is dwarfed by other British IS 'failures' such as Wessex regional health authorities RISP project (£63m) (HMSO, 1993), and the UK stock exchange's TAURUS settlement system (£75-£300m) (Waters, 1993). Flowers (1996) is a text which collates material on each of these projects as well as presenting a number of other case studies of failure.

IS failure is of course not specifically a British malaise. For instance, Oz (1994) takes a similar documentary approach to describing the important case of the CONFIRM reservation system in the US (\$125m). Also, Sauer (1993) comprehensively describes a large Australian government IS project – Mandata (A\$30m) – that was abandoned during the 1970s.

#### *Survey Research*

Lucas (1975) describes a series of quantitative investigations used to verify a number of general hypotheses on IS failure. Unfortunately, the data collected does little to illuminate the complex reality of organisational IS and indeed the factors contributing to IS failure.

Lyytinen (1988) describes an exploratory study of the expectation failure

concept by looking at systems analyst's perceptions of information systems failure. Systems analysts are discussed in terms of being a major stakeholder group in systems development projects. Their view of IS and IS failures differs from users or management views. Interestingly, Lyytinen found that systems analysts believed that only 20% of projects are likely to turn out to be failures. They also preferred to explain failures in highly procedural and rationalistic terms. Reasons mentioned for failure include inexact development goals and specifications, inadequate understanding of user's work and inadequate understanding of system contingencies. This, Lyytinen concludes is concomitant with a set of professional expectations of the IS development process which conceives of it as an overtly rational act involving high technical and professional competence.

Ewusi-Mensah and Przasnyski (1994) distinguish between what they call IS project abandonment and IS failure. IS failure in their terms deals with the failure of usage and/or operations of the IS, whereas IS project abandonment is concerned with the process of IS development. This is similar to Lytinnen's distinction between development and use failure. In Ewusi-Mensah and Przasnyski (1991) they distinguish between:

1. *Total abandonment*. Complete termination of all activities on a project prior to full implementation
2. *Substantial abandonment*. Major truncation or simplification of the project to make it radically different from the original specification prior to full implementation
3. *Partial abandonment*. Reduction of the original scope of the project without entailing significant changes to the IS's original specification, prior to full implementation

They suggest that from their small survey study, total abandonment is the most common type of development failure experienced in the US. They also found that organisational factors, particularly the amount of senior management involvement and the degree of end-user participation in the project development were the most widespread and dominant factors contributing to IS success/failure.

### 37.4 Levels of Analysis

The discussion above should clearly have indicated to the reader that the topic of information systems failure can be addressed on a number of levels:

1. *Technical failure*. Failure of hardware, software and communications.
2. *Project failure*. Failures in project management and control such as cost or time overruns.

3. *Organisational failure.* Failures of a system to deliver organisational benefits such as increases in efficiency or effectiveness.
4. *Environmental failure.* Failure caused by changes in environmental factors such as changes in regulation, labour relations, etc.

Clearly, solutions or avoidance strategies to the problem of failure tend to become more tractable the lower level one is in this hierarchy:

1. At the level of information technology systems the appropriate selection and use of tools can significantly reduce the risk of failure. Appropriate use of analysis and design techniques will also have a critical bearing on reducing risk.
2. At the level of projects, good and effective management and control of IS personnel and their activities can significantly reduce the occurrence of failure. Participation by the business community in development projects has also been shown to increase the likelihood of IS success.
3. At the organisational level, effective planning of the IS development portfolio and the structure and activities of the IS service can affect risk of failure. Proper management of the IS services function is also crucial to the long-term health of organisations.
4. At the environmental level, effective alignment of information systems strategy with business strategy is critical to the performance of the information systems function and the IS under their development and control. Effective business analysis needs to be conducted to guide such alignment.

In summary, ideally an organisation wishing to avoid IS failure would do well to engage in all of the suggested stages in the IS life-cycle: business analysis; IS planning; IS management; project management; IS development; IS maintenance. Many years of experience has been accumulated in the development of information systems and many lessons have been learned as to how, for instance, to manage IS projects. Yet IS failures are still commonplace. This may be partly because, although frequently most medium to large-scale organisations engage in IS management, development and maintenance, few seem to take business analysis and IS planning seriously.

### **37.5 Conclusion**

Information systems failure is a commonplace phenomenon. Because of its ubiquity IS failure is an important topic of study. Evidence seems to suggest that most well-publicised failures have not been caused by simple breakdowns in functioning. Most IS failure seems to be concerned with the socio-technical interface. This is important in that it is inherent in our definition of an information system (chapter 1). Hence, even if the information system is built well

using the techniques and tools discussed in parts 2 and 3 of this text, and even if the project is well managed and controlled using a methods discussed in part 4, it may still fail for organisational and/or environmental reasons.

The tendency might be for the information systems engineer to define the area of information systems development to be limited to technical or project concerns. Organisational and environmental factors are then seen as outside the ISD domain. However, this is a dangerous strategy. It denies the importance of the way in which the success of an information systems is heavily determined by the issue of fitness for purpose (chapter 38). It also denies the importance of the role of the IS specialist within organisations and society (chapter 39). Finally, it denies the multi-disciplinary nature of the discipline of information systems (chapter 40).

An information systems project can be said to have failed if it does not meet the expectations of a particular stakeholder group. A stakeholder is a claimant, either inside or outside the organisation, who has a vested interest in an IS project. Stakeholders are different from participants, in that a stakeholder can influence the trajectory (either during development or after delivery) of an IS.

A termination failure occurs when a project is abandoned prior to delivery. Termination failures normally occur because stakeholder groups (usually managerial groups) which provide resources for the development of an IS withdraw their support for the development work.

Termination failures involve the total abandonment of an IS project prior to delivery. Projects however need not be totally abandoned to class as failures. Flaws arising during a project may cause the project to be substantially or partially abandoned.

The withdrawal of support, particularly for large development projects, may occur quite late on in the project. In Sauer's terms, a project may continue in the face of quite serious flaws. Frequently this is because some form of escalation has occurred in a project.

Use failure occurs after the delivery of an IS. Frequently use failure happens because end-users are not seen as a significant stakeholder group in an IS project. They are not involved in the development process, and certainly have little influence over the shape of a development. Such a stakeholder group may therefore choose to engage in patterns of resistance.

The trajectory of a project is defined as the historical shaping of an information system both before and after delivery. Frequently, the shape of an information system is determined by the power-play between different stakeholder groups. It should be recognised that an IS, and an IS project, is a significant power-resource in organisations.

### 37.6 References

- Abdel-Hamid T.K. and Madnick S.E. (1990). 'The Elusive Silver Lining: how we fail to learn from software development failures'. *Sloan Management Review*. Fall. 39-48.
- Bignal V. and Fortune J. (1984). *Understanding Systems Failures*. Manchester University Press, Manchester.
- Benbasat I., Goldstein D.K., Mead M. (1987). 'The Case Research Strategy in Studies of Information Systems'. *MIS Quarterly*. Sept.
- Beynon-Davies P. (1995). 'Information Systems "Failure": The case of the London Ambulance Service's Computer Aided Despatch System'. *European Journal of Information Systems*. 4. 171-184.
- Beynon-Davies P. (1997). *Analysing Information Systems Failure*. Financial Business Report, London.
- Beynon-Davies P., Mackay H., Tudhope D., Slack R. (1996). 'Prototyping in Information Systems Development: a study of development practice in natural settings'. *European Conference on Information Systems '96*.
- Boddie J. (1987). 'The Project Post-Mortem'. *Computerworld*. 7th Dec.
- Cohen S. (1980). *Folk Devils and Moral Panics: the creation of the mods and rockers*. (New Ed.). Martin Robertson, London.
- Coopers and Lybrand. (1996). 'Managing Information and Systems Risks: results of an international survey of large organisations'.
- Ewusi-Mensah K. and Przasnyski Z.H. (1994). 'Factors contributing to the abandonment of information systems development projects'. *Journal of Information Technology*. 9. 185-201.
- Ewusi-Mensah K. and Przasnyski Z.H. (1995). 'Learning from abandoned information system development projects'. *Journal of Information Technology*. 10. 3-14.
- Flowers S. (1996). *Software Failure/Management Failure: amazing stories and cautionary tales*. John Wiley, Chichester.
- Galliers R. (1992). *Information Systems Research: issues, methods and practical guidelines*. Blackwells.
- Gladden G.R. (1982). 'Stop the Life-Cycle, I Want to Get Off'. *Software Engineering Notes*. 7(2). 35-39.
- Goffman E. (1990). *Stigma: notes on the management of spoiled identity*. Penguin.
- Lyytinen K. and Hirschheim R. (1987). 'Information Systems Failures: a survey and classification of the empirical literature'. *Oxford Surveys in Information Technology*. Vol. 4. 257-309.
- Lyytinen K. (1988). 'The Expectation Failure Concept and Systems Analysts View of Information Systems Failures: results of an exploratory study'. *Information and Management*. 14. 45-55.
- Lucas H.C. (1975). *Why Information Systems Fail*. Columbia University Press, New York.

- McKenzie D. (1994). 'Computer-Related Accidental Death: an empirical exploration'. *Science and Public Policy*. 21(4). 233-248.
- Petrowski H. (1985). *To Engineer is Human: The role of failure in successful design*. Macmillan, London.
- Sauer C. (1993). *Why Information Systems Fail: A Case Study Approach*. Alfred Waller, Henley-On-Thames.
- US. (1979). US Government Accounting Office Report FGMSD-80-4. Reported in *ACM Sigsoft Software Engineering Notes*. 10(5). Oct. 1985.

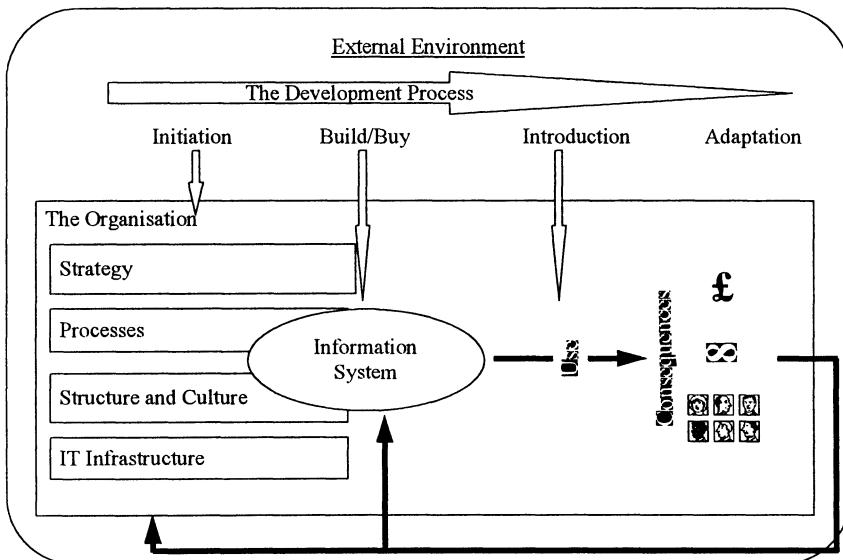
### 37.7 Exercises

1. See if you can estimate the % of IS projects in an organisation known to you that are regarded as successes.
2. What % of projects in the organisation are regarded as failures?
3. In terms of the failures what are seen as the major reasons for failure?
4. In terms of the successes what are seen to be the major features of success?
5. Analyse one of the well-documented case-studies of IS failure, such as the London Ambulance case, in terms of Sauer's framework.

# **38 Information Systems and Their Interaction with Organisations**

## **38.1 Introduction**

In this chapter we shall use what Silver, Markus and Beath (1995) have referred to as an IT interaction model as a means of discussing the key focus of information systems as a discipline and, in particular, the place of IS development activities within organisations. We prefer to call it an IS/IT interaction model because we wish to include both IS and IT in our discussion. This model is useful in discussing the place of information systems in organisations. It is referred to as an interaction model because it is founded on the premise that the consequences of IS in organisations follow from the interaction of the technology with the organisation and its environment. The effects of an information system for an organisation emerge over time as the result of the interaction of the IS with its organisational context. Understanding the nature of this interaction is therefore central to obtaining the benefits of IT as well as avoiding the hazards that IT can hold for organisations (see figure 38.1).



**Figure 38.1: IS/IT Interaction Model**

The IS/IT interaction model can be broken down into the following elements:

1. The external environment of the organisation
2. The organisation
3. The information system
4. The IS development process

Within this chapter we consider each of these elements in turn, then summarise the interrelationships between components in terms of a model of ‘fitness for purpose’.

### **38.2 Why do Organisations Invest in Information Systems?**

Firstly, it is useful to step back from the information systems area and ask the question, why do organisations build information systems? We can answer this question in the following terms: business organisations overtly invest in information systems for one or more of the following reasons:

1. To make more money, i.e. to increase profitability.
2. To be more efficient, usually to increase productivity.
3. To be more effective. This normally equates to attempts to increase the market share of a business.

But does the introduction of an information system always have these effects? Evidence suggests that there are some inherent paradoxes/contradictions involved with the introduction of IS.

### **38.3 Productivity Paradox**

One key reason why organisations use IS is the expectation that employing such systems will raise the productivity of the workforce. However, over a number of years, this link between IS usage and productivity has been questioned. This has become known as the productivity paradox.

Moshowitz (1976), for instance, has estimated that of IT investments in the US: 20% achieve something like their intended benefits; 40% fail; 40% achieve only a marginal impact on the organisation. Of the 20 office automation projects sponsored by the DTI (DTI, 1981) in the late 1970s, only 15 were continued after the trial period, and some that continued had not achieved the planned objectives. Thurrow (1991) has argued that ‘Specific cases in which the new technologies have permitted huge increases in output or decreases in costs can be cited, but when it comes to the bottom line there is no clear evidence that these new technologies have raised productivity ... or profitability. In fact, precisely

the opposite is true. There is evidence, in the United States at least, that the investment in New Technologies has coincided with lowered overall productivity and profitability.' Strassman (1990) has argued that 'There is no relationship between expenses for computers and business profitability.' Davenport (1993) cites the case of Japanese business where there is evidence of a lower degree of IT use, but by any traditional measure of productivity, Japanese firms do quite well.

Brynjolfson considers four main explanations for the phenomenon of the productivity paradox:

1. *Mismeasurement of inputs and outputs.* A proper indicator of IT impact has yet to be formulated and analysed. Traditional measures such as the number of service transactions multiplied by their unit value, tend to ignore non-traditional sources of value such as increased quality and speed of customer service.
2. *Lags due to learning and adjustment.* The long-term lag between cost and benefit may be due to the extensive learning required on the part of the individual and the organisation, to fully exploit IT.
3. *Redistribution and dissipation of profits.* This explanation proposes that those investing in technology benefit at the expense of others in a particular industry. Hence, there is no aggregate benefit to an industrial sector such as financial services.
4. *Mismanagement of IT.* This is the basic IS argument. It proposes that companies have systematically mismanaged IT and that this explains the lack of piecemeal benefit from the introduction of IS/IT.

### **38.4 Other Effects of IS/IT**

Besides poor productivity there are numerous other negative effects that arise from the introduction of IS/IT. Some people have argued, for instance, that IS/IT has been largely used not to enhance effectiveness but to increase job loss, deskilling and a consequent loss of security amongst the workforce. IS/IT has also been particularly used as a way of closely monitoring and hence controlling the behaviour of workers. Many IS/IT projects have also either been abandoned at great expense, or have failed to deliver expected benefits in use (chapter 37).

So IS do not always have intended benefits. How then can we ensure that: failure does not occur?; negative effects do not occur?; positive effects do occur? The main point is that we cannot do this by directing attention and resources solely at the IS itself. We must consider IS in the context of its organisation, and its organisation in the context of its environment. Perhaps a better way of putting this is that IS in some way contributes to the success of some organisation within its environment. But what makes a successful organisation?

Peter Drucker's (Drucker, 1994) 'The Theory of the Business' attributes organisational success to three factors:

1. Businesses understand their external environments.
2. Businesses undertake missions consistent with their external environments.
3. Businesses develop core competencies needed to accomplish their missions.

## **38.5 Effects of IS/IT**

When an information system is introduced into some organisation it has some affect on that organisation. In general terms we may distinguish between three levels of effects when a new IS is introduced.

### ***38.5.1 First Order Effects (Use)***

The first order effects arising from the introduction of an IS concern the shape of usage of said system. Firstly, we must recognise that many systems may not actually get used; they may be abandoned before they become used. Secondly we must recognise that if a system is used, it may be used in ways unintended in terms of its original design. Such unintended use may be positive as in the case when a decision support system serves not only as a tool for enhancing decision-making, but also as a tool for improving customer relations. The unintended use may also be negative, as when an executive information system is used to intimidate subordinates and stifle creativity rather to enhance managerial processes.

### ***38.5.2 Second Order Effects (Consequences)***

Second order effects or what we have called consequences on figure 38.1 may be tangible improvements in profitability or intangible improvements in effectiveness such as quality, service, and customer satisfaction.

Consequences are also likely to be relevant in terms of human issues. Shifts in power and influence may arise, as for example experienced in relation to the position of office workers such as secretaries when information systems are introduced to replace typical secretarial skills such as typing. In contrast, job enrichment or greater degrees of worker empowerment (chapter 28) may become feasible with the introduction of an IS.

Finally, future flexibility (the infinity sign on figure 38.1) may be affected by an IS. This concerns the ways in which the IS may constrain future IS and strategic initiatives by the organisation. For example, heavy investment in mainframe technology may constrain movements towards client-server in a particular organisation.

### **38.5.3 Third Order Effects (*Adaptation and Change*)**

Third order effects concern the way in which the introduction of an information system feeds back either on itself, or in terms of the organisation as a whole. Firstly, it must be acknowledged that information systems rarely stay still. They change over time in terms of adjustments made to the way both the IS and its context of use works. Secondly, how the system is used, its perceived consequences for performance and people, will affect the organisational context over time. For instance, it may lead to organisational restructuring. There is much evidence of the way in which the development and use of an IS can act as major vehicles for organisational learning (Zuboff, 1988).

## **38.6 The External Environment**

An organisation's external environment is defined by such factors as:

1. The competitive structure of its industry
2. The relative power of buyers and sellers
3. The basis of competition
4. Whether the industry is growing, shrinking or stable
5. The state of regulation
6. The state of technological deployment

An example of a competitive environment is the retail food industry. In the UK this is characterised by the dominance of three big supermarkets which have enormous power in determining pricing levels for key foodstuffs from their suppliers. However, the food retail industry is subject to quite heavy degrees of regulation in such areas as environmental health legislation. In recent years the major supermarkets have increased their levels of technological deployment quite dramatically and are already attempting to utilise their information systems in new areas such as financial services.

Of particular interest to information systems is the position of the organisation in its external environment, e.g.

1. The competitive position of the firm
2. The relationship of the firm to its customers and suppliers
3. Its relative technological capabilities

Strategic information systems (chapter 33) are attempts to influence an organisation's position and relationships through its relative technological capability. A key example here is the way in which the Republic of Singapore's extreme reliance on foreign trade led it to develop an IS (Tradenet) that significantly reduced the time required for shippers to clear customs. The objective was

to give Singapore a competitive advantage over other ports in the Far East.

The external environment and the firm's position in it also influences:

1. Which IS the firm chooses to develop
2. The design of the IS
3. The effect of the IS on its organisation and its industry

The external environment frequently influences the design or adaptation of an organisation's internal information systems. For example, regulatory changes frequently impact on the logic of an organisation's transaction processing and reporting systems. Examples here may be the way in which Electronic Data Interchange (EDI) may be adopted because of indirect pressure from trading partners, or the way in which European Monetary Union may force standardisation amongst financial information systems in the European Union.

The external environment is also especially important for understanding inter-organisational information systems (IOS) and the effect of IOS on an individual firm's performance. Commonplace examples of IOS are the joint networks of automatic teller machines (ATMs) run by consortia of banks and building societies in the UK and the way in which groups of airlines band together to run large airline reservation systems.

An IOS can improve an organisation's performance by reducing transaction costs and differentiating a firm from its competitors. IOS may also significantly change the external environment itself by altering the basis of competition or shifting the balance between buyers and sellers. When airline reservation systems were first introduced into the industry by American Airlines it had a significant effect on the structure of the industry (chapter 33).

### **38.7 Organisational Structure and Culture**

An organisation's structure and culture may influence the design of information systems as well as the success of these IS. Conversely, an organisation's IS may contribute to changes in structure and culture (chapter 3).

Structure refers to the formal aspects of an organisation's functioning: division of labour; hierarchical authority; job descriptions, etc. Clearly a number of structural issues have a direct bearing on both the design as well as the introduction and use of an IS. For instance, a heavily centralised organisation is likely to develop quite different information systems from a decentralised one. Also, divisional, functional, matrix or network structures for organising people (chapter 31) can all influence the likely portfolio of IS.

The deployment of IS can lead to changes in organisational structure. For instance, the introduction of data management may lead to a greater centralisation of decision-making. Alternatively IS/IT can push decision-making down the organisation closer to the workforce. IS/IT can also facilitate newer forms of

organisation, such as virtual or network organisations (chapter 33).

Culture refers to a common set of shared basic assumptions or meanings held by groups of people. Elements of organisational culture that may directly impact upon IS include whether an organisation values individuality or teamwork, whether it favours bigger or smaller organisational units, and whether it favours risk taking or risk aversion in decision-making.

Culture can interact with IS/IT in various ways. For example, groupware systems may find difficulty in being introduced into an organisation where individuality is valued rather than team-work.

### **38.8 Organisational Strategy**

In 1991 a key report was published which documented the results of a major investigation into management issues. Called the 'Management in the 90s' programme, a key emphasis of this report was on how IS strategy needed to align itself with business strategy (Scott-Morton, 1991). By business strategy we mean the general direction or mission of the organisation.

Many IS projects are closely linked to organisational strategy, many are not. In chapter 33 we considered IS planning as being the process driven by a clear idea of strategy. Modern organisations may be subject to a number of different organisational strategies:

1. Low-cost production
2. Focus on quality and service
3. Globalisation
4. Right-sizing
5. Customer intimacy
6. Supplier intimacy
7. Just-in-time manufacturing

In chapter 33 we considered a number of frameworks that have been suggested for analysing the IS/organisational strategy relationship, such as Porter's five forces model, value-chain analysis, etc. The key emphasis of the literature in this area is on the development of strategic information systems (chapter 33). A strategic information system is one which demonstrably improves the competitive advantage of the organisation. However, it is important to recognise that:

1. Strategic IS only stay strategic for a given length of time, usually competitors replicate such systems. The systems then become 'support' or 'factory' systems rather than strategic
2. Strategic systems tend to be high risk systems and hence are perhaps more prone to failure than other types of system (chapter 37)

### 38.9 Processes

A business process is a set of activities cutting across the major functional boundaries of organisations by which organisations accomplish their missions, particularly the key one of delivering value to the customer.

A major emphasis of the recent BPR literature (chapter 25) is that many of the problems of traditional IT arise from the tendency of the IS industry to ‘pave over the cowpaths’. In other words, IT has been used conventionally to automate existing processes. People like Davenport, Hammer and Champy have argued that, when business processes are automated without first improving them, organisations will not continue to achieve significant benefits from their large investments in IT.

The major premise here is that when automation is confined to small pieces of a business process (such as that which falls within the domain of a particular functional unit) then the large process may become degraded. IS cannot work in isolation. Most effective use of IT frequently involves redesigning entire business processes.

Designers of Tradenet, for instance, began by considering which business processes were the key to the success of the Singapore economy. They came to the conclusion that trade was the number one priority. Within trade the processes judged to have the greatest impact on economic competitiveness were those that directly influenced the speed of transit for goods passing through the port of Singapore; among these customs clearance was the most visible. Tradenet's designers were consequently not happy with incremental improvements to this process; they radically transformed the process, then computerised it.

### 38.10 Infrastructure

An IT infrastructure equals the set of organisational resources that give the organisation the capacity to generate new IT applications. An IT infrastructure includes not only physical resources such as existing hardware, software, and communications, but also non-physical resources such as knowledge, plans, people and skills.

It must be recognised that an IT infrastructure is both enabling and constraining.

Any application of IT can leverage or extend the existing IT infrastructure. A given IS leverages the infrastructure when it draws upon the resources the existing infrastructure offers. IS extends the infrastructure by contributing physical or non-physical resources that can be drawn upon by other applications. Tradenet leveraged existing governmental transaction processing systems by linking traders in. The process of implementing Tradenet also extended Singapore's stock of IS related skills.

However, an IT infrastructure may also be a constraining influence. For

instance, legacy systems, that is, large and ageing corporate support systems, generally constrain the organisation in the sense that they determine the way in which much corporate data must be collected, manipulated and distributed.

The so-called 'millenium bug' is one important example of the constraining influence of the existing IS portfolio. This bug represents the way in which many systems have been found to be deficient in their ability to handle 21st century dates. The presence of this bug has forced many organisations on an international scale to divert vast development sources to solving the problem and consequently has placed a brake on new information system developments.

### **38.11 Features of an Information System**

Traditionally, design features of an information system fall into one of two categories: functionality or what the system does; usability or how the system is used. One should note that both functionality and usability are inherently related to the place of the information system within the context of organisational work. Functionality is hopefully determined by a close examination of organisational requirements. Usability is evident in the way in which an IS embeds itself within work.

To these two we should add a third, the utility or efficacy of an IS. Whereas functionality defines what a system does and usability defines how a system is used, utility defines how acceptable the system is in terms of doing what is needed.

The focus within IS development has historically been on issues of functionality. The concern has been and still is to produce well-engineered systems. But well-engineered systems are insufficient in themselves. They must also be usable. Hence, much work has been undertaken in improving user interfaces (chapter 23).

However traditionally IS have been built to merely automate existing patterns of work – particularly to replicate manual functions. There is evidence to suggest that the utility of such systems, however important, is limited. So, increasingly IS are being used to design new ways of working in an attempt to gain greater leverage in terms of utility for organisations.

Functionality, usability and utility are three necessarily interdependent variables that serve to define the interaction between the design of an information system and the work that it is meant to support or enable. Clearly, the design of an IS can affect this interaction in various subtle ways.

For instance, Shoshanna Zuboff (1988) has illustrated how computer systems can be designed in which users are placed outside the system, engaged in merely monitoring the work process. The alternative is that the computer system can be defined such that work becomes more visible and workers are given close involvement in the work process. Zuboff calls this the choice between automating on the one hand and informing on the other. This is similar to the work of

Mumford (chapter 28) in which the emphasis is on the design of work in parallel with the design of technical systems. As an example, an information system may be designed to continually monitor the work of insurance clerks, or support and enable the individual and team-work of clerks.

### 38.12 IS Development Process

Figure 38.1 outlines a four stage model of the IS development process roughly compatible with the standard system development life-cycle or waterfall model. However, the general stages represented here do not prohibit alternative development approaches such as prototyping and spiral approaches (chapter 7). The four stages are:

1. *Initiation*. The decision to begin the development process and the necessary planning that needs to be put in place (chapter 7).
2. *Build/buy*. The construction of the information system. This may either be conducted by a team internal to the organisation or undertaken by an outside contractor (chapter 35). Many information systems are now also bought in as a package and tailored to organisational requirements.
3. *Introduction*. By this we mean the implementation and use of the information system within the organisation.
4. *Adaptation*. This is the feedback process which involves changes to information systems and to organisational elements.

Note that the way in which a system is developed can affect system acceptance. Giddings (1984) characterises information systems as domain-dependent software. Information systems are domain-dependent because there is a necessary interdependence between the software system and its domain or universe of discourse. Such systems are characterised by an intrinsic uncertainty about the universe of discourse. In particular, the use of the software may change the nature of the universe of discourse and hence the nature of the problem being solved.

Conventional approaches to development tend to treat this phenomenon as something of an aberration that has to be worked around in the context of a development project. In this chapter we treat this phenomena as a fundamental premise of systems development that must be accommodated within the development process itself. We believe that this uncertainty which Giddens describes is merely a symptom or a reflection of the organisational learning focused in the introduction of information technology. However to achieve such accommodation demands a critical reflection on the way in which information systems are conventionally developed.

### **38.13 Fit Between IS and Organisational Context**

The central idea discussed in this chapter is that an information system must ‘fit’ its organisational context: the organisation, its strategy, its business process, its environment. Information systems that do not fit are likely to be resisted, underused, misused, sabotaged, etc. Information systems that do not fit are likely to have negative effects on organisational performance.

The obvious lesson here is that organisations should conduct a thorough analysis of needs prior to systems development to promote positive effects and systems success. This is what we described in chapters 25 and 26 as business analysis.

However, a model of ‘fitness for purpose’ tends to assume a stable context. What if the organisation wants to use IT to change users’ behaviour, organisational culture, or business performance? In this situation the organisation must confront the need to change other aspects of the organisation (structure, culture etc.) either prior to or simultaneously with the introduction of the IS.

The BPR literature has argued that developers have sought to fit at the expense of change by automating flawed business processes. When developing information systems organisations therefore adopt one of two strategies:

1. Incremental improvement
2. Radical change

The fit model can be used in the first case to examine systems success. In the latter case, the situation is more complex since the system may fail:

1. Either by fitting the existing organisation too well so that not enough change occurs
2. Or by fitting a new organisation too poorly.

This may account for the evidence of the high failure rate of re-engineering efforts

In summary, the success of an information system is largely defined in terms of its contribution to the success of an organisation in its environment. All the topics relating to this are highly interconnected and are discussed as separate entities merely for convenience of presentation. Nevertheless, Kling and Allen (1996) have argued that an understanding of what they call organisational informatics is essential for any persons working in the computer industry. This is the issue of the next chapter. Also, the link between organisations and information technology is a critical defining characteristic of the discipline of information systems. This is the topic of chapter 40.

### 38.14 References

- Brynjolfson E. (1993). 'The Productivity Paradox of Information Technology'. *CACM*. 36(12). 67-77.
- Cash J.I. Mcfarlan F.W., McKeney J.L. (1992). *Corporate Information Systems Management*. 3rd Ed. Richard Irvin.
- Drucker P.F. (1994). 'The Theory of the Business'. *Harvard Business Review*. 72(5). Sept/Oct. 95-104.
- Giddings R.V. (1984). 'Accommodating Uncertainty in Software Design'. *CACM*. 27(5). 428-434.
- Kling R. and Allen J.P. (1996). 'Can Computer Science Solve Organisational Problems? The Case for Organisational Informatics'. In Kling R. (Ed.). *Computerisation and Controversy: value conflicts and social choices*. 2nd Ed. Academic Press, San Diego.
- Moshowitz A. (1976). *The Conquest of Will: Information Processing Human Affairs*. Addison-Wesley, Reading, Mass.
- Scott-Morton M.S. (Ed.). (1991). 'The Corporation of the 1990s: information technology and organisational transformation'.
- Silver M.S., Markus M.L. and Beath C.M. (1995). 'The Information Technology Interaction Model: a foundation for the MBA core course'. *MIS Quarterly*. 19(3). Sept. 361-390.
- Strassman P. (1990). *Business Value of Computers*. The Information Economics Press, New Canaan, Conncticutt.
- Thurrow L.C. Foreword. In M.S.Scott-Morton. *The Corporation of the 1990s: IT and Organisational transformation*. Oxford University Press, New York.
- Zuboff S. (1988). *In the Age of the Smart Machine: the future of work and power*. Heinemann, London

### 38.14 Exercises

1. In terms of the information systems portfolio of some organisation known to you, try to identify a list of benefits under the three headings of profitability, productivity and effectiveness.
2. In terms of a chosen organisation try to describe: the external environment, organisational strategy, business processes, structure and culture and IT infrastructure.
3. Choose one information system developed by some organisation. Try to see if you can identify first, second and third effects which have arisen from the introduction of the information system.

# **39 The Professionalisation of IS Work**

## **39. Introduction**

The ubiquity of modern information systems has caused a debate within the community involved in their development about the status of their work. Societies such as the British Computer Society (BCS) and the Institute of Electronic Engineers (IEE) have attempted to cast information systems work as a true profession in much the same guise as lawyers, accountants, architects and the medical profession (Beynon-Davies, 1993).

There are clearly a number of benefits for organisations, economies and societies that may arise from the professionalism of IS. For instance, it is a truism at the current time that any person with a PC and an interest in computing can develop and sell software. Hence, there is very little regulation in place to ensure that software developed is of a sufficient quality and standard to meet the needs of industry and the economy. Also, there is concern that there is little recourse to the purchaser of such software in terms of the market (Myers *et al.*, 1997).

However, there are also a number of barriers that exist which will inevitably slow down and possibly prevent the professionalism of IS work. In this chapter we try to assess what constitutes a profession and how likely it is that IS engineering will become a true profession in the 21st century.

### **39.2 Trait Models of Professions**

Many occupational groups have sought professional status. However, there is little agreement about what constitutes a profession. Elliot (1972) discusses how the term *profession* is widely and imprecisely applied to a variety of occupations. The adjective *professional* is even more overworked, extending to cover the opposite of amateur and the opposite of poor work, two concepts which need not be synonymous. Some sociologists, disillusioned somewhat with the hazy nature of the term, have concluded that the concept of a profession is of little scientific value. They therefore cast the term profession, merely as a symbolic label for a desired status.

The traditional model of professionalism in the literature might be referred to as the trait model (Johnson, 1982). Trait models of professionalism present a list of attributes which are said to represent some common core of professional occupations. In this model the process of professionalisation is portrayed as a process made up of a determinate sequence of events. An occupation is seen as passing through predictable stages of organisational change, the end-state of which is professionalism.

Trait models are normally constructed by defining some common features from occupational groups with a readily agreed professional status, e.g. law and medicine. A feature analysis of such occupations leads to the development of the following ‘ideal-type’ (Jackson, 1970).

A profession might be defined as a group of persons with:

1. An accredited corpus of specialist knowledge and skills.
2. One or more formal bodies involved in organising the profession. In particular, the formal body will be involved in: maintaining a code of conduct/practice and monitoring adherence to the code(s); ensuring that only suitably ‘qualified’ persons enter the profession.
3. Some form of recognition of the professional status of members.

### ***39.2.1 An Accredited Corpus of Specialist Knowledge and Skills***

To be a profession some agreement has to have been reached in the occupational group regarding the content of theoretical knowledge and practical expertise demanded of members of the profession.

A profession normally sees itself as having an essential underpinning of abstract principles which have been organised into some theory. Alongside the set of basic principles are various practical techniques for the recurrent application of at least certain of the fundamental principles. Simon (1972) sees professions as denigrating knowledge that is ‘intuitive, informal or cookbooky’. Murray (1991) describes this ideology in terms of technical rationality.

Derber *et al.* (1990) cast professionals in the role of a new class which relies mainly on claims to knowledge rather than labour or capital as the basis for their quest for wealth and power. Such knowledge is credentialised, i.e. usually certified by some higher education institution. Jackson (1970) also sees a definite link between the rise of the universities and the established professions such as law and medicine.

### ***39.2.2 One or More Formal Bodies Involved in Organising the Profession***

Established professions such as medicine have long-standing bodies such as the British Medical Association (BMA) which organise professional activities within the UK. Such bodies maintain a code of conduct defining the limits of medical practice. The BMA also accredits recognised medical schools and ensures that only suitably qualified persons enter the medical profession.

Most professions build an ideology based around notions of professional behaviour. This usually includes aspects such as:

1. An orientation to the community interest.

2. An internalised code of ethics.
3. Rewards based primarily in symbolic work achievement.

### ***39.2.3 Some Form of Recognition of the Professional Status of Members***

Recognition is usually a critical aspect of professionalisation. Such recognition can take two forms: state recognition and public recognition.

In the UK, Bott *et al.* (1995) discuss state recognition as being embodied in the assignment of a royal charter to a collective body. This charter defines the extents of the profession's authority and requires it to undertake certain duties and responsibilities.

A profession also needs public recognition, particularly from its potential customers. Only with such recognition does the status of a profession act as a useful lever in the marketplace.

## **39.3 Information Systems Work as a Profession**

In this section we shall discuss whether information systems work in the UK constitutes a profession at the present time.

### ***39.3.1 Corpus of Knowledge and Skills***

There is little agreement concerning the corpus of knowledge and skills constituting information systems work. In the UK, a number of standards have been developed for IS work in the public sector, the most notable being the systems development methodology SSADM and the project management methodology PRINCE. However, it is still not true to say that an information systems specialist has an agreed body of transferable knowledge and skills which he can take with him when moving between positions.

There is also a big question about whether purely technical knowledge is sufficient. Friedman and Kahn (1994) have discussed the importance of including material on social and ethical issues within standard computer science curricula. Also, institutions like the British Computer Society have recently become interested in broadening the skills expected of information systems engineers. The idea of a hybrid manager has been much discussed (Palmer, 1990). The term hybrid manager was coined by Michael Earl to define a manager who combines information systems and business skills (Earl, 1989):

‘People with strong technical skills and adequate business knowledge, or vice versa...hybrids are people with technical skills able to work in user areas doing a line or functional job, but adept at developing and supplementing IT application ideas.’

### **39.3.2 Formal Body**

A number of bodies are competing to represent work within commercial computing in the UK. The main practical problem in turning information systems work into a profession is the fact that as little as 20% of the persons involved in IS belong to bodies like the BCS.

The BCS lays down a code of conduct and a code of practice. The code of conduct is concerned with questions of ethics; standards of behaviour. A member breaching these principles is said to be guilty of professional misconduct. The code of practice lays down, in very general terms, the way in which members shall approach the development of information systems.

The BCS has a number of routes to membership: a person can apply for membership after a certain number of years experience in the field, a person can pass the BCS part 1 and part 2 examinations, or a person can pass a degree in computing at an accredited university and obtain exemptions from part 1 and/or part 2 exams.

### **39.3.3 Recognition**

The British Computer Society attained chartered status in 1984.

There is some debate about the public recognition of the BCS. For instance, during the highly publicised systems failure at the London ambulance service the BCS was not reported in the general press. Most of the statements made by the BCS were made in the computing press (Beynon-Davies, 1995).

The main arguments for professional status are that in a time of increasing disquiet over the quality of information systems, a body such as the BCS can guarantee persons able to build quality systems. This can quite clearly be seen as an attempt to promulgate an ideology of public service in information systems work. It is clearly one manifestation of the attempt to link issues of accountability with good practice in systems development (Nissenbaum, 1994).

## **39.4 Semi-profession**

The key conclusion is that information systems engineering constitutes at most a semi-profession at the present time. Etzioni (1969) defines a semi-profession as being an occupational group in which 'their training is shorter, their status is less legitimised, their right to privileged communication less well established, there is less of a body of specialist knowledge, and they have less autonomy from supervision or societal control than 'the professions'. Teachers, nurses and social workers are given as three prime examples of the semi-profession.

### **39.5 The Process of Professionalisation**

A number of critiques have been made of trait models of professionalism. Such critiques maintain that a profession is not an ideal-type. A profession is not an occupation, it is a means of controlling a profession. For instance, Johnson (1982) clearly presents an analysis of professions in terms of power relations in society. Professionalism is seen as a process of yielding power resources, such as the application of knowledge and skills, to further the interests of a given occupational grouping.

The overall objective of most aspiring professions is to claim a monopoly over occupational activities. This monopoly is normally justified in terms of ensuring quality of work and accountability. The monopoly is normally achieved by the professional body enforcing certain barriers to entry. Only those persons credentialised with professional knowledge and expertise are allowed entry into the profession.

Illich *et al.* (1977) see professions as disabling in that they remove the individuals right to determine problems, suggest solutions and determine needs. Professions impute needs as a means of preserving power. They embody an enslaving illusion that people are born to be consumers and that they can attain any of their goals by purchasing goods and services.

Much of the work of bodies like the BCS can be cast in terms of this power-play. Two examples are given here of the way in which IS workers are attempting to improve their power-position.

### **39.6 Professional Development**

As a practical step towards the goal of professional status, a formal scheme for professional development has been constructed by the BCS. This professional development scheme (PDS) has also been adopted by many large IT using organisations such as BP, British Gas, BT and ICI. The scheme is built on an industry structure model which defines some 40 roles in computing management, systems development, technical support, auditing and training. Each role can have a number of experience levels. At each experience level there is a description of the type of work done, the experience and skills expected and the training needed to prepare a person for the next level. A European Informatics Skills structure based upon this model is being positioned to support the mobility and transfer of IT skills in the single European market (Him, 1993). In this light, the BCS helped found the Council of European Informatics Societies (CEPIS), through which it gains access to the European Commission.

The main practical problem is that the BCS PDS has only diffused to a limited degree throughout the industry. This makes it difficult for the body to enforce the PDS as an established career structure for information systems personnel.

### 39.7 Information Systems Work as Engineering

The BCS calls itself the Society of Information Systems Engineers. It is therefore not surprising to find that members of the BCS now have a route to chartered engineering status. The BCS became a chartered engineering institution in May 1990.

It is no accident that professionalism and chartered engineering status are married in the minds of a body such as the BCS. In most people's minds the concept of engineering is equated with the application of scientific principles to the construction of artefacts such as buildings and bridges. For instance, Wright (1989) defines engineering as:

‘The profession in which a knowledge of the mathematical and natural sciences gained by study, experience, and practice is applied with judgement to develop ways to utilise, economically, the material and forces of nature for the benefit of mankind.’

This straightforward characterisation of engineering as science is one which has influenced much of the movement attempting to place information systems work on an engineering footing. This is clearly evident in the creation of the ‘hacker’ as a negative role model for computing specialists.

However, it is interesting that many in traditional engineering disciplines such as civil and structural engineering have long cast doubt over the purely scientific nature of the discipline. Petrowski (1985), for instance, quotes the following as a definition of structural engineering:

‘Structural engineering is the science and art of designing and making, with economy and elegance, buildings, bridges, frameworks, and other similar structures so that they can safely resist the forces to which they may be subjected.’

The important part of this definition is that structural engineering is seen to be as much art as science. Elegance of design is given equal footing with economy of design. The principles of aesthetics are at least as important as the principles of mechanics.

What is also interesting is the emphasis in the definition on safety. This is resonant of much of the discussion in software engineering, particularly in the area of safety critical systems (Leveson, 1986). The ability to formally prove that an application meets its specification is seen to be important in certain applications such as military command and control systems. Indeed, recently the concept of a safety-critical system has been broadened in the context of systems failures such as the London Ambulance Service system (Beynon-Davies, 1995).

However, perhaps the most important reason that information systems work has been portrayed as engineering is that it is inherently bound up with the

attempt to place the information systems specialist on a professional footing. What is interesting is that although information systems work has used engineering as a professional status symbol, there may be problems embodied in the professionalisation of engineering disciplines. For instance, Larson (1977) has cast doubt on the position of engineering as a profession. She makes a useful contrast with medicine. Unlike medicine:

1. Engineering is not, and never has been a homogeneous area.
2. The physical nature of the engineer's product immediately involves the possibility that the buyer can be a different person from the consumer.
3. The engineer's marketplace is subordinated, i.e. the services of an engineer are more likely to be mediated by organised clienteles than in a profession such as medicine which provides intangible services.

All of these characteristics make it difficult for engineers to maximise their effectiveness. The engineer is usually dependent on knowledgeable and powerful clients. The nature of an engineer's product implies that the public can obtain more direct evidence of a professional's capacity. This also makes it easier for the state to intervene and regulate an engineer's authority.

### **39.8 Conclusion**

Professionalism of IS would be likely to accrue a number of benefits for organisations, societies and economies. However, a number of appreciable barriers exist in the way of the professionalisation process of this body of work. An analysis of established professions gives us the following key features as defining an ideal-type of profession: an agreed corpus of knowledge and skills; a collective body; recognition. Assessing IS work against each of these features leads us to conclude that at present it is at most a semi-profession. The trade has many of the trappings of a profession such as a formal body. However, such a body at present suffers from low membership and diffuse public regard.

Information systems work as an occupation is clearly seeking full professional status. Statements made by many in the aspiring professional bodies emphasise some of the classic elements of professional ideology. However, a number of positive and negative forces are likely to influence the process of professionalisation in this occupation over the next decade.

On the positive side, collective bodies are attempting to strengthen their institutional position in the marketplace. The PDS and CEng initiatives in the BCS are clear examples of this.

On the negative side, rapidly changing technology has meant a rapidly changing body of knowledge and skills in the industry. The division of labour within the industry is less than clear cut, being subject to hybridisation on the one hand and specialisation on the other. Information systems engineering is

also subject to the classic low power position of engineers compared to other occupational groups.

### 39.9 References

- Beynon-Davies P. (1992). 'The Realities of Database Design: an essay on the sociology, semiology and pedagogy of database work'. *Journal of Information Systems*. 1(2). 207-220.
- Beynon-Davies P. (1993). *Information Systems Development: an introduction to information systems engineering*. 2nd Ed. Macmillan Press, Basingstoke.
- Beynon-Davies P. (1995). 'Information Systems "Failure": The case of the London Ambulance Service's Computer Aided Despatch System'. *European Journal of Information Systems*. 4. 171-184.
- Bott F., Coleman A., Eaton J., Rowland D. (1995). *Professional Issues in Software Engineering*. 2nd Ed. Pitman, London.
- Computer Weekly. (1992). 'BCS President blames training for 999 failure'. 19th Nov. 1992.
- Earl M.J. (1989). *Management Strategies for Information Technology*. Oxford University Press, Oxford.
- Elliot P. *The Sociology of the Professions*. Macmillan, London. 1972.
- Etzioni A. *The Semi-Professions and their organisation: teachers, nurses and social workers*. Free Press, New York. 1969.
- Friedman B. and Kahn P. (1994). 'Educating Computer Scientists: Linking the Social and the Technical'. *Comm. of the ACM*. 37(1). January. 65-70.
- Him D. (1993). 'Professional Development Initiatives'. *Computer Bulletin*. 5(2). April. pp 24-25.
- Illich I., Zola I.K., McKnight J., Caplan T., Shaiken H. (1977). *Disabling Professions*. Maria Boyars, New York.
- Jackson J.A. (Ed.). (1970). *Professions and Professionalisation*. Cambridge University Press, Cambridge.
- Johnson T.J. (1982). *Professions and Power*. Macmillan, London.
- Larson M.S. (1977). *The Rise of Professionalism: A Sociological Analysis*. Univ. of California Press. Berkeley, Calif.
- Leveson. N. G. (1986). 'Software Safety: why, what and how'. *ACM Computing Surveys*. 18(2). 125-163.
- Murray F. (1991). 'Technical Rationality and the IS Specialist: Power, discourse and identity'. *Critical Perspectives on Accounting*. 2 (59-81). 1991.
- Myers C., Hall T. and Pitt D. (Eds.). (1997). *The Responsible Software Engineer: selected readings in IT Professionalism*. Springer, London.
- Nissenbaum H. (1994). 'Computing and Accountability'. *CACM*. 37(1). 73-80.
- Palmer C. (1990). 'Hybrids – A Growing Initiative'. *The Computer Bulletin*. 2(6). August.

- Petrowski H. (1985). *To Engineer is Human: The role of failure in successful design*. Macmillan, London.
- Simon H. *The Sciences of the Artificial*. MIT Press, Cambridge, Mass.
- Skyrme D.J. and Earl M.J. (1990). 'Hybrid Managers: what should you do?' *The Computer Bulletin*. 2(4). May.19-21.
- Wright P. (1989). *Introduction to Engineering*. John Wiley, New York.

### 39.10 Exercises

1. In your organisation find out how many people see themselves as professionals? How many belong to an organisation such as the BCS? Ask some of the members and non-members about their attitudes to the BCS. What do they see to be the key benefits of BCS membership?
2. Discuss briefly what you feel are the main advantages of having a professional development scheme in place across the UK, and perhaps Europe.

# **40 *The Development of the Discipline of Information Systems***

## **40.1 Introduction**

In this, the final chapter, we discuss the development of the discipline of information systems. First, we return to the issue of what constitutes the discipline. Second, we consider the place of information systems development within the discipline of information systems. Third, we discuss areas of much-needed research in information systems development. Fourth, we describe some possible strategies for ISD research.

## **40.2 Definitions**

Two definitions for the discipline of information systems are provided below:

‘The information systems discipline is defined as the effective analysis, design, delivery, and use of information and information technology in organisations and society and implies that issues relating to people, organisation and society are relevant as are non-computerised information systems.’

(Avison and Fitzgerald, 1990)

‘The study of information systems and their development is a multi-disciplinary subject and addresses the range of strategic, managerial, and operational activities involved in the gathering, processing, storing, distributing and use of information and its associated technologies, in society and organisations.’

(UK Academy of Information Systems Newsletter, 1995)

There are a number of elements contained within these definitions that have been elaborated upon throughout the various chapters of this text:

1. That IS concerns itself with information in general, as well as the more specific topics of information systems and information technology.
2. That an information system need not necessarily be computerised; that the processes of gathering, processing, storing and distributing information have been undertaken in human societies for many thousands of years with various technologies. Computer and communication technology is only the latest, if highly ubiquitous, example of such ‘information technology’.

3. That the idea of information and an information system cannot be understood properly without a description of its context. Usually the level of context is an organisation or part of an organisation within which such systems are placed. Sometimes, the level of societies and economies are important, as is the case for instance with systems making up the financial infrastructure of a nation. More recently, the focus has shifted up to the global scale in the case of such an 'information system' as the Internet.

#### **40.3 How Do We Know that IS is a Discipline?**

Information systems is a relatively young discipline striving currently for a distinctive face as compared to older more established disciplines such as computer science and management science. So how do we know that it constitutes a distinct discipline? Some of the reasons are provided below:

1. There is currently a substantial community of scholars both within the UK and throughout the world who label themselves as IS academics.
2. Within the UK there are some 10-20 departments which either call themselves Information Systems departments, or have the term somewhere in their title. There are many more of such departments world-wide. IS people can also be found in Computer Science/Studies departments, Information Science/Library Studies departments, as well as Business Studies departments and Business schools. The UK Academy of Information Systems lists some 85 departments in the UK that run Information Systems courses.
3. There are a number of established conferences for the discipline such as the *International Conference on Information Systems* (ICIS) and the *European Conference on Information Systems* (ECIS).
4. A number of journals exist for publishing the results of research in this area such as the *Information Systems Journal*, *European Journal of IS*, and *MIS Quarterly*.
5. A number of bodies exist to foster communication, co-operation and collaboration for IS academics such as the Association of IS and the UK Academy of IS.
6. Professional Bodies exist which support the work of IS practitioners. Within the UK, for instance, the British Computer Society (BCS) sees itself as the professional society for Information Systems Engineering (chapter 39).

#### **40.4 The Composition of IS**

In very general terms it is useful to see IS as being made up of four inter-dependent areas of interest (figure 40.1):

1. *IS environment.* This constitutes the economic and social environment within which the IS domain takes place.
2. *IS management.* In very broad terms we may define this area as mainly interested in the link between IS and organisational effectiveness.
3. *IS technology.* Information technology is an important component to IS work. However, IS takes a balanced interest in both the use of technology as well as the principles underlying technology.
4. *IS development.* Here the concern is with appropriate ways of constructing information technology systems that support human activity, particularly decision-making. This means that certain classes of software system such as real-time process control are not seen as areas of concern.

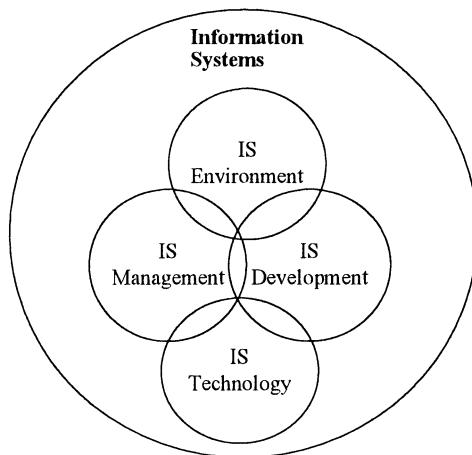
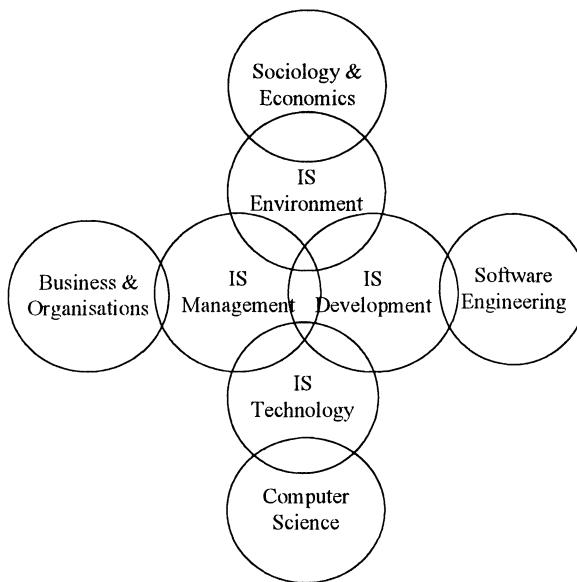


Figure 40.1: Areas of Information Systems

In terms of the three-fold division of IS interests detailed above, IS can be seen to overlap with five other established disciplines (see figure 40.2):

1. Through its emphasis on context, particularly the social and economic effects of IS/IT, it overlaps with the disciplines of economics and sociology.
2. Through its emphasis on organisational issues, there is a clear overlap with management science/organisational theory as well as business studies.
3. In its interest in computing and communications technology, there is a clear overlap with many of the issues underlying computer science.
4. In terms of its interest in the development process there is clear overlap with the best practice described in software engineering principles.



**Figure 40.2: Overlap of Areas**

IS is therefore a multi-faceted and potentially a multi-disciplinary endeavour. But what defines the core essence or mission of IS? To risk over-simplifying we might argue that the concern is with the necessary interaction between information systems and organisations (chapter 38).

## 40.5 UKAIS Definition of Information Systems

The UK Academy of Information Systems (UKAIS, 1997) has proposed a definition for the discipline of information systems in terms of product, domain and scope.

### 40.5.1 Product

Information systems are the means by which organisations and people, utilising information technologies, gather, process, store, use and disseminate information.

### 40.5.2 Domain

The domain of information systems requires a multi-disciplinary approach to studying the range of socio-technical phenomena which determine their development, use and effects in organisation and society.

#### 40.5.3 Scope

1. Theoretical underpinnings of information systems
2. Data, information and knowledge management
3. Integration of information systems with organisational strategy and development
4. Information systems design
5. Development and maintenance of information systems
6. Information technologies as components of information systems
7. Management of information systems and services
8. Organisational, social and cultural effects of technology-based information systems
9. Economic effects of technology-based information systems

Figure 40.3 illustrates how we might plot these sub-areas against the previously defined areas of interest.

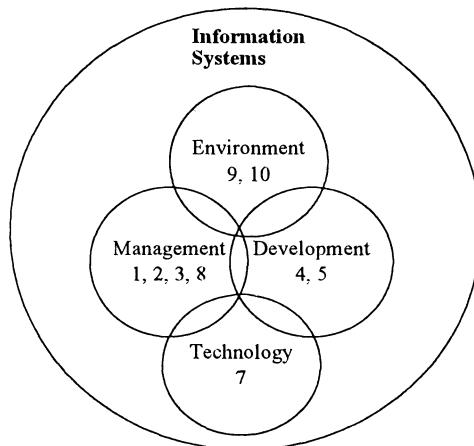


Figure 40.3: Breakdown of sub-areas

What is perhaps noteworthy about this assignment is the concentration of sub-areas in what we have called IS management. IS development comes a close second. This perhaps reflects something of an imbalance in the amount of IS research being conducted on issues relating to the management of IS, as compared to that devoted to issues of development.

To take an example, the empirical study of information systems development methodologies (ISDMs) is a much neglected area of information systems research. Wynekoop and Russo (1997), for instance, conducted a systematic survey of the existing literature on ISDMs. They found that over half of the 123 research papers examined consisted of normative research in which concept development was not based on any empirical grounding or theoretical analysis,

but merely on the authors' speculations and opinions. Of those which constituted empirical research, almost half were undertaken to evaluate ISDMs or parts of ISDMs. Few studies were undertaken to identify how ISDMs are selected or adapted or how they are used.

This is a key example of the paucity of real research data that we have on ISD. It should also stimulate some much needed research in this area.

#### **40.6 Levels of ISD Research**

But to what sort of issues should ISD research be addressed? Here we suggest that ISD research should be seen as addressing a number of levels of focus that slice through the experience of ISD:

1. *Tools*. The technology available for developing information systems.
2. *Techniques*. Practices of analysis, design, implementation, evaluation and maintenance.
3. *Methods*. Frameworks of suggestive approaches to building IS.
4. *Management*. Issues surrounding the management of ISD within and between organisations.
5. *Process*. The study of the practical experience of ISD. The way in which tools, techniques, methods and managerial approaches are utilised to address specific organisational problems.
6. *Context*. The environment in which ISD takes place. This includes issues of organisation, industry, society and economy.

The table on the following page plots these levels against the two dimensions of theoretical research and empirical research. Theoretical research in relation to ISD tends to be necessarily prescriptive. In contrast, empirical research is necessarily descriptive. The cells of the table contain with examples of research in each of the areas of focus.

Wynekoop and Russo (1997) indicate that normative or prescriptive research tends to dominate in the area of methods, whereas the focus on practical issues in terms of ISDMs tends to involve evaluation issues only. Although there is very little data on this issue, experience of the field tends to suggest that this imbalance in terms of description tends to occur across tools, techniques, process and context also. For instance, many of the tools, techniques and approaches described in previous parts of this work although heavily promoted in the industry have been subject to little empirical investigation. To take one example, although object-oriented approaches to development have been much promoted in recent years, there is a lack of systematic evidence which allows us to assess the costs and benefits of this particular approach.

Focus/Area	Theory	Practice
<b>Tools</b>	Descriptions of new development tools Theoretical assessments of tools Theoretical comparisons of tools	Empirical evaluations of development tools Investigations of how tools are selected Investigations of how tools are used
<b>Techniques</b>	Descriptions of new development techniques Theoretical assessments of techniques Theoretical comparisons of techniques	Empirical evaluations of development techniques Investigations of how techniques are selected Investigations of how techniques are used
<b>Methods</b>	Proposals for new ISDMs or parts of ISDMs Theoretical assessments of ISDMs Theoretical comparisons of ISDMs	Empirical evaluations of ISDMs or parts of ISDMs Investigations of how ISDMs are selected Investigations of how ISDMs are adapted and used
<b>Management</b>	Proposals for new managerial approaches Theoretical assessments of existing managerial approaches Theoretical comparisons of managerial approaches	Empirical evaluations of managerial approaches Investigations of how managerial approaches are selected Investigations of how managerial approaches are adapted and used
<b>Process</b>	Descriptions of approaches to the development process	Empirical evaluations of approaches to the development process
<b>Context</b>	Theoretical descriptions of issues in the ISD environment	Empirical investigation of issues in the ISD environment

#### 40.7 Research Methods

A key conclusion is that empirical research of ISD is seriously lacking. Whilst there are no shortage of suggestions as to appropriate ways of developing information systems we have very little data on whether any of the vast array of tools, techniques, methods and approaches are really effective. But if more empirical research is needed, how should such research be conducted?

Since ISD is a branch of IS one would expect it to use the research techniques appropriate to this field. IS has generally utilised techniques from the

social sciences to gather data (Galliers, 1992). The list below indicates some, but not all, of the research techniques currently being used in IS research:

1. *Surveys.* These are particularly used to gain an overview appreciation of the distribution of certain characteristics amongst a population. Surveys have been used in the context of ISD research, for example, to determine the take-up by organisations of particular tools, techniques or methods.
2. *Documentary analysis.* The analysis of documents is useful to gain an in-depth appreciation of how an individual, group or organisation accounts for its activities. In terms of the process of ISD, documents are frequently used as a major means of communication between diverse groups. They therefore form an invaluable source of data in the description of the practice of development work.
3. *Interviews.* This research technique is frequently used as a means of gaining an in-depth appreciation of some phenomena. In terms of ISD, interviews have been used as a means of gathering data on issues such as the way in which methodologies have been used in practice.
4. *Observation.* This research technique is important as a means of gaining detailed data on what people actually do. Hence observation is important, for instance, as a means of verifying whether developers follow 'best practice', and what sorts of everyday contingencies affect the trajectory of a development project.

In terms of any one particular IS research project, one or more of these research techniques may be used in combination. This is frequently known as 'triangulation' of data collection methods, and is intended to provide a more complete picture of some phenomenon through exploiting the inherent strengths of each technique. Hence, if we were studying a phenomenon such as the contemporary practice of object-oriented programming we probably would first wish to conduct a survey to gain an appreciation of the scale of adoption of this practice. We would then wish to interview staff at a number of organisations that had indicated adoption to gain an in-depth understanding of their utilisation of this approach. Finally, we would probably wish to observe on one or more development projects utilising object-oriented programming to perceive some of the strengths and weaknesses of this development paradigm in practice.

#### **40.8 Conclusion**

Capers Jones (1996) makes the interesting comment that unlike the military and software systems areas, the IS development industry has no institution such as the Software Engineering Institute tasked with continually examining best practice in this area. This is particularly surprising given the fact that the IS industry forms a larger proportion of the business of computing than the military and systems areas.

One characteristic of the ISD industry is that new tools, techniques, methods and managerial approaches tend to arise out of the practitioner community with very little empirical validation. Hence, much of the industry tends to take on board such new tools, techniques, methods and managerial approaches largely on the basis of blind faith. Harel (1980), usefully characterises much of the knowledge underlying ISD as being based on what he calls folk theory – a collection of accepted wisdom that has three major characteristics: popularity, anonymous authorship, and apparent age.

It seems to us that it is essential to the practice of ISD that this cycle of generating folk theory is broken. Only if a true discipline of ISD is created, based upon the systematic investigation of this activity and the construction of best practice on the basis of empirical evidence, will this area of work achieve anything like the degree of professionalisation that it desires and produce anything like the products to which it aspires.

#### 40.9 References

- Avison D. and Fitzgerald G. (1990). Editorial. *Information Systems Journal*. 1(1).
- Academy of Information Systems Newsletter. September 1995.
- Galliers R. (1992). *Information Systems Research: issues, methods and practical guidelines*. Blackwells.
- Harel D. (1980). 'On Folk Theorems'. *CACM*. 23(7). 379-389.
- Jones C. (1996). *Patterns of Software System Failure and Success*. Thomson, London.
- Wynekoop J.L. and Russo N.L. (1997). 'Studying system development methodologies: an examination of research methods'. *Information Systems Journal*. 7.(1). 47-65.

# ***Index***

- Abstraction Mechanisms 145-6  
Accommodation 136, 147-8  
Action Research 256  
Active Data Dictionary 180  
Aggregation (Relationship) 145, 216  
*AI See Artificial Intelligence*  
AKO Relationship 116  
Application (System) 39  
Applications Development Portfolio 313  
Application Generator 96  
Artificial Intelligence 122  
Assembly Language 69  
Association (Relationship) 116, 146  
Attribute 142, 217  
Authority 7  
  
Backus-Naur Form (BNF) 181-2  
Backward Chaining 122  
Back-End CAISE 104  
Batch Systems 41  
Behavioural Abstraction 217-21  
Bespoke Software 39  
Bell Daniel 29-30  
Binary Relations 117-18  
Binary Relational Store 118  
Black Boxes 198  
Bottom-Up Data Analysis 125  
Bottom-Up IS Planning 317  
Bounded Rationality 23  
*BPR See Business Process Re-Engineering*  
Bracketing Notation 132-3  
Breakdowns 353  
  
British Computer Society (BCS) 55, 374, 377, 378, 379  
Bureaucracy 16, 22  
Business Analysis 18, 59  
Business Analysis Methods 243  
Business Process 244, 369  
Business Process Re-Engineering (BPR) 25, 244-51, 369  
Business Uncertainty 282-3  
  
CAISE *See Computer Aided Information Systems Engineering*  
Cardinality 144  
Central Processing Unit (CPU) 37  
Change Management 245  
Checkland, Peter 252  
Class-Based Languages 78  
Classification 215  
Classes *See Object Classes*  
Classic Data Models 141  
Clean Rooms 285  
Client-Server Systems 41  
Coad and Yourdon (OO Method) 291-2  
Cohesion 200  
Command Language 226  
Communications Layer (Subsystem) 40  
Competitive Advantage 320  
Component CAISE 104  
Computer Aided Information Systems Engineering (CAISE) 95, 103-108, 179, 349  
Computer Supported Co-operative Work (CSCW) 26

- Conceptual Data Dictionary 179  
Conditions 72-73, 171-2  
Configuration Management 302-3  
Connection 200  
Context (For ISD) 1  
Control Structures 71-3, 171-3  
Cooperative Design 277  
Corporate Gearing 105-6  
Correspondence Failure 354  
Council of European Informatics Societies (CEPIS) 378  
Couple 200  
Coupling 199-200  
Covering Subclasses 215  
Create Table Statement 90  
Critical Success Factors 320-1  
CSCW *See Computer Supported Cooperative Work*  
Culture 7-8, 367-8  
Cultural Enquiry 254-5  
Customer Resource Life-Cycle 322-4
- Data 3, 8  
Data Abstraction 85  
Data Analysis 125, 212  
Data Control Language (DCL) 92-3  
Data Definition 87-8  
Data Definition Language (DDL) 90-1  
Data Dictionaries 179-87, 264  
Data Elements 181  
Data Flows 156, 157  
Data Flow Diagramming 155-68, 200-1, 263  
Data Independence 85-6  
Data Integration 84-5  
Data Integrity 85, 89, 100-2  
Data Layer (Subsystem) 40  
Data Manipulation 88-9  
Data Manipulation Language (DML) 91-2
- Data Model 42, 86, 141  
Data Modelling 141, 213, 265  
Data Protection 31  
Data Security 85  
Data Sharing 84  
Data Stores 156, 158  
Data Structure 181  
Data-Driven Methods 261  
Data-Oriented Program Design 205-10  
Database 83-4  
Database Administration 86, 265  
Database Design 141, *See also Normalisation, Entity-Relationship Diagramming*  
Database Management Systems (DBMS) 42, 83, 86  
Database Systems 83-94  
Decision-Making 10, 23, 25  
Decision Support Systems (DSS) 26, 41  
Decision Tables 173-5  
Decision Trees 175  
Declarations 73-4  
Degree *See Cardinality*  
De-Normalisation 136-7  
Dependency *See Determinancy*  
Designing by Doing 277  
Determinancy 133-4  
Determinancy Diagramming 134-6  
Development Methods *See Methods (Development)*  
Development Uncertainty 283  
Dialogue Design 225-6  
Dialogue Specifier 96  
Direct Manipulation Interface 228  
Discipline 351 *See Information Systems Discipline*  
Disjoint Subclasses 215  
Documents Flow 159

- Dynamic Systems Development Method (DSDM) 284, 286, 288
- Early Prototyping 234
- Effects of IS/IT 364-6
- Embedded SQL 97-8
- Empirics 9, 225
- Empowerment 276, 365
- Engineering 379-80
- Entity 141-2, 188, 218
- Entity Integrity 89
- Entity Life Histories 188-95, 218
- Entity Models 141-2, 184-5
- Entity-Relationship Diagramming 141-53
- Entity-Relationship Data Model 141
- ETHICS 277-9
- Events 189-90, 219
- Evaluation *See Information Systems Evaluation*
- Evolutionary Development 64
- Evolutionary Prototype 236
- Executive Information Systems (EIS) 26
- Expectation Failure 354
- Expert Systems 120-2
- External Entities 156-7, 158-9
- Factoring 202
- Factory Organisations 312
- Feasibility Report 61
- Feasibility Study 61, 266
- File Maintenance Anomalies 127-8
- File Transfer Protocol (FTP) 112
- Financial Management of the IS Service 50
- Fifth Generation Language 70
- First Normal Form (1NF) 129-30
- Fit *See Organisational Fit*
- Five Forces Model 321-2
- Fourth Generation Environments 95-102
- Fourth Generation Language (4GL) 42, 70, 95
- Forms 226
- Formal Information Systems 15-16
- Formative Evaluation 329
- Formatted Diagrams 173
- Frames 120
- Front-End CAISE 104
- Function Points 346-9
- Functional Determinacy 134
- Functional Languages 69
- Functionality 370
- Generalisation (Relationship) 145, 214, 215
- Goods Flow *See Materials Flow*
- Gopher 112
- Goronwy Galvanising Case Study 19, 43-5, 137-8, 150-2, 165-6, 176-7, 185-6, 192-3, 209-10, 230-1, 348-9
- Graphical User Interface (GUI) 41
- HASA Relationship 116-17
- Hardware 36-9
- Hawthorne Studies 22
- High-Level Languages 69-70
- Human Activity System 13-15, 24, 32, 252
- Hybrid Manager 18-19, 376
- Hypermedia 110-11
- Hypermedia Information Systems 109-15
- Hypertext 109-10
- Hypertext Markup Language (HTML) 112
- Incremental Prototype 236, 285
- Inference 119-20

- Individual Gearing 105  
Informal Information Systems 15-17  
Information 1-12, 30  
Information Architecture 316  
Information Centre 48-9  
Information Economy 32-4  
Information Economics 334-5  
Information Engineering 56-7, 264-6,  
  318  
Information Highways 33  
Information Management 307-8  
Information Retrieval 86  
Information Society 29-35  
Information Strategy 316  
Information Systems 13-20  
Information Systems Assessment  
  319-20  
Information Systems Architecture  
  316-17  
Information Systems Benefits 331  
Information Systems Costs 330-1  
Information Systems Engineers 24  
Information Systems Environment 385  
Information Systems Development  
  *See Information Systems Engineering*  
Information Systems Discipline 3,  
  383-91  
Information Systems Engineering  
  53-66, 385  
Information Systems Evaluation 47,  
  329-36, 353  
Information Systems Failure 298-9,  
  352-61  
Information Systems Maintenance 46,  
  59  
Information Systems Management 46,  
  59, 306-15, 385  
Information Systems Outsourcing  
  337-42  
Information Systems Operation 46  
Information Systems Planning 46, 59,  
  106, 316-28  
Information Systems Services Function  
  46-52, 308-10  
Information Systems Services  
  Operational Plans 327  
Information Systems Services Strategic  
  Plan 326  
Information Systems Specialist 18-19  
Information Systems Strategy 316, 368  
Information Technology 36-45, 385  
Information Technology Architecture  
  316-17  
Information Technology Strategy 316  
Information Technology Systems  
  15-17, 24, 26, 30, 36, 40  
Information Value 331-2  
Infrastructure 369-70  
Inheritance 77, 214  
Input/Output 74  
Institute of Electronic Engineers (IEE)  
  374  
Integrated Project Support  
  Environments (IPSE) 106  
Integrative Methods 261  
Interface Layer (Subsystem) 40  
Integrated CAISE 104  
Inter-Organisational Information  
  Systems 33-4, 367  
Interaction (between IS and  
  Organisations) *See Organisational Fit*  
Interaction Failure 354  
Intervention Analysis 254-5  
Intranet 112  
Internet 34, 112-13  
ISA Relationship 116  
IS/IT Interaction Model 362  
IT Assimilation Model 308-9

- Jackson Structured Programming (JSP) 205  
Java 78, 114  
Job Design 276  
Join 89  
Joint Application Design (JAD) 275, 284-5
- Knowledge 3, 122  
Knowledge Based Systems (KBS) 116-23  
Knowledge Elicitation 122  
Knowledge Engineering 57  
Knowledge Management 57  
Knowledge Representation 122-3  
Knowledge Workers 57
- Late Prototyping 235  
Legacy Systems 370  
Levels of Maturity 310-11  
Levelling 160  
Life-Cycle of Information Systems Development 60-5, 371  
Local Area Network (LAN) 38  
Location of the IS Service 49-50  
Logic Driven Stream 253-4  
Logic Programming Languages 69  
Logical Systems 165  
Logical Data Dictionary 179, 180  
Loops 72-3, 172-3  
Low-Technology Prototyping 277
- Machine Code 69  
Management 297  
Management Information Systems (MIS) 26, 41  
Materials Flow 159  
Meaning Triangle 5-6  
Message 76  
Meta-CAISE 104
- Methodologies *See Methods (Development)*  
Methods (Development) 59, 242, 258, 259-61, 284, 349  
Methods (Object-Oriented) 76, 217, 220-1  
Metrics *See Software Metrics*  
Menus 226  
Middle Prototyping 235  
Millenium Bug 370  
Module 200  
Multimedia 109-15  
Multimedia Interface 228  
Multiple Inheritance 214  
Mutual Learning 277  
Multiview 256  
Mumford, Enid 277
- Natural Language Interface 227  
Network Computers 113  
New Technology Group 309-400  
Non-Functional Determinancy 134  
Non-Loss Decomposition 128-33  
Norm 7  
Normal Forms 128-32  
Normalisation 127-40  
Normalisation Oath 133
- Object Life Histories 218-20  
Objects 76, 213  
Object Classes 77, 217  
Object Modelling 213, 221-2  
Object Modelling Technique (OMT) 293-294  
Object Pascal 78-81  
Object-Based Languages 78  
Object-Orientation 212, 349  
Object-Oriented Analysis and Design 212-23  
Object-Oriented Methods 291-6

- Object-Oriented Programming 76, 197  
Object-Oriented Programming  
  Languages 69, 76-82  
On-Line Systems 41  
Operating Systems 39  
Optionality *See Participation*  
Oracle 96-7  
Organisation 21-8  
Organisation of the IS Service 47-9  
Organisation Theory 21  
Organisational Fit 362-71, 372  
Organisational Learning 366  
Organisational Strategy 368  
Organisational Structure 367  
Organisations as Constructions 23, 26  
Organisations as Environments 22, 25  
Organisations as Information  
  Processors 23, 25-6  
Organisations as Networks 22, 24-5  
Organisations as Productions 23, 26  
Organisations as Structures 21-2, 24  
Other Techniques 196  
Outsourcing *See Information Systems*  
  *Outsourcing*  
Overriding 77  
Overlapping Subclasses 215  
  
Packaged Software 39  
Paradigms 55  
Partial Sub-Classes 215  
Participation 145, 261  
Participatory Design 25, 273-81  
Partitioning 198  
PARTOF Relationship 117  
Passive Data Dictionary 179  
Payback Period 333  
PC Select 107  
Physical Data Dictionary 95, 179  
Physical Systems 164-5  
Piecemeal Development 83  
PL/SQL 99-100  
Political Systems Analysis 255  
Polymorphism 77-8  
Power 7-8  
Pragmatics 8, 225  
Primitive Data Models 141  
Projects In Controlled Environments  
  (PRINCE) 303-4, 376  
Procedures 74-5  
Processes 156, 157, 188  
Process Analysis 154, 212  
Process Descriptions 169-78  
Process Failure 354  
Process Mode 142  
Process-Driven Methods 261  
Productivity Paradox 363-64  
Profession 374-6  
Professiona l 374  
Professional Development 378  
Professionalisation 374-82  
Professionalism 60, 353, 374  
Programme Evaluation Review  
  Technique (PERT) 299  
Programming Languages 39, 69  
Project 89  
Project Abandonment 357, 365  
Project Estimation 299-300  
Project Gearing 105  
Project Organisation 300-1  
Project Planning 299  
Project Management 46, 59, 298-305,  
  349  
Project Selection 61  
Prototypes 232-3  
Prototyping 232-41  
Pseudo-Code 169  
  
Quality Assurance 47, 343-350  
Query Language 96

- Rapid Application Development (RAD) 282-90  
Rapid Development Tools 285  
Rapid Prototyping 63  
Referent 5  
Referential Integrity 89  
Relational Algebra 88-9  
Relational Data Model 86-9  
Relationship 142  
Report Generator 96  
Requirements Analysis 266  
Requirements Specification 61-2, 266  
Research (Information Systems Development) 388-9  
Research Methods 389-90  
Restrict 89  
Return on Investment 332-3  
Rich Pictures 255  
Risk 353  
Role 7-8  
Root Definition 253-4  
Rules 121-2  
Rules Subsystem (Layer) 40
- Scandinavian Tradition (Participatory Design) 274  
Scientific Management 21-2  
Screen Painter 95  
Second Normal Form (2NF) 130  
Semantic Data Models 116-17, 141  
Semantic Modelling 142-3  
Semantic Nets 120  
Semantics 8, 225  
Semiology *See Semiotics*  
Semiotics 5-9, 225  
Sequences 71-2, 171  
Semi-Profession 377  
Shrink-Wrapped Software 39  
Sign 5-6  
Sign System 5
- Simon, Herbert 23  
Social Group 7  
Social Systems 6-8  
Social Systems Analysis 255, 261  
Socio-Technical Systems Thinking 25, 274-5, 278  
Soft Systems Analysis 18, 24-5  
Soft Systems Methodology 252-7  
Software 39-42  
Software Engineering 24, 55-6, 58  
Software Quality Assurance  
*See Quality Assurance*  
Software Metrics 344  
Software Problem 53-4  
SOMA 294-5  
Specialisation 214-15  
Spiral Model 64-5  
Summative Evaluation 329  
Stakeholders 273  
State Transition Diagrams (STDs) 193-4, 219-20  
Statements 71  
States 219  
Status 7-8  
Strategic Information Systems 25, 325-6, 366-7, 368  
Strategic Organisations 312  
Structural Abstraction 213-17  
Structure Charts 200-5, 264  
Structure Diagrams 205-10  
Structured English 169-73  
Structured Query Language (SQL) 89-93  
Structured Methods 259-72  
Structured Program Design 197-211  
Structured Programming Languages 69-75  
Structured Prototyping 63  
Structured Systems Analysis (STRADIS) 262-4

- Structured Systems Analysis and Design Method (SSADM) 188, 224, 225, 259, 261, 266-9, 284, 304, 376
- Structured Walkthroughs 345-6
- Support Organisations 313
- Symbol 5
- Syntactics 9, 225
- System 13, 24
- Systems Analysis 17-18, 58, 245
- Systems Construction 58
- Systems Delivery 59
- Systems Design 58
- Tacit Knowledge 276
- Taylor, Frederick 21
- Technical Information Systems  
*See Information Technology Systems*
- Techniques 59, 124
- Termination Failure 355
- Ternary Methods 291
- Theory 60
- Third Normal Form (3NF) 131
- Thin Clients 113-14
- Third Generation Language (3GL) 41
- Throwaway Prototype 235-36
- Timeboxing 285
- Tools 59, 67
- Top-Down Data Analysis 125
- Top-Down IS Planning 317
- Transaction Analysis 201
- Transaction Layer (Subsystem) 40
- Transaction Processing 86
- Transaction Processing System 42
- Transactional Information 30-1
- Transform Analysis 201-3
- Transmission Control Protocol/Internet Protocol (TCP/IP) 114
- Triggers 101
- Turing Machines 36-7
- Turnaround Organisations 312
- UK Academy of Information Systems (UKAIS) 386-7
- UK Tradition (Participatory Design) 274-5
- Unary Methods 291
- US Tradition (Participatory Design) 275-6
- Usability 370
- Usability Engineering 229-30
- User Interface 224
- User Interface Design 224-31
- User Participation 276
- Utility 370
- Value Chain 324-5
- Virtual Organisations 32-3
- Virtual Reality Interface 228-9
- Waterfall Model 60-2, 232
- Work Analysis 276
- World-Wide Web (WWW) 112
- Wide Area Information Servers (WAIS) 112
- Wide Area Network (WAN) 38