INFORMATION SYSTEMS DEVELOPMENT

Paul Beynon-Davies

Department of Computer Studies
Polytechnic of Wales

M
MACMILLAN

First published 1989

Published by
MACMILLAN EDUCATION LTD
Houndmills, Basingstoke, Hampshire RG21 2XS
and London
Companies and representatives
throughout the world

Laserset by
Ponting–Green Publishing Services, London

Printed in Great Britain by
Billing & Sons Ltd, Worcester

British Library Cataloguing in Publication Data

Beynon-Davies, Paul
Information Systems Development.–(Macmillan computer science series).
1. Information systems
I. Title
001.5

ISBN 0-333-48034-1
ISBN 0-333-48035-X Pbk

continued overleaf

For Gillian

Preface

Most of the conventional material on information systems development is deterministic in the sense that it imposes some prior framework on the use of the techniques available. The present book is designed to be as flexible or non-deterministic as possible. It provides a series of relatively discrete, self-contained sections or notebooks on important topics in the field. It therefore largely leaves it up to readers of the book to impose their own determinism.

This mode of presentation will, I feel sure, be of greater benefit to persons running courses in systems development than the traditional approach tied to one methodology. It will also, I feel sure, more clearly suit the purposes of computer professionals with a desire to know more about particular topic-areas. This applies not only to those persons who perhaps do not wish to work within the confines of any one methodology, but also to those looking for a more detailed treatment of specific subject-areas that are perhaps not well covered in the documentation of an existing methodology.

The book is divided into four major parts. The first part provides an initial discussion of some of the background issues involved in the need for, and development of, a systematic discipline of information systems development. We discuss the software problem, the project life-cycle, the development of structured analysis, design and programming, and relational database systems.

The second part discusses the major techniques of contemporary systems development. It describes a tool-kit out of which most of the contemporary methodologies have been built: data-flow diagramming, data dictionaries, normalisation, entity–relationship diagramming, entity life-histories, structured walkthroughs and structured design. Each of these sections contains three parts: a detailed discussion of the technique, a small example, and a set of problems for further study.

The third part of the book discusses a number of tools designed to aid or enhance the software development process. We cover the present generation of computer-aided software engineering (CASE) tools, most notably integrated project support environments and fourth-generation languages. We also highlight the role that knowledge-based systems may have in the development environments of the future.

In the final part we discuss issues relating to the organisation of systems development. That is, how some of the tools and techniques previously covered take their place within a general methodology or 'philosophy' of systems development.

Information systems development has become almost exclusively associated in Britain with one particular government standard methodology which goes under the title of structured systems analysis and design method (SSADM). A recent paper has however criticised the trend towards developing competing methodologies – a trend that we might label as 'methodolotary'. Benyon and Skidmore (1987) believe that the methodolotary trend is hampering progress towards successful systems analysis. They consider it is unlikely that a single methodology can prescribe how to tackle the great variety of tasks and situations experienced by the systems analyst.

There are, however, alternatives to the 'waterfall' model of systems development as characterised by frameworks such as SSADM. In the fourth part of the book, therefore, we also consider rapid prototyping and James Martin's suggestion for an encompassing discipline of information engineering, based around the explicit use of CASE tools.

The concluding chapter considers some of the possibilities for the future of information systems development. We organise our discussion in terms of one central premise, that information systems development is primarily a case of conceptual modelling. This premise encourages us to discuss three areas of modern computing that are contributing new tools, techniques and philosophies to this endeavour: artificial intelligence, database work and programming languages.

Contents

1 *Introduction*

The software problem

Over the last 20 years hardware performance 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 whole series of smaller problems:

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

Software engineering

Many solutions to these problems have been proposed. One of the most influential, probably because it is the most comprehensive, has been to try to cast software development as an engineering exercise.

> Software engineering is the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate and maintain them. (Boehm, 1976)

This rather general definition captures the all-encompassing nature of the term software engineering. Software engineering is an attempt to found the entire project life-cycle in a systematic approach to software development.

The above definition, is however, somewhat vague. It contains at least one term, namely 'scientific knowledge', which is subject to a number of different interpretations. A more practical definition of software engineering might therefore be:

> The systematic application of an appropriate set of techniques to the whole process of software development.

This definition concisely presents the three 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, that is, over the entire life-cycle of a project.

One of the important themes of software engineering is its emphasis on a clear *structure* for software development. This is usually contrasted with the traditional *ad hoc* approach to software development, an approach that is seen as being the major contributory factor to the software problem.

Structured programming, design and analysis

In response to a dissatisfaction with the traditional *ad hoc* approach to software development, three more rigorous areas of computing have been developed (King, 1984):

1. Structured programming: the attempt to construct a disciplined programming methodology based upon firm notions as to an appropriate syntax for procedural programming languages. This was the emphasis of the late 1960s and early 1970s in the computing world
2. Structured design: the discipline of building hierarchical systems of modular software. This was the emphasis of the mid to late 1970s (Yourdon and Constantine, 1979)
3. Structured analysis: an attempt to separate the logical from the physical description of information systems. The emphasis of the early 1980s (Weinberg, 1980).

These three areas, traditionally seen as sub-disciplines of software engineering, have now become accepted practice within conventional systems analysis and design. They correspond roughly to the three major stages of the software development process: analysis, design and implementation.

This book concentrates on systems analysis and design. Because of the interdependent nature of software development, however, we will necessarily have to touch upon aspects outside this fuzzy domain. For instance, we will discuss the context of information as an organisational resource, and some of the tenets underlying the construction of well-designed programs.

The software development process

Structured software development is usually seen 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 (see figure 1.1).

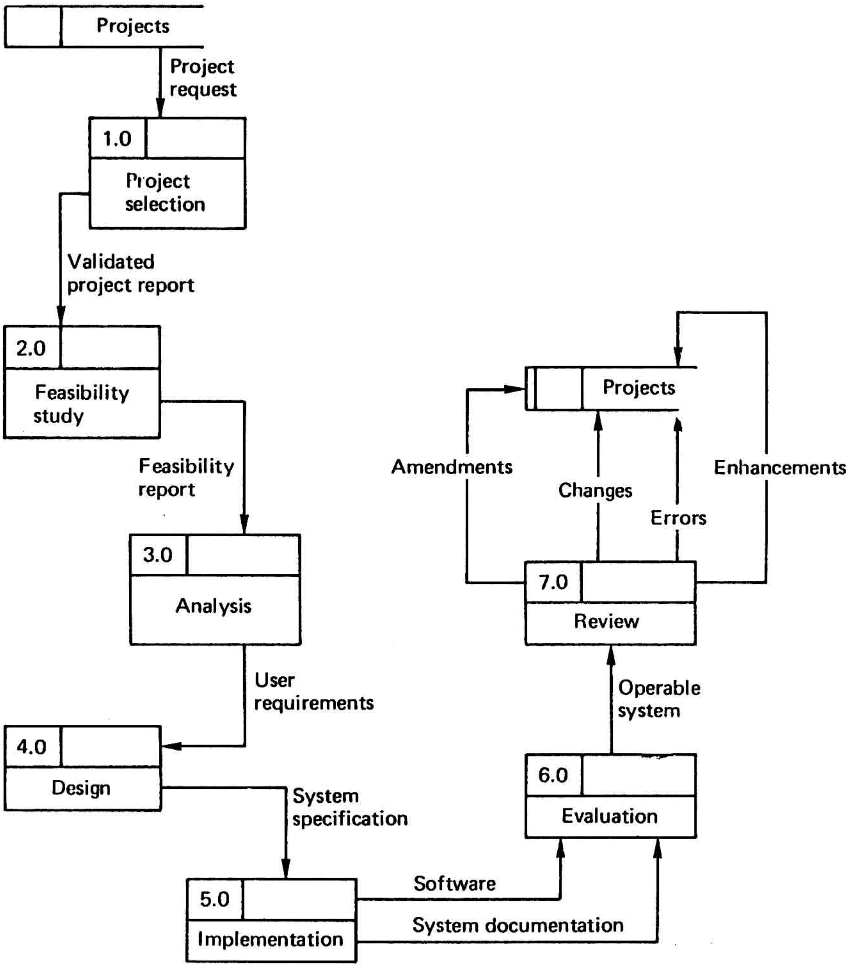

Figure 1.1

In large organisations, there are usually a large number of requests for applications systems. To handle such a diversity of applications formally, most 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.

The primary mechanism of the project selection process is the project selection committee. The project selection 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 is presented to the relevant users who offer their opinions. These opinions may then be fed back into the feasibility study which produces a revised report, and so on.

In a sense, the project selection process and the feasibility study are two 'filters' at the beginning of the development life-cycle. Project selection is a coarse-grained filter. Its objective is to reject those projects which are clearly unsuitable. The feasibility study is a fine-grained filter. Its objective is to reject projects on more detailed grounds (see figure 1.2).

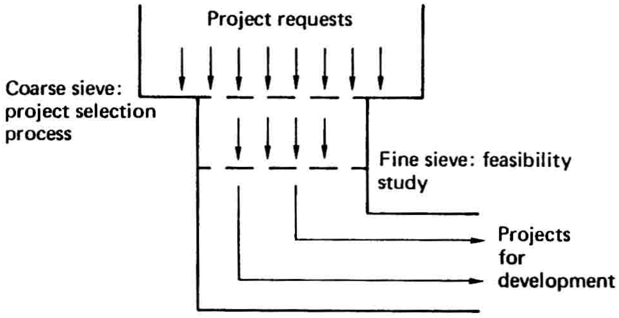
Project requests
Coarse sieve: project selection process
Fine sieve: feasibility study
Projects for development

Figure 1.2

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

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

Once the completed design is ready, it is used to produce 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 which must fully describe the 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 requirements document to check that it has achieved its aims.

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

Software development is often referred to as a cycle for two major reasons:

1. It is subject to a whole series of small iterations surrounding the user reviews, both within phases and between phases
2. A system is seldom 100 per cent complete. Users will continually want errors in the system corrected, parts of the system changed, or major extensions added. Software maintenance is therefore a critical factor which should influence all aspects of the development process.

This 'waterfall', linear, or 'loopy linear' model of software development is the one most often adhered to within the information systems development community. Recently, however, a more iterative or incremental method of systems development has been proposed. This participative approach, which is usually called prototyping, or sometimes rapid prototyping, is considered in Chapter 16 (Dearnley and Mayhew, 1983).

The major themes

Contemporary information systems development can thus be characterised in terms of five major themes:

1. The emphasis on structure. That is, as we have already discussed, the software development process is made up of a series of well-defined procedures which act on well-defined inputs to produce well-defined outputs. It is therefore a framework which constrains the software development process in directions that have been shown to produce better systems. For instance, it has been shown that clear and well-considered structure lead to more maintainable systems (Yourdon, 79)
2. The emphasis on data. That systems analysis and design should

concentrate on data rather than process. This means that the prime area of concern is the data needed to support organisational behaviour. The procedures undertaken in any enterprise are seen to be by-products of, or reliant upon, organisational data. This emphasis encourages a global integrated view of organisational data

3. The emphasis on user and peer group participation. All stages of the software development process are subject to some form of user and peer group review. This is particularly important in the analysis and design stages. User involvement has been proved to produce better systems in terms of a closer match between user requirements and finalised systems. Peer group reviews of the products of the development process are a necessary validation mechanism (Yourdon, 78)
4. The emphasis on logical modelling. That is, the importance of building an initial model of a system which is not tied to any specific implementation plan. In other words, structured systems analysis and design enforces a necessary 'fire-wall' between the analysis and design stages in development. Analysis involves primarily producing a documentation of user requirements. Design involves turning such requirements into an implementable plan
5. The emphasis on graphical presentation. Most of the techniques used in structured systems analysis and design are graphical in nature. This results from the view that one of the major problems of traditional software development is the problem of communication. A problem of communication exists between the systems analyst and the end-user, between the systems analyst and the programmer, and between the systems analyst and management. Diagrams are a proven method of enhancing communication; they are easier to understand by everyone involved in the development of software. They are also easier to manipulate, and represent information far more concisely than the written page (Martin and McClare 1985).

Data-directed development

Structured systems analysis and design constitutes data-directed development. Data-directed analysis and design emphasises two opposing but complementary views of data: the dynamic view, and the static view.

1. The dynamic view emphasises the importance of data flow. That is, how data moves through the system, and is transformed by the various processes making up the information system.
2. The static view emphasises data structure. That is, how data is or should be organised in a system in order to support organisational processes.

Two major techniques have been used to support these views: data flow diagramming and entity–relationship (E–R) diagramming. These two methods form the core of our discussion in the techniques section.

The other techniques to be discussed, such as data dictionaries, entity life histories and structure charts, are really means for enhancing this dynamic duo. For instance:

1. Data dictionaries are a method of enhancing and extending the documentation of data flow.
2. Entity life histories are a method for connecting up the activities or events represented on a set of data-flow diagrams with the entities on a set of entity–relationship diagrams.
3. Structure charts are fundamentally a means for transforming a data-flow diagram into something closer to a programmable structure – a hierarchically organised set of program modules.

Databases and database management systems

Given the emphasis on data in modern information systems development it is not surprising that perhaps the greatest influence on the approach in recent years has been the development of databases and database management systems. In particular, one database model has laid claim to supremacy, at least in methodological terms. This is the relational database model to be discussed in some detail in chapter 2. Without a basic understanding of this model, the chapters on normalisation, E–R diagramming, and indeed the whole question of information resource management cannot be properly understood.

Insufficient improvements from structured techniques

As has been mentioned, the main hope for improving productivity over the last two decades has been the application of structured techniques. Recently however, it has been claimed that few installations have benefited from an increase in productivity of greater than 50 per cent from the application of structured techniques alone (Martin, 1984). This has led many people to direct attention to a whole range of other issues which are considered to be further sources of productivity enhancement.

In this book, we shall address a number of such issues. First, we shall address the way in which relational databases and relational database management systems (RDBMS) have not only influenced the application of structured techniques, but are influencing the whole future development of computing. Using RDBMSs as a core we shall then address a whole range of environmental issues that are moulding the software development process. For instance, we shall consider the use of prototyping, fourth-generation languages (4GLs), integrated project support environments and knowledge based systems. Each of these topics will be cross-referenced so that in the concluding chapter we can make some sensible predictions as to the future of the activities of systems analysis and design.

Information systems engineering

At the start of this chapter we described how software engineering has been proposed as one of the most comprehensive solutions to the software problem. In recent years, a yet more sophisticated solution called information engineering or information systems engineering has been proposed: (see for example Martin, 1984):

> 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 which 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. The primary focus of software engineering is the logic that is used in computerised processes (Martin, 1984).

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 an optimal way to represent it logically.
4. That although data are relatively stable, the processes that use such data change fast and frequently.
5. That because the basic data types are stable, whereas processes tend to change, data-oriented techniques succeed if correctly applied where process-oriented techniques have previously failed.

Although Martin tends to contrast information engineering with software engineering, the author believes that it is more appropriate to cast information engineering as an overarching discipline which encompasses not only traditional software and systems engineering issues, but a global concern with the management of information in organisations. This is more realistic