COMIC

ESPRIT BASIC RESEARCH PROJECT 6225

Informing CSCW System Requirements

Abstract

The thrust of this Strand of the COMIC Project is mainly methodological in investigating aspects of the relationship between the social analysis of work settings and the systems development context. It has been assumed throughout the work that real world CSCW systems will be large scale and, thus, need be produced by appropriate software engineering processes. In addition, it has also recognised that CSCW raises special problems regarding the effective analysis of the social organisation of work settings and that among the more promising methods in this regard, namely ethnography, does not sit easily with many of the current methods of requirements elicitation and system development.

Document ID	D2.1
Status	Accepted
Type	Deliverable
Version	2.0
Date	October 6, 1993
Editors	Lancaster University and Manchester University
Task	2.1

Project coordinator:
Tom Rodden
Computing Department
University of Lancaster
Lancaster LA1 4YR
United Kingdom
Phone: (+44) 524 593 823
Fax: (+44) 524 593 608
Email: tom@comp.lancs.ac.uk

The COMIC project comprises the following institutions:
Gesellschaft für Mathematik und Datenverarbeitung (GMD), Bonn, Germany
Risø National Laboratory, Roskilde, Denmark
Royal Institute of Technology (KTH), Stockholm, Sweden
Swedish Institute for Computer Science (SICS), Stockholm, Sweden
Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
University of Amsterdam, Amsterdam, The Netherlands
University of Lancaster, Lancaster, United Kingdom (Coordinating Partner)
University of Manchester, Manchester, United Kingdom
University of Milano, Milano, Italy
University of Nottingham, Nottingham, United Kingdom
University of Oulu, Oulu, Finland

Contributors to this report:

Liam Bannon
Department of Computer Science and Information Systems
University of Limerick
Limerick. Ireland
Phone: (+353) 61 333644
Fax: (+353) 61 330876
Email: bannon@ul.ie

John Bowers
Department of Psychology
University of Manchester
Manchester M13 9PL
United Kingdom
Phone: (+44) 275 2599
Fax: (+44) 275 2514
Email: bowers@psy.man.ac.uk

Peter Carstensen
Risø National Laboratory
Cognitive Systems Group
P. O. Box 49
DK-4000 Roskilde. Denmark
Phone: (+45) 42 37 12 12 ext. 555
Fax: (+45) 46 75 51 70
Email: phc@risoe.dk

John A. Hughes
Department of Sociology
Lancaster University
Lancaster LA1 4YL
United Kingdom
Phone: (+44) 524 594 174
Fax: (+44) 524 844 788
Email: J.A.Hughes@lancaster.ac.uk

Kari Kuutti
Department of Information
University of Oulu
Linnanmaa
SF-90570 Oulu. Finland
Phone: (+358) 81 553 1904
Fax: (+358) 81 553 1890
Email: Kuutti@rieska.oulu.fi

James Pycock
Department of Psychology
University of Manchester
Manchester M13 9PL
United Kingdom
Phone: (+44) 275 2556
Fax: (+44) 275 2514
Email: pycock@psy.man.ac.uk

Tom Rodden
Department of Computing
Lancaster University
Lancaster LA1 4YR
United Kingdom
Phone: (+44) 524 593 823
Fax: (+44) 524 593 608
Email: tom@comp.lancs.ac.uk

Kjeld Schmidt
Risø National Laboratory
Cognitive Systems Group
P. O. Box 49
DK-4000 Roskilde. Denmark
Phone: (+45) 46 77 51 46
Fax: (+45) 46 75 51 70
Email: kschmidt@risoe.dk

Dan Shapiro
Department of Sociology
Lancaster University
Lancaster LA1 4YL
United Kingdom
Phone: (+44) 524 594 175
Fax: (+44) 524 844 788
Email: D.Shapiro@lancaster.ac.uk

Wes Sharrock
Faculty of Economic and Social Studies
University of Manchester
Manchester M13 9PL
United Kingdom
Phone: (+44) 275 2510
Fax: (+44) 275 2514
Email: msrssws@cms.mcc.ac.uk

Stephen Viller
Department of Computing
Lancaster University
Lancaster LA1 4YR
United Kingdom
Phone: (+44) 524 65201 ext. 3041
Fax: (+44) 524 593 608
Email: viller@comp.lancs.ac.uk

ISBN 0-901800-29-5
Lancaster University, 1993
This report is available via anonymous FTP from ftp.comp.lancs.ac.uk.

Table of Contents

Preface

The thrust of this Strand of the COMIC Project is mainly methodological in investigating aspects of the relationship between the social analysis of work settings and the systems development context. It has been assumed throughout the work that real world CSCW systems will be large scale and, thus, need be produced by appropriate software engineering processes. In addition, it has also been recognised that CSCW raises special problems regarding the effective analysis of the social organisation of work settings and that among the more promising methods in this regard, namely ethnography, does not sit easily with many of the current methods of requirements elicitation and system development.

The objectives of the Strand are as follows:

- the development of techniques and guide-lines to support cross-disciplinary working in CSCW systems development.
- evaluate existing requirements and development methods to see how and what way they might incorporate data derived from ethnographic studies of work settings.
- investigate tools to support the effective translation of ethnographic analyses in CSCW system development.

CSCW, Requirements and Development

It was felt that the failure of many previous system designs to adequately reflect the 'real world' environments in which they operated, together with changing ideas about the nature of work, and the role of new technologies within it, dictated the need for innovative reconceptualisations and techniques for the design of future CSCW systems. Not only will this require an even greater sensitivity to users' needs, it will also require a corresponding sensitivity to the socially organised character of the setting for which the system is being designed. To date, designers and developers have only had partial knowledge of the relevant domains through a set of requirements provided at the start of the design process with the result that, very often, there is a mismatch between the system and the demands made upon it in the 'real world' work setting.[1] Thus, the objective of this Strand is to investigate the interaction between systems development and the work setting including an examination of the role of empirical studies in generating systems' requirements and subsequent system development.

Although this is the main focus of this Strand within the COMIC project it parallels and complements the research within other strands, particularly Strand 1 and Strand 3. Strand 1 is concerned to investigate the organisational context of interaction with respect to designing appropriate system support. Strand 3 concentrates on an examination of the languages and notations that can be used

---

1 A recent dramatic illustration of these kinds of problems was the failure of the Computer Aided Despatch system developed for the London Ambulance Service. See Page *et al* (1993).

within CSCW design to convey system requirements to developers. Thus, these strands represent different emphases on some of the important issues in CSCW design.

The themes of this Deliverable can be stated simply:

- how to determine what to design and build. This is an issue of what is commonly called 'requirements capture' or 'requirements elicitation'
- how can this be effectively communicated to designers. This is a matter of 'requirements specification'.
- how to organise the system development process.

Clearly, these themes are closely related and not all their ramifications will be dealt with in equal depth in what follows.

There are a number of preliminary points to note:

- *Terminological Issues*

  There are problems of terminology regarding 'requirements capture', 'requirements elicitation', 'requirements specification', and so on, which, in many respects reflect the arguments advanced about the process. There are also issues concerning what constitutes a requirement as well as over the relationship to other means of expressing the needs of users, such as the use of a wide range of prototypes, the development of acceptance criteria and the development of process oriented contracts.

  Rather than becoming directly embroiled in a syntactical discussion of what constitutes a requirement, we wish to examine more generally the process by which requirements emerge rather than consider in isolation the form in which requirements are expressed. For our part, the requirements process refers to determining the needs a system should meet irrespective of whether this is seen as a staged process or one which runs concurrently with the design itself.

- *Relating requirements to development*

  By their very nature requirements are intimately bound up with the development process and an understanding of the nature of design and development is central to a consideration of requirements. In much of the discussion we use the terms 'design' and 'designer' to indicate the process of determining what to build and deciding how to build it. We intend no implication, however, that this is a specific task that is the responsibility of a single person, be it the software engineer or the cognitive scientist, or whatever. In some perspectives, of course, there is a more specific nomination of what this process consists of and who performs it. As necessary we will draw these distinctions.

The Structure of the Deliverable

This Deliverable has been produced as a single integrated document because it was felt that many of the issues that needed to be addressed were relatively new, and

also because there was much needed baseline work which needed to be done in order to inform the research in subsequent years, both in this and in other Strands. In particular, the need to clarify and integrate a number of issues was a principal effort in the development of this Deliverable.

A number of researchers have made contributions and the Deliverable draws on the many Working Papers submitted over the course of the year.

A summary of the various sections of the Deliverable is as follows:

Part I: Panaceas and Rhetorics in System Design

This is an extensive discussion of some of the arguments about the role of requirements and development methods that have arisen not only in CSCW, but software engineering more generally. Many of these arise from the interest that social science, particularly sociology, has recently begun to take in CSCW. In particular, it reviews, as an example of these debates, some of the arguments that have arisen over the utility of the Waterfall Model of systems design and development, including those which tend to see this as symptomatic of critiques of the organisation of work in modern society.

The argument that is developed in this Part is that it is important to recognise the practicalities of system design and development, many of which are to do with the practicalities of engineering the design and development process as well as informing it with well-grounded analysis of the social setting in which the system will serve.

Part II: Techniques to Capture System Requirements

This is an extensive discussion of most of the extant methods of system design and development. These have been discussed in terms of the implied models of development, user-oriented approaches and system oriented approaches. The purpose of this discussion and evaluation is to provide a methodological context for CSCW system development.

The Part concludes that a common deficiency of many methods is that they employ *a priori* conceptions of the social organisation of work, often dictated by the necessities of system development.

Part III: A Social Point of View for Design

The objective of this Part is to explicate the basis of a social point of view on design, the promise of ethnographic methods and some of the problems that need to be addressed if the method is to be used extensively in CSCW design. It also illustrates this approach by drawing on two on-going studies of the design process, both of which and in different ways are intended to inform the design and development of DNP, a tool to support CSCW requirements and design.

Part IV: Requirement Methods and CSCW Systems Development

This concluding Part draws out some implications for CSCW design from the previous discussions. It places the development of CSCW system requirements within the general context of systems design and development, reviews the issues surrounding those techniques which have relevance for the

construction of CSCW systems, and highlights the key issues which need to be addressed in the future work of the Strand. The argument is that design is a matter of servicing inescapable tensions or choices that are involved in system development.

It concludes with a summary of the achievements of the Strand's first year of work along with a statement of future research.

Acknowledgements

This deliverable has involved significant contributions from a number of people across the project and has also drawn upon different working documents from a number of strands. Our thanks and gratitude are extended to all these contributors. The final formatting of the document was done by Kjeld Schmidt at RISØ.

Introduction: Design, Requirements and CSCW

Over the last decade, CSCW has emerged as an identifiable research area that focuses on the design of computer systems to support cooperative activities. The term CSCW was coined by Grief and Cashman who, in 1984, organised a workshop among a small but varied group of researchers in OIS (Office Information Systems), Hypertext, CMC (Computer Mediated Communication), to discuss the development of computer systems to support ensembles of people in their work activities. It has now grown into a research endeavour exploring a range of issues concerning the support of cooperative work arrangements via various forms of information technology. It has succeeded in attracting a wide mix of disciplines which include, along with computing and software engineering, OIS, CASE (Computer Aided Software Engineering), CIM (Computer Aided Manufacture), CAD (Computer Aided Design), and Participative Design, the additions to cognitive science, namely, sociology and anthropology.

Whatever the definition of CSCW, and there is still some dispute over this (see, for example, Greif, 1988; Bannon and Schmidt, 1989; Wilson, 1991; Schmidt and Bannon, 1992; Grudin, 1991; Bowers, 1991), because of its focus on the cooperative ensemble, it is open to a wider mix of disciplines than traditional HCI, and includes some branches of sociology, anthropology, cognitive science, social psychology, among others, as well as new methods for informing the formulation of requirements, such as ethnography and Participative Design. Howard (1988), in a panel discussion at one of the earlier conferences on CSCW, discerned two rough groupings within CSCW; the 'strict' and the 'loose' constructionists. The former are those whose focus is on the design and development of computer systems to support group work and who are mainly interested in the construction of tools and systems. They tend to see CSCW as a possible leverage for creating novel applications. The latter consists of a heterogeneous collection of people, some of whom are drawn to CSCW because of a dissatisfaction with the current use of technology in work, and others because they see the opportunity to have a voice in the design of computer systems where they did not have one before. Yet others seek to make the design of computer systems more democratic and, accordingly, see cooperation, both in work and design, as having a positive value that goes beyond the more instrumental concerns of system design and development.[2]

Such a mix is a symptom of a number of trends and concerns which have served to shape the development of CSCW and include long term changes in economic life and the nature of work, developments in computer technology and, as a consequence, new problems for the field of human-computer interaction.

---

2 On this latter point see Greenbaum and Kyng (1991), for example.

Changes in the Nature of Work

In the literature on the sociology of work, management and organisations there has recently been an extensive discussion of emerging new forms of the organisation of work which are seen as a response to major structural changes in contemporary advanced industrial societies. Among the changes which have been identified include a weakening of the welfare state, an increasing dependence between the state and capital, the breakdown of the mass market and the need for new forms of production, changes in consumption patterns and life styles, major technological innovations, the globalisation of firms, and so on.[3] Although there is much debate about the precise nature and extent of these changes, it has also resulted in debate and discussion about their implications for the organisation of work itself.

During the latter part of the 1980s and the early 1990s one of the major themes which has been addressed are ways of inculcating more active participation and flexibility among the work force which has, for management particularly, raised the need for a much closer acquaintance with the work process than is typical of more dominating forms of work organisation. The argument here is that developed countries are moving towards a 'post-Fordist' pattern of work organisation in place of the more traditional methods of industrial production archetypically represented by the car assembly line as the standardised system of production for mass markets, emphasising low unit costs and a strict division of vertical and horizontal divisions of labour.

The central features of this shift are the need for flexibility and intensive rather than extensive growth requiring a more efficient use of material and human resources. The implication is that workers need to become more skilled, more aware of the whole process of production, more capable of taking part in planning and design, quality control, and error retrieval: tasks which traditionally belonged to management. Thus, on this view, work organisation has to be appropriately shaped in order to reap the benefits of this new atmosphere of cooperation and has profound implications for training, commitments, rewards, and industrial relations more generally.

Information technology is seen as a key element in these changes both as an enabler of flexibility and as one of the more effective ways in which work reorganisation is achieved, though not always consciously and not always with satisfactory effects. To survive these transformations in work, IT offers the chance of support through advanced information systems that can facilitate the coordination of decision-making, skill and knowledge (Hughes *et al*, 1991; Scott Morton, 1991).

The general impact of these changes is to create a need for a much greater knowledge of work, its activities and its organisation than was necessary under older forms of production. In particular, it involves recognising that human skill

---

3 The literature on this is voluminous and, as is to be expected, the debates rage without much resolution. See, as a selection Bell (1976), Aglietta (1987), Atkinson (1986), Bagguley *et al*, (1990), Jones (1991), Hughes *et al*, (1991).

and knowledge is a vital ingredient in even the most routine of work and that the new requirements for flexibility require technical support that enhances such skill and knowledge. Thus, in this respect CSCW places itself within the context of thinking about new forms of work and its organisation and how these can be coupled to new forms of system support.

Changes in Computing

Friedman (1989, 1990) writes of the history of IT development in terms of the features that have been most restrictive to its development. The restricting factors are the problems that attract the most attention and, when alleviated, the focus moves on toward other, and newer, problems. In the early days of computing the cost of hardware and processing capacity were the main constraints. When, as a result of research and development, it was possible to build acceptably efficient and reliable systems although technical problems did not disappear, the most restrictive problems were to do with program development. Only when enough experience in building systems had accumulated did it become possible to seriously consider fulfilling the needs of a growing and increasingly variegated body of users.

In other words, the problems of system design are, in significant part, moulded by the technology in at least the sense that as technological problems are solved, new possibilities emerge and, with them, new problems to solve. This is not to argue for any technological determinism, but simply a reminder that in system development the technology constitutes an essential base infrastructure which shapes the possibilities of use. However, of major relevance to the rise of CSCW are the problems identified in the third of Friedman's phases, namely, those attaching to the use of systems, particularly their incursion into the world of work and organisation.[4] And it is this phase which connects with the changes in work organisation noted above.

It has been recognised that for some time that the next major step in computing would be its convergence with communications; a trend indicated by the emergence of a 'groupware' market, especially in the United States but increasingly in Japan and Europe, which will receive further impetus with the emergence of high bandwidth network services such as ISDN. The most recent figures for Internet, for example, state that the system now operates in 26 countries and supports several million users on over 5000 nodes and there is every reason to expect that this will increase rapidly.

However, CSCW is not solely concerned with widely distributed network systems, although these will, of course, be one of its major thrusts. It is also, and importantly, concerned with the development of systems that support collaborative work activities, whether these be within more traditionally conceived work settings or enabling new patterns of working to emerge

---

4 Incidentally, this is what Friedman (1989) discusses as a possible fourth phase in system development.

It is in this respect that CSCW complements, enhances and, sometimes criticises, the concerns of orthodox HCI and system design and, in doing so, brings a new perspective on the human-computer equation.

Human computer interaction

HCI grew out of the problem of trying to evolve methods of design which would improve the efficiency of machine use by the human operator. Better designed controls, reductions in both mental and physical strain on the operator would, potentially, improve the performance of the human machine system (Bannon, 1991). This provoked a new field of study known variously as 'human factors engineering' or 'ergonomics'. Physiology, medicine, behavioural science, industrial engineering, and more, contributed to the effort to understand the human capabilities and limitations that affected human-machine performance. The advent of the computer, however, introduced, though not immediately, a wholly new set of considerations in machine design which can be encapsulated in the distinction between 'operators' and 'discretionary users' (Bannon, 1991). In the still recent past machine design had tended to focus on operators, that is, persons who would be direct users of the machine and, accordingly, who had to be trained in its technological functionalities. However, with a growing number of computer users, mainly as a result of the introduction of personal computers, whose interest was in using the computer as a tool to serve their primary job, ease of learning and use became paramount concerns. It was this trend which stimulated the rise of the field of HCI which had a more cognitive and theoretical slant than human factors. Its aim was to achieve a better 'cognitive coupling' between the human and the computer, particularly in respect of interface design, and is a field which has grown enormously since its beginnings a little over a decade ago (Bannon. 1991).

Though HCI has undoubtedly made considerable advances in many areas of system design (Shneiderman, 1987), it has evoked serious criticism, particularly for its lack of relevance both to system design itself and to the 'real world' context in which systems are placed.

The first of these complaints refers to the 'cognitive bias' of HCI in its emphasis on experiments and the use of naïve users, its focus on the individual user, and its lack of relevance for the 'real world' of system design. Proctor and Williams (1992) argue that the contribution of psychology to HCI theory remains "mired in the low level mechanics of human performance". Carroll (1990) goes as far as claiming that information processing psychology has had no identifiable impact of design practice. While these are arguable conclusions, more recent work within the HCI tradition itself has begun to broaden the previously narrow cognitivism which has hitherto dominated the literature. One response to this, for example, is the recent attention given to the notion of 'distributed cognition' (Hutchins, 1990; Olsen and Olsen, 1991) as well as a growing awareness of the sociological literature which argues that 'cognition' is best understood as a situated, pragmatic activity rather than an 'invisible' mental event (Coulter, 1983).

The second complaint, and of more direct interest to CSCW concerns, is the need to acknowledge the social context of work activities and to bring this to bear upon system design. While there is a growing acceptance within HCI, as indicated by the way in which the term 'usability' is frequently employed in the literature, of the need to be concerned with the totality of the design problem, traditional HCI's repertoire of tools remains woefully inadequate (Proctor and Williams, 1992; Grudin, 1992). Although computers have always supported, and equally often disrupted, collaborative work settings from the earliest days of mainframe computers, there is now a growing awareness of the need to provide such support in ways that 'resonate' more suitably with work activities (Schmidt and Bannon, 1992). This means recognising that in most work situations the accomplishment of the tasks that constitute the work involves multiple individuals integrated in some way with others and, as a result, appropriate computer support must allow for the smooth interleaving and coordination of tasks across people and machines. Grudin expresses this trend as a movement of the user interface "farther and farther out from the computer itself, deeper into the user and the work environment" (Grudin, 1990). Thus, whether or not CSCW represents a new 'paradigm shift' in computing (Hughes *et al*, 1991) it broadens the concerns of HCI as previously conceived.

The influences sketched out above have all played their part in the development of CSCW as an identifiable research field drawing upon a number of developments in software engineering and informed by social science traditions in the analysis of interaction and group activities. It is fair to say that CSCW remains, by and large, a research endeavour. Most existing CSCW systems are prototypical existing in research laboratories and addressing problems associated with supporting relatively small groups. Certainly market penetration and user acceptance of such systems has remained low. While such a state of affairs is understandable in a new field, if it is to progress beyond this stage there is a need for basic research to be more directed toward the design and the development of 'real world' systems, and it is this that the COMIC project sets out to address in its various themes (see The Comic Project Proposal).

CSCW Design and the Great Divide

Although many of the problems of CSCW design are problems which attend system design more generally, one of the distinctive features of CSCW is the prime importance it attaches to understanding the socially organised character of work as a prerequisite for effective system design. This implies, and has been widely recognised from the early days of CSCW, that CSCW design will involve a number of disciplines (Grief, 1988; Galegher *et al*, 1990). Although the unity within disciplines can be overemphasised, perhaps the more significant divide in CSCW design is that between the human scientists and the engineers.

The crucial feature of this 'great divide' (Bowker *et al*, 1993) is not so much the obvious fact that one group deals with building technology while the other deals

with aspects of human life, but the consequently different 'mentalities' that are typical of the two communities. On the one hand, and briefly, system designers require that problems be broken down into manageable proportions, that they be thoroughly analysed and specified, that the process of design and development be organised methodically, that they be simplified and systematically laid out, and so on. As far as the social sciences are concerned, however, while there is less of what can be described as a common 'mentality' among them, with the possible exception of experimental psychology, the divergence with that of engineering is sharp. It is far more difficult to simplify the problems of social science, for example, much more difficult to measure and analyse the phenomena of interest, much less easy to secure a clear and more or less agreed conception of what the problems are, and more.[5] The result is that while engineers make, as they have to, a number of presumptions about the social and cognitive character of putative users, they often do so without systematic or satisfactory input from social science. The danger is that the social dimension is seen through, and only through, the eyes of engineering. On the other side, however, social science is singularly ill-equipped to offer the kind of detailed analyses of work that is required by the engineering aspects of system design.

A number of consequences of this difference have been noted for CSCW since the 'divide' is an area which is crucial to CSCW design. One major response is to call for the development of a common vocabulary, or common framework, to bridge the gap (see, for example, Schmidt, 1993; Schmidt and Carstensen, 1993; Kuutti, 1993). Another, and less dramatic, response is to argue for the development of a better grounded sociology of work as a 'workaday world' (Moran and Anderson, 1990) drawing on the inspiration of Ethnomethodology and, as a means, ethnographic studies of the settings of sociality technology and work practice. Both of these are major research issues and, as such, reflected in much of the work of COMIC in all the Strands.

It is realistic to assume, however, that CSCW will, at least for the foreseeable future, be an inter-disciplinary endeavour in which human scientists and technologists, bearing in mind that these are broad categorisations themselves, will have to find ways of working together in the design process; and this itself is a major research focus, particularly in this Strand, which explores the issue through a predominantly methodological gaze. For us the issue is one of finding appropriate places within CSCW design for the analysis of the sociality of work and for the concerns of software engineering in ways that bridge the 'divide'.

---

5 It is important to be as clear as we can as to what we mean here. There is no implication that one group of disciplines is superior in its approach than the other. What we are trying to recognise is the inevitable differences in approach. There are, course, many and diverse arguments as to the source of such differences which are beyond to scope of this report.

The Requirements Problem

The main purpose of this Strand is to investigate the requirements process; that is, the process by which system designers and developers come to understand what is to be built and how it may be done. In initial terms it is directed to informing the following issues in system design and development which can be characterised in general terms as follows:

- understanding the domain of application
- communicating knowledge of the domain of application to designers and developers
- managing the development of the system itself

Understanding the Domain of Application

This aspect is what is typically referred to as 'requirements capture' or 'requirements elicitation'. It is that aspect of the design process, be this conceived as a single stage in the design process or a continuing one, which seeks to understand and characterise the relevant features of the domain of use for system design. In this respect it is of relevance to determining what functions the system should perform, how it may display these to users, what parameters of the human-computer equation should be specified, and so on. In CSCW, the prevalent domains of concern are those of work which involve individuals in cooperation with others.

Relating Knowledge of the Domain to Designers

This refers to that aspect of the process which is concerned to relate or communicate knowledge of the domain to the design and development process. To the extent that design and development is likely, with respect to large scale systems, to involve a number of people with different expertise, this issue becomes a much more explicit one. The question of how far and in what ways users should be involved is also a pertinent issue here.

Managing the Design and Development Process

Any large scale complex system will need to be built under the familiar resource constraints of time, manpower and finance. As such, it will need to be a systematically organised process involving some division of labour among possibly changing personnel of varying specialisms. Thus, the relevant features of the domain of application must be rendered in a form which supports this feature of design and development as part of managing the system design process.

Concluding Remarks

We have deliberately stated the issues in general terms to avoid prejudging much of the discussion later in the Deliverable. In Part II we review and evaluate some of

the more salient methods and techniques for specifying requirements and for formalising the system development process in the context of CSCW.

As stated above the issues represent collections of problems involved in system design and development and any method or approach would need to say something about each of them. We are less concerned with the question of what requirements are, so much as trying to understand the place of the requirements process within large scale system design and development. After all, many of the problems of CSCW are issues of system design more generally. Clearly, such issues are matters of method if only in the broad sense that tackling them requires the systematic application of some publicly available and agreed upon procedures intended to achieve some desired result. Of course, design is about more than method and we do not, in speaking of method here, necessarily imply a high degree of formality to them, though some of the available methods do seek this as an objective. Further, and this is very characteristic of the debates surrounding requirements, methods run the risk of being seen as panaceas when it would be more appropriate to treat them as pragmatic solutions to the very real problems that system design poses.

In Part I we review some of the arguments surrounding methods, using the Waterfall Model to develop these. Our aim is to argue for the consideration of requirements identification within the broad structure of the system development process within the context of industrial organisation.

Part I
Panaceas and Rhetorics in System Design

Part I: Panaceas and Rhetorics in System Design

Preamble

In this part of the Deliverable we face a particularly delicate task in seeking to set aside certain issues and approaches without summarily dismissing them. The remit we have set ourselves is to approach the problem of system development and requirements elicitation in the expectation that, for the foreseeable future, a substantial proportion of system design and development will be governed by many of the current methods within relatively unchanging conditions of employment and the industrial division of labour. This is not to argue that there are no problems arising from the industrial production of large scale systems, only that dealing with them will have to be, in practical terms, within this framework.

This is not to say that some of the more utopian proposals might have some practical and valuable contributions to make, modifications which could well improve the system development process even if they do not have much hope of realising the larger objective of the programme from which they derive. The case of Participatory Design is a good example of what we have in mind here. It is an approach which is often vested with much broader goals than just the development of more 'effective' information systems including a desire to restructure the nature and organisation of work itself by taking it in a more democratic direction. The fact that the actual practical success of Participatory Design has been limited in scale and effect does not mean that the enthusiastic campaign for this approach to design has not made a contribution to changing sensitivities in the world of design. At least it has contributed some impetus to the growing concern to giving 'users' a more active part in the design process.[6] However, alerting designers to the need for more user involvement can be achieved without any consequences for the advancement of the democratisation of labour relations. Accordingly, our objective is not to debate the value of the longer term practicality of thoroughly democratised work relations, but simply set these issues aside as having little immediate import for the practical conduct of current system design and development.

This part of the Deliverable is one in which 'sociological' questions loom large and it needs to be said that the kind of comments we have made in regard to Participatory Design also applies to many of the sociological approaches which have begun to appear in connection with system design. We are dealing with matters which are often the focus of intense polemics but seek to treat them in a relatively detached manner. We are also alert to the fact that sociology, as a discipline, is one in which matters are often complicated and difficult to resolve. Nor do we claim that our own preferred sociological positions are unassailable.

---

6 We will not keep placing user in scare quotes but we are aware of the gnomic character of the expression.

Rather, one of our aims is to counsel caution with respect to the enthusiasm with which some approaches are embraced within the context of system design. Our particular focus is upon the practicalities of system design arguing that many of the approaches which we discuss in the following sections will only compound the problems of practical system development. Thus, setting aside work within the tradition of the Frankfurt School or the Strong Programme in the Sociology of Science and Technology is motivated as much by considerations of their implications, or lack of them, for CSCW system development, as it is by the disciplinary considerations of sociology.

A major concern of this Part of the Deliverable is to review aspects of the debate about the nature of the requirements capture and specification process in the system development process. We use the Waterfall Model (Royce, 1970; Boehm, 1976) of the process as a stimulus for this review arguing that much of the debate surrounding it has tended to produce confusion rather than clarification. In this respect, the Waterfall life-cycle model is a point of departure. Many of the confusions are inherited from more traditional HCI concerns and sit alongside the new sets of problems arising out of the distinctive focus of CSCW, in particular, its insistence that system design be much more informed by the analysis of 'real world' situations of use. In addition, the invitation to bring in new disciplinary resources for CSCW design, including sociology, social anthropology, along with new methods, such as ethnography, have combined to open a Pandora's Box of issues which are not always clear to see, certainly not easy to solve but which, nonetheless, require serious attention and consideration with a view to making them manageable and effective contributions to the very practical problems of designing CSCW systems.

As we have already indicated, one of the salient features of recent discussions of system design is the invocation of 'social science' as of more than just passing relevance to the task of developing successful systems. Robertson (1993) goes so far as to say that "with or without designers' involvement, the ideas and issues evolving in sociology, psychology and anthropology will shape design for the balance of the decade and into the twenty-first century". However, he continues with an important warning that "this is a case where a little knowledge is decidedly dangerous. There are vast differences in theory and in basic assumptions that underlie the various social science methods out there."

Much of this part of the Deliverable is written in broad concurrence with Robertson's judgement. While we are confident that 'social science', more particularly 'sociology', potentially has the capacity to make useful contributions to the resolution of some of the problems of system design, we are also confident that the development of this potential needs to be undertaken with some circumspection. 'Social science' and even 'sociology' are the titles of very heterogeneous enterprises consisting of assorted approaches which are often profoundly at odds with each other. Accordingly, working out the value and the meaning of these various approaches and their contribution to system design is not likely to be a straightforward matter. Indeed, this is one of the issues, in various guises, that

permeates the COMIC project as a whole. Moreover, and as already indicated, in this Part of the Deliverable we shall have occasion to doubt some of the implications that have been drawn from sociology and its contribution to system design. Too often, we argue, sociology has been treated like a 'world view' used to attack many of the techniques and practices of system development, or the organisation of work more generally, as emblems of yet another 'world view'. For example, and one which we discuss at some length, the Waterfall Model is taken as exemplifying the 'rationalist mentality', particularly in its formalist bent, and, as such, a target for all that is allegedly wrong with modern society. For us, such criticisms tend to have their source in the abuses, excesses and misapplications of formal techniques rather than a criticism of formalism *per se*, and also exhibit no little failure on the part of social scientists and sociologists to understand the nature of formal disciplines and the constructs which they generate. This aspect of the discussion we will take up more thoroughly later. For now, we want to begin our discussion of the Waterfall Model.

Requirements and The Waterfall Model

What is commonly referred to as the Waterfall Model of software development is an approach to system building which specifies the process as a series of well-defined steps. It evolved from general systems theory and from the planning and logistic work inspired by Operations Research during the Second World War (Agresti, 1986). The idea of the 'life-cycle' is intended to capture the overall process of system development, from the inception of the idea through to the eventual elimination of the system from service. In significant respects, and as its origins suggest, software system development is treated as an instance of project organisation and management. Crucial to the model is the importance attached to the early phase of determining system requirements for the design and holding off implementation of these in software until late in the whole process. There have been a number of variants proposed since the original version appeared in the early 1970s, though all follow much the same logic.[7]

As indicated, the requirements phase is a critical first step. The derivation of requirements involves a number of phases:

1. *Feasibility study*

   This involves making an estimate of whether the identified user needs are satisfiable using current software and hardware technologies, whether the proposed system will prove to be cost-effective and whether it can be developed within the constraints of time and budget.

2. *Requirements capture and analysis*

---

7 Variants often emphasise different phases. Thus, for example, the development of Structured Methodologies is often treated as an advance on classical methods by providing a more elaborate articulation of the middle stages of the life cycle and a more systematic working out of the design and its implementation. Other approaches place their stress on 'front end' loading and postponing detailed design until the requirements and specifications are more assured.

This is the process of deriving the system requirements through the observation of existing systems, discussions with potential users and procurers, task analysis, and other methods.

3. *Requirements definition*

   A system model is formulated and used as the basis for an abstract description of the systems' requirements. This is primarily a document which describes the system from the user's point of view.

4. *Requirements specification*

   A detailed and precise description of the system requirements is set out to act as a basis for a contract between the client and the software developer. The creation of this document might be carried out in parallel with some high-level design (indeed, this is often essential) which will influence each other as they develop.

The sequence of activities and activity iterations are shown in Figure 1 where activities are represented as round-edged rectangles and deliverables as square boxes.

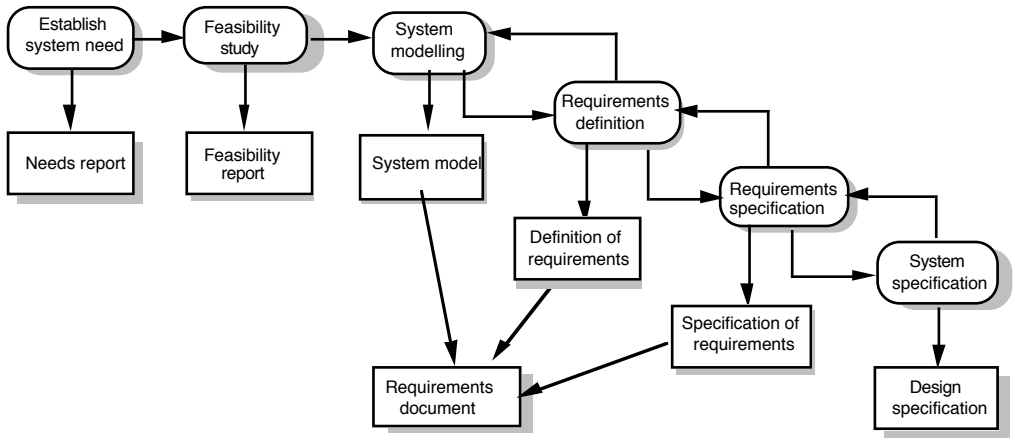

Figure 1: The activities involved in requirements capture and analysis

Essentially the process is a step-by-step one, each stage intended to move closer toward a more detailed specification of what the system is to provide. Of course, the activities in the requirements process are not simply carried out in sequence but are iterated. The requirements analysis continues during definition and specification and new requirements may emerge during this process. Nevertheless, requirements capture and analysis is typically seen as an early vital stage in design and one which has a major impact on the final system as the outcomes, agreements, and decisions reached during the process of 'capture' constrain 'downstream' options and can be difficult and expensive to change once implemented (Bowers and Pycock, 1993).

There are also a number of methods which have evolved to guide this phase of the design process, and others are being developed. Many of these methods are reviewed in Part II of this document.

The Software Requirements Document

A crucial element in the process is the production of a software requirements document which defines the system to be built and, thus, is often the basis of the contract between the system procurer and the system contractor.[8] The point of the Waterfall Model is to lead to the production of a requirements document which spells out, as clearly as possible, what services the system should deliver — an answer to the question, what should be built? — and, through this, making it possible to systematically organise the building of the system. It categorises the process of system development into distinct stages which occur one after the other although there is some feedback between the stages.

In principle, the requirements set out in such a document ought to be complete and consistent. All system functions should be specified and no requirement should conflict with any other. In practice, this is difficult to achieve, particularly, it is often argued, if the requirements are stated in natural language. Errors and omissions will inevitably exist in the document so it should be structured so as to be easy to change. Heninger (1980) claims that there are six requirements which a software requirements document must satisfy:

1. It should only specify external system behaviour.
2. It should specify constraints on the implementation.
3. It should be easy to change.
4. It should serve as a reference tool for system maintainers.
5. It should record forethought about the life cycle of the system.
6. It should characterise acceptable responses to undesired events.

The requirements document is a combination of requirements definition and requirements specification. The task of developing a software requirements document should not be underestimated. Bell *et al* (1977) report that the requirements document for a ballistic missile defence system contained over 8000 distinct requirements and support paragraphs and is made up of 2500 pages of text.[9] This is perhaps larger than most systems but it illustrates the point that a great deal of resources have to be dedicated to the production of a requirements document and that this represents a significant cost.

The software requirements document is not intended as a design document. It should set out what the system should do without specifying how the requirements it states should be implemented. However, the distinction between requirements specification and design is not one which can necessarily be sustained in practice. The requirements should be stated so that the design may be validated. If the services, constraints and properties specified in the software requirements

---

8 The terminology here can sometimes be confusing. Requirements definition is sometimes seen as the basis for bidding for a system contract and the requirements specification, or functional specification, the basis for the contract. Sommerville (1992) suggests that the distinction between these is arbitrary but it is important that the combined definition and specification is structured in such a way that it may be readily understood and analysed as a whole.

9 This has occasioned the wry comment that perhaps the documents should have been dropped on the enemy!

document are satisfied by the software design then that design is an acceptable solution to the problem.

The Waterfall Model as an Emblem of all that is Wrong with Design

The Waterfall Model has become, for many, the emblem of much that is undesirable in the system development process and, again for some, an example that CSCW system development should avoid at all costs.

One of the prominent objections to the Waterfall Model is that its 'staged' structure involves the false and counterproductive discrimination of design activities from one another. There is also some dissatisfaction with the notion of 'capture' as if this was a 'one-off' stage which produces sufficiently stabilised specifications of the system's requirements which are then regarded as 'complete' and the end of the story. Once the requirements capture phase is completed the process of design and development can be set in train. However, and as the experience of many designers and system developers attest to, no design process is like the Waterfall Model says that it is. In other words, as a description of the actual process of design and development the Model is seriously misleading.

Nonetheless, even as a set of prescriptions, the 'requirements capture' stage is condemned as effectively segregating those who need to be brought together if adequate design is to be achieved, namely, the designer and the user (cf. Bowers, 1991). More particularly, such Models along with their associated methods have been criticised as not being sufficiently user-centred. As Norman and Draper (1986) argue, an important focus for the requirements capture process should be establishing who end users are and what their tasks, goals and concerns might be. Others are prepared to go further than this injunction which, after all, is the objective of many of the methods used in the requirements capture process. They challenge the conception of the user which is presumed in the kind of narrow cognitive science which has traditionally dominated system design and in the methods which support the requirements capture and analysis process (see Bannon and Bødker, 1990; Grudin, 1990; Bowers and Rodden, 1993). Participative Design, for prominent example, urges a 'cooperative prototyping' approach to involve the user more directly in the design process (Bødker and Grønbæk, 1991).

The Model also reinforces the sense that the developer is a critical figure and that requirements are an input into his, or her, activities with the process, consequently, dominated by the kinds of considerations which facilitate the designer's work rather than ensuring the eventual adequacy of the design itself. This is particularly apparent in the complaints made against the methods used to support and formulate requirements, which are condemned as positivistic, formalistic and geared to the interests and concerns of design engineers rather than those of users.

However, there are criticisms which are derived not so much from the practical concerns of software engineering but from more metaphysically inspired critiques,

the main effect of which is to sow confusion. It is these which have opened up a Pandora's Box for system design.

Opening the Pandora's Box of System Design

We characterise the debate which has arisen over the Waterfall Model as opening a Pandora's Box because the range of issues that were opened up suddenly and massively multiplied beyond computer science, system design and software engineering. System development becomes a multidisciplinary venture, with the software engineer seemingly needing to understand and have a mastery in the practical application of a plurality of disciplines, as the following quotation from Goguen and Linde (1993) suggests:

> "...the majority of computer based systems are developed without any systematic help from the social sciences (sociology, psychology, linguistics, anthropology, etc.). This means that the needs of the user, both an individual and as organisation, are not addressed systematically; in general, they are only completely known to the development team, and there are often some serious misconceptions".

Indeed, the very idea of system design as engineering is called into question. The design process is seen by many critics as overly formal, atomistic and technocratic. It has all the characteristics of engineering by breaking things down into small manageable parts so as to deal with them in strict, separated divisions of tasks and driven by the myopic supposition that the brute and persistent application of these methods will solve all problems. Shotter (1991), for example, has complained that the use of cognitive psychology in articulating the design process is equivalent to a Taylorism of the mind. What such a mentality fails to appreciate is that there are problems in design other than the technological.

Thus, the critique of the Waterfall Model, and similar ones, changes from being just a critique of a managerial and engineering device to a critique of engineering itself and the broader tradition of scientific and technological thought which the Waterfall Model is taken as representing. What has happened, and it is partly the result of social science interest in system and work design, is the opening up of a mass of issues bundled together so that they become difficult to disentangle. The arguments reach the point of becoming a Manichean struggle and the consequence of opening the Pandora's Box is the call for new paradigms of system design and development — 'situated action', 'work oriented design', 'participatory design', 'activity theory' among others — which draw upon assumptions that are, at the least, open to debate and which confront, in their turn, serious difficulties in their practical application to design under real world conditions.

In Pandora's case, once the evils of the world had been let loose, only Hope remained behind in the Box. However, in the case of system design what has been loosed is not evil but confusion. Argument about the Waterfall model is often over what that model has been taken to represent, which is the adoption of an 'engineering' approach to the building of software. The debate, therefore, can become polarised between those who want to pursue a straightforward engineering approach — in the context of CSCW, what Howard (1988) would term 'strict

constructionists' — and those who would want at least to loosen the ties with the engineering tradition, if not abandon them altogether.

Disentangling the Social Issues

As indicated one of the major problems here is obtaining a clearer sight of just what the problems are. In this case, it is disentangling what we refer to as the 'metaphysical' problems of system design from the more practical ones. To approach this we will take up two issues:

- formalism and representation
- the nature of work.

These are intended to be illustrative of our concerns rather than any attempt at an exhaustive checklist. They have been selected not only because they are key issues in system design and development, but also because they are topics which are of direct concern to arguments about the role of sociology and social science in system design and, as such, have inherited a large agenda of issues and arguments.

Formalism and Representation

As has already been suggested, what is being attacked by many critics of system design is nothing less than the rationalist programme which exemplifies modernity. Bowers (1992) offers a working definition of formalism as a "representational system" which generates "representations through the operation of rules over some vocabulary". The exemplar here is that of mathematics but would also include computational procedures, computer languages, techniques for specifying the organisation of information in a database, notations and representational schemes used in system analysis and design, such as Yourdon (1989). The critique of the role of particular formal representations within system design may point to their specific practical inadequacies, such as the way in which they produce unnecessary rigidities within the organisation of activities, but the particular representations are not the true focus of attack. Rather, it is an entire and main tradition of thought within modern Western civilisation — that associated with 'the Enlightenment' — which is being criticised. The techniques of formal representation are merely instantiations of the working of the 'rationalist' mentality which has been responsible for, *inter alia:*

> "The rise of individualism, the invention of perspective, Protestantism, the discovery of alphabetical order for dictionaries, the ordering of practical and abstract knowledge in the *Encyclopédie,* the invention of interchangeable machine parts, the standardisation of weights and measures, Linnæus' ordering of the species, the substitution of legal codes and constitutions for spontaneous common law, and the creation of standardised tasks in the factory system — these and other events were symptoms of the imposition of the 'clear and distinct' on the complex, explicitly oppressive, and hierarchical *communitas* of the Middle Ages."[10]

This 'modernist' mentality sees itself as an emancipating one, achieving liberation through the use of rational, formal, techniques. 'Post-modern' critiques,

---

10 The quote is from Wilden (1980) and is also quoted in Bowers (1992).

however, maintain that the modernist mentality serves an essentially oppressive function (Lyotard, 1984). Rationalism may aspire toward the creation of a universal system of objective knowledge, but apparent progress toward this objective is achieved only through the *mis*-representation of reality, disregarding the extent to which reality is inherently contingent and perspectival. Rationalism's agenda is entirely transparent: it is that of acquiring objective knowledge of reality. Its critics, however, challenge the claim to transparency, and seek to expose its hidden agenda as one which involves rationalism in the service of dominating and exploiting interests and, thereby, in the falsification of the facts of domination and exploitation[11]. The immense enthusiasm which there is for technological and scientific 'solutions' are, then, regarded as manifestations of the power of the rationalist programme, the belief that the application of the techniques of investigation and analysis will generate impartial answers to all questions. Scientific knowledge and technology have therefore been prominent foci for the critique of rationalism, very commonly through the attempted demonstration that scientific and technical work is *permeated* by socio-political interests and which is therefore a medium for the struggle between interests (Winner, 1980; MacKenzie and Wajcman (eds.), 1985) or through the argument that it involves the building up of social relations which involve the concentration of power (Latour, 1990).

The foothold these arguments obtain in system design is through the latter's extensive use of formalism in the design process. The critique of rationalism motivates the view that systems of representation cannot truly capture reality and that, therefore, the complexities for which designers seek to design will escape the representational tools that they employ. Thus, Schmidt (Schmidt, 1991b; Schmidt and Carstensen, 1993), argues that any system which embodies a model or some formal representation of the social world will inevitably come across situations where the model will be inapplicable. No representation or model can be adequate to all the contingencies it may come across. However, extending the model 'beyond its bounds' can result in 'disasters' in that users are forced to use the model imposed on them by the system.

Similarly, Robinson and Bannon (1991) express an unease about the use of concepts such as 'role' in systems such as COSMOS (cf. Dollimore and Wilbur, 1991) since it imposes an ordering which is likely to bear little relationship to the interpretative reality within which people act. Fatefully, such formal representations create major problems of communication between designers and users. Robinson and Bannon (1991), for example, speak of this as a process of 'ontological drift' in which designers are prone to confuse their formal models with the reality they are supposed to represent.

The critique of rationalism also motivates the view that it is not merely the empirical inadequacies of formal representations which is the problem, but also the 'hidden agenda' of control which is involved in their implementation. Thus, in a

---

11 This critique owes much to the Frankfurt School's critique of 'instrumental reason' as sharply expressed in Marcuse' (1964), and continued down to the present in the work of Habermas (1971).

recent paper Suchman (1993) argues that speech act theory, as instantiated in Winograd and Flores' COORDINATOR (1985), in being formal is attractive to computer scientists because it offers the sort of model which, formal, abstract and general will appeal to the rationalist cast of their outlook, but that it is in fact a system, following Foucault (1977), that is a means of extending institutional control over the actions of individuals.

> "In the move to inscribe and encode organisation member's intentions, as commitments or otherwise, we find a recent attempt to gain members' compliance with an externally imposed regime of institutional control...with a scheme of standardised, universalistic categories, administered through technologies implemented on the desktop."

The rationalist influence on 'system design' is, of course, most manifest in the faith of those who see it as a straightforward engineering discipline and who expect that persistence in the application of the formal techniques of engineering will eventually resolve all the serious problems of development. The critiques we have just been outlining obviously suggest scepticism rather than faith, for they are suggesting that there are inherent limitations on the capacities of formal representations, of the sort involving what Woolgar (1988) has termed 'the methodological horrors'. These are, following Bowers (1992), set out below:

- *Indexicality* — the connection between the representation and the object represented is radically contextual in its meaning and sense, which means, for Woolgar, that it is always open to defeasibility.
- *Inconcludibility* — any attempt to precisely define the representation is open to the request for further clarification, elaboration, elucidation, etc..
- *Reflexivity* — the relationship between a representation and the thing represented is 'internal' in that knowledge of the former is explicated by knowledge of the latter.

Bowers and Middleton (in press) offer two more to this list:

- *Dialogism and Dissensus* — representations are contingent on the fact that other competing representations might be possible and might be offered. Thus, the sense of any representation has to be relativised against the dialogue of which it is a part.
- *Incompleteness* — any representation picks out some features of its object and it is uncertain to the extent to which these features may become seen as 'wrong'. Higher order representations to handle the uncertainties of lower order ones will also be incomplete.

This is the simplest and briefest possible outline of the 'critique of modernity' and its rationalist orientation, and it is clear that the examination of the actual meaning of these arguments for the actual practice of design would require a considerably more extensive discussion. There should certainly be caution about drawing ready conclusions from this critique. It would be mistaken, for example, to suppose that the 'methodological horrors' deny all validity to formal representations for, as Bowers (1992) points out, these 'methodological horrors' — if that is what they are — are practically managed, as they would have to be if

any system was ever to be built or if any of the many formalism-permeated practices in which we engage were to run. The pursuit and use of formalisms need not involve a blind faith in its utilities and capacities, as though all of those who employ techniques of representation must have unquestioning confidence in their effectiveness. The thoughtful contrivance and use of formalisms should involve an awareness of the extent to which this involves simplification, even falsification, and thus of the extent to which the application of those formalisms in practice must be constrained by the (often incalculable) complexities of the circumstances to which they are to be applied.

As we shall develop further in Part III of this Deliverable, many sociologists fail to understand the role of simplification in modelling as well as many in formal disciplines lose sight of the fact that their models are simplifications. The tendency that Robinson and Bannon (1991) refer to as 'ontological drift' is just this failure to see that prescriptions derived from formalisms are only applicable where the necessary simplifications obtain. However, the interpretation of a set of simplifying assumptions in realistic terms and the incautious application of prescriptions without regard to the necessary qualifications, is a deviation from, not the essence of, formalisms.

We now turn to examine critiques of Waterfall type models which are, in effect, critiques of the nature of work in modern society.

The Critique of Work in Modern Society

Both Marx and Weber, whose thought decisively shaped the course of modern sociology, viewed the worker's experience of work within modern, capitalist society as an essentially alienating one, the evolution of industrial work being one which progressively divested the individual's work of any meaning and denied the individual's control over it. Such views suggest the need to critically examine the character of business organisation for its tendency to increase its control over the worker. This means that understanding the techniques for organising work should go beyond the purely 'technical' but needs to be analysed as part of a wider set of relationships within the work place, particularly between those who control and those who are controlled, and also the relationship of work within the lives of those who carry it out. Clearly, the introduction of information technology and of computer and information systems can be examined in such terms, particularly with reference to the question of whether they effect managerial control over 'the labour process' through the deskilling of work (cf. Braverman, 1974; Wood, 1989), and more or less indirectly, the influence of Marxist and Weberian views about the organisation of work in modern society can be seen, with concern for the 'democratisation' of work and the autonomy of the individual worker at the forefront.

Though Ehn (1988), for a prominent example, draws substantially upon Heidegger and Wittgenstein to provide the intellectual justification for the view of the role of work in human life and as a rationale for the UTOPIA project, many aspects of that project's actual recommendations reflect the Marxist view that

management's attempts to reorganise work relations are very commonly efforts at 'deskilling' the actual task involved, part of the process which Braverman (1974) terms 'the degradation of work in modern society. [Aspects of this are further considered in Strand 1 of this project] The aim of Ehn's argument is to emphasise the importance of work within human life and to highlight and overcome the discrepancy there is between the designer's understanding of work and the meaning that it has for those who do it, using a contrast between the kind of work which is involved in a craft with that characteristic to much mechanised, bureaucratised work in modern industry. In Ehn's terms, it should be possible to recognise and legitimate the 'semantic diversity' of different work communities and close the 'ontological gap':

> "If designers and users share the same form of life it should be possible to overcome the gap between the different language games. It should at least in principle be possible to develop the practice of design so that there is enough family resemblance between a specific language-game of design and the language games the design of the computer artefact is intervening in. A mediation should be possible".

Comparably Floyd (1987) formulated a contrast between two paradigms of software development, one of which approximates to the positivistic conception (which equates, in practice, with a technocratic standpoint), again which the Waterfall Model exemplifies, with an emerging 'humanistic view'. Under the influence of such critiques as those advanced by Habermas (1971) the 'rationalist' cast of this positivistic conception aligns it with the organisational forms of capitalist production and bureaucratic administration, combining to exert rigorous and unilateral control over the activities of workers. Hence, UTOPIA and similar projects are as much concerned with the democratisation of work as they are with system design and development.

In many respects this kind of debate strongly echoes those of some years ago over Scientific Management, or Taylorism, which began at the end of the nineteenth and the beginning of the twentieth century and which realised its apogee in the vehicle assembly line.[12] The Waterfall model, in its attempt to break the process of system development down into a multiplicity of small and closely co-ordinated tasks has a distinctly 'Tayloristic' aspect. Scientific Management sought to maximise productivity and efficiency by the application of the following principles:

- dissociation of the labour process from skill
- a reduction of tasks to their simplest components
- the planning and control of all elements of the labour process

For Braverman (1974) Scientific Management was an aspect of the development of monopoly capitalism and the means by which management effected greater control over the workers and the labour process. Despite the demonstration by the Human Relations School of Industrial Relations and, later, by Socio-Technical Systems design that approaches such as Scientific Management could be self-

12 See Rose (1988) for a good account of Scientific Management.

defeating by reducing employee motivation (Trist *et al*, 1963), it has survived as a practice in many industries.

It is not only in manufacturing that such influences have been detected. A number of commentators have argued that the lessons of the factory are tending to become the guiding principles of office automation. They have argued that for many professionals and managers, computerisation signals a loss of autonomy, the fragmentation of tasks and closer supervision, processes which had earlier affected the manual labour force (Kling and Dunlop, 1992; Clement, 1988). Drawing on Braverman's argument that owners and managers relentlessly seek to enhance their control over workers to cut labour costs, Perrolle (1991) notes how 'application generators' can simplify the production of computer programs so that they can be written by less qualified clerks.

It is not difficult to see how such arguments might well be applied to the process of system design as it is organised under industrial conditions and, accordingly, shaped by the requirement of industrial organisation and the wider workings of the capitalist economic system.

The main problem about such critiques is that it is hard to see what follows for system design other than a root and branch redesign of modern society on the basis of what are profoundly debatable social-cum-philosophical theories. This is not to say, either, that abstaining from making such comprehensive indictments of work in modern society equates with an uncritical acceptance of all forms in which work might then be organised.

As we have made plain, the purpose is not, here, to argue about, or against, the validity of such a critique of work. We simply notice that it has a 'Utopian' character and then set it aside because the prospect of realising its Utopian objectives are anything but imminent, if they are realistic at all (and would require much in the way of political organisation beyond the re-thinking and re-working of the design and development process itself). What is required is a more pragmatic attitude toward problems of design.

Within any foreseeable time scale the wholesale transformation of work is unlikely to result in the thoroughgoing restructuring of work in the directions desired by those who offer radical critiques of alienation and control. It is difficult to imagine Participative Design, for one example of an approach to design which is often association with the kind of critiques we are discussing here, being tried on anything like a societal scale. Outside of Scandinavia, it is, rightly or wrongly, unlikely to be adopted precisely because of its political agenda which, in turn, makes it unlikely that sufficient resources or political goodwill will be forthcoming to support the large scale development of such systems. In any event, even were it to be adopted, then this still leaves it with the problems that face any set of design principles; namely subjecting its proposed solutions to validation and test. Can, for example, the requirements capture process, as a separate, independent stage in the design process, be abandoned? Can formal diagramming and notational techniques be dispensed with? Can the problem of determining what the system is to do be resolved by means of direct interaction between the designer and the user in relation

to the realisable prototypes of the system's character? Can the user become, in effect, the designer with the 'designer' adopting the role of technical advisor on matters to do with feasibility, etc.? If these are possible, under what conditions are they possible? To what extent are Participative Design methods scalable up to projects which involve large numbers of designers and coders over many years? While it is a misconception to offer the Waterfall Model as an effective way of organising design projects involving small numbers of experienced and mutually known designers, would it be a similar abuse of Participative Design to use it as a method suitable for the kind of problems life-cycle models were designed for?[13]

Were it to be adopted it is likely that the political agenda associated with it will be truncated, and that its character as a vehicle of democratisation thereby transformed. Whatever long term movements there might be toward the restructuring of our way of life, the strong possibility is that the development of large scale systems is going to continue in the immediate future to be undertaken within conventional market and bureaucratic settings.

As we have already indicated, our argument is not that everything in the system design world is satisfactory: on the contrary. Nor are we offering a defence of the nature of contemporary work and its organisation. These are important matters which have a bearing on the role of technology, not just computer technology, in our lives. Our argument is that design and software engineering have their own imperatives which are likely to remain in place for the foreseeable future.

In the next sub-section we want to raise the question of the part that sociological perspectives can play in design, not merely as sources for the critique of conventional software engineering approaches.

Sociology and Requirements

Earlier we remarked that many of the confusions which have recently arisen in debates about the nature of system design are the result of the attention which social scientists have begun to devote to this topic. Yet, the argument made by CSCW that design needs to pay close attention to the 'real world' character of domains, a character which is 'social' whatever else it may be, is to also argue for an important role for sociology in the derivation of requirements in system design despite the confusions this appears to be causing.

In this part we intend to discuss some of the issues at reasonable length to prepare the way for our own view on what kind of sociological attention should be devoted to the analysis of work settings for CSCW. We begin with a paper that recommends the use of sociology in requirements elicitation. The objection they offer to extant methods of requirements elicitation, such as introspection,

13 To date, Participative Design has tended to be used on research oriented projects using a manageable group of users. They also tend to be devoted to demonstrating a principle rather than producing a thoroughgoing application. It is relevant, for example, that the UTOPIA project involved printers, the aristocracy of labour, who, because of their high morale, skill and sense of occupational identity were more likely to participate in the kind of method proposed by Participative Design.

interviews, focus and application development groups and protocol analysis is similar to that of the Procrustes paradigm (Schmidt and Carstensen, 1993), that is, the "imposition of an analyst's order on the social world, with no guarantee that this is the same order that members perceive, and with no way of even posing this as a research question" (Goguen and Linde, 1993). They identify two distinct but related tensions this judgement:

- the presumption of 'conventional' requirements elicitation methods of an *a priori* conception of science which determined the form that research must take;
- the consequent sense that social life, being intractable to these scientific procedures, in not amenable to scientific study and, therefore, chaotic.

As a corrective to these suppositions Goguen and Linde recommend 'ethnomethodology and sociolinguistic' inquiries. Much of their discussion is devoted to demonstrating that social life involves social order and that the presumption of its 'chaotic' character is a product of the incongruity between the order traditional requirements elicitation methods seek and that order which is intrinsic to the phenomena to which those methods are applied; that is, the order which is detectable in and through the understandings of the members of society and their ways of categorising phenomena.

They argue that requirements engineering should be concerned with the 'reproduction of social order'. To consider, that is, "the effect of a new system of social structures as suggested by the following questions: Will the new system reproduce the existing social order? Or will the order be altered in significant ways? Do the existing social structures suggest requirements that would negate the improvements expected from the new system?"

They acknowledge that "combinations of the various methods can be usefully applied to particular problems" and, overall, prescribe a 'zooming' method of requirements elicitation in which the sociological methodology is used for the fine-grained investigations of issues which have already been determined on the basis of other methods, for the sociological methodology is 'expensive and requires careful preparation'. In other words, what is required is the scoping of techniques.

While agreeing in principle with the need to scope methods, as well as agreeing with the injunction that the socially organised character of settings needs to inform certainly CSCW system design, this paper is indicative of a two problems which we have already alluded to and which we shall discuss in more detail later.

- the comprehension of sociological arguments and the relationship between different sociological positions is more complex than is made apparent in perfunctory summations and comparisons of them. For example, Goguen and Linde exhibit serious confusion about the manner in which ethnomethodology understand issues about the reproduction of the social order. They also seem to assume that the concern of ethnomethodology is

with 'subjective reality' whereas its concerns are with the inter-subjective character of social life.[14]

- it also seems to assume that turning requirements engineering into a species of 'social engineering' will solve the problems of the former. However, as we have already suggested, it may simply add a multiplicity of unsolved and unresolvable problems.

To repeat the point we made with respect to social-cum-philosophical theory, the invitation to take the requirements of the 'social order' into account is to multiply the complexities, ambiguities, and difficulties of the issues involved in the designer's task.[15]

Another claim that is made is that requirements methods make assumptions 'about society' that are 'implicitly embedded' in these schemes and methods (Bickerton and Saddiqi, 1993). Thus, there are 'unitary hard' methods which make assumptions similar to those of sociological functionalism which attempts to provide essentially rational and general explanations of social life. These 'hard methods' assume that "the organisation and its information technology are destined to achieve a particular set of *functional prerequisites* such as profitability, survival and stability and these (often enunciated as objects) are set by management." By contrast, the 'unitary soft' approach "assumes that there are different world views, so it is impossible to obtain a single objective statement of a system's purpose." The 'dualistic approach' assumes that society has a fundamental infrastructure that exhibits a conflict of interests between those who own the means of production and the workers who are controlled and exploited by the management in the interests of owners and shareholders. 'Critical social theory' alleges that the "social world is seen as self-structured, unfolding and full of many contradictions between many groups" and "within this framework requirements engineering would be seen as an effort within which groups work to pursue their own ends."[16]

Why Sociology is not the Answer

The main problem we have with characterisations such as the above is that they are not only sociologically impoverished but offer no clear guide to system design. It is not clear, for example, in what sense system designers, implicitly or explicitly, 'embed' assumptions about society in their methods. The adoption of a purportedly sociological perspective often involves a zeal for ascribing ideological presuppositions to people, with the consequent risk of 'reading in' much more (and

---

14 These will be detailed in a forthcoming COMIC working paper on ethnomethodology and the idea of organisation.

15 Elsewhere Goguen (1993) makes the point that 'traditional Requirements Engineering methods tend to ignore social issues' such as the fact that 'organisations have their own normative concepts ("myths") which do not reflect what actually happens, but rather reflect members' beliefs about that should happen". The notion that a normative order is a 'mythology' is hardly a promising basis for understanding the issues here. Further, the relationship between beliefs and action, the role of normative concepts in organisation, are all issues which, in sociology, are radically contestable.

16 This seems to assume that 'groups' have unproblematically unified and identifiable ends.

often, much more sinister) meaning into their activities and products than need necessarily be ascribed to them. Such zeal, furthermore, is often fairly unreflective and does not usually involve the consideration of whether some other, quite different, configuration of presuppositions might equally well, may even better, fit the data which is available.

Turner (1992), for example as we have illustrated earlier, has discussed the way in which many contemporary practices and products are interpreted by sociological theorists as expressions of the 'modern project' and agrees that, in fact, the developments concerned may, historically, have derived their impetus from that project. In the way that Weber (1930) argued that the attitude to work which he called 'the spirit of capitalism' originated in the religious beliefs of Calvinist and other Reformation sects, but then dissociated itself from them when it no longer needed the justification they initially supplied, so Turner argues that the success of the ideas set in train by the 'modern project' has become sufficient motivation and justification for their pursuit such that they are now quite detached from that original project. For sure, those who engage in system design and development have all kinds of assumptions in their methods, mostly to do with engineering the design. Whether these amount to assumptions 'about society' makes doubtful sense. Of course, with ingenuity one can 'find' a theory of society in almost anything, a glass vase, a mathematical proof, a book, a cigarette lighter, and so on. The warrant for, and purpose of, so 'finding' is, however, elusive, and the basis for the 'finding' often tenuous.

More to the point, however, is the prominent place given in the above characterisation of the social theories 'embedded' in requirements methods are the *a priori* assumptions made about the nature of social groups and the extent of a 'general consensus' over values and ends within groups.[17] In the case of the 'dualistic approach', for further example, the category of 'interests' is not an ordinarily empirical notion since 'interests' may obtain 'objectively' without regard to the expressed avowals, wishes and desires of those to whom the interests are ascribed. The utility and legitimacy of such ascriptions is not, in any case, a matter of general agreement within the domain of 'social studies of science and technology' (cf. Woolgar, 1981). Further, the claim that "requirements engineers have a choice, as system's objectives can be collected from the two conflicting sides, profit for management and working conditions for labour" is not quite what it seems. The capacity to confront this choice and to determine the way in which it is manifest is scarcely available short of accepting a great deal of the prior theoretical apparatus which will enable one to discount 'empirical' evidence of, say,

17 Many of the sociological theories discussed are travesties of what the theories are claiming. Particularly prominent in this respect is the account of Functionalist theories and the claim that it makes an *a priori* assumption that there is value consensus, that it presupposes that social organisation must involve universal ends. Functionalism, however, makes an assumption, which is functional in the mathematical sense, that the degree of social conflict is inversely related to the extent to which persons share common purposes. It does not postulate that in any actual social system there must be a uniformity of objectives. Empirical cases may or may not be marked by value consensus and may be so riven by conflict that we would hesitate to call them 'units'.

complementarity in the interests of management and labour as not 'really' or 'ultimately' complementary.

What is worrying here are the 'essentialist' assumptions that, for example, social relations are *essentially* conflictual. From the point of view of the system developer it would be ill-advisable to make an *a priori* determination of the extent of agreement or disagreement in purposes or understanding among the members of any group or organisation. It would be unwise, to pursue the point, that there is any *necessary* agreement among them or, for that matter, any *necessary* disagreement. While it would be cavalier to disregard the possibility that an information system for a complex organisation may require some awareness of the complexities of that organisation, including the possibility of discrepant outlooks and serious conflicts, it will not be corrected by an equally cavalier insistence on viewing these complexities through *a priori* and simplifying assumptions about the necessary extent and the character of divergence.

A more serious attempt to apply a systematic conspectus of sociological standpoints to system development is Hirschheim and Klein's (1989) use of Burrell and Morgan's (1979) classification of sociological approaches, though this, too, suffers from problems attaching to the latter's not altogether successful scheme for comprehending the underlying sociological arguments.[18]

Hirschheim and Klein's analysis is closely akin to opening Pandora's Box in that it encourages a view that the system developer must become, of necessity, a true Renaissance creature capable not only of handling the complexities and technicalities of system development work but also capable of comprehending and choosing among diverse, and tendentious, sociological theories and philosophical profundities. Their classification of sociological approaches depends upon ontological and epistemological claims, that is, the "attitudes adopted toward reality and how to obtain knowledge about it". It is these attitudes which are built into particular development approaches and, as a consequence, "important social consequences result from applying a particular systems development approach".

The Burrell and Morgan scheme was designed to provide a systematic taxonomy of approaches to the understanding of sociological theories of organisation in terms of the theoretical approaches which underlay them, and their analysis complements and informs Morgan's subsequent description of the various metaphors in terms of which organisations have been conceived.[19] Sociological approaches are classified in terms of 2 dimensions: subjectivist-objectivist and order-conflict. These, when mapped onto one another, yield four sociological paradigms as set out below in Table 1.

- The objectivist position seeks to apply "models and methods from the natural sciences to the study of human affairs". The subjectivist, on the other hand, "denies the appropriateness of natural science methods for studying the social

18 Bickerton and Saddiqi (1993) also make use of Burrell and Morgan (1979)

19 Morgan's work is described in the Deliverable for Strand 1

world, and seeks to understand the social world by delving into the depths of subjective experience of individuals". (Hirschheim and Klein, 1989).

- The order-conflict dimension distributes sociological theories in terms of the respective emphasis which they place upon the extent of inherent conflict in society. The order, or integrationist, view, emphasises a social world characterised by order, stability, integration, consensus and functional co-ordination, while the conflict view stresses change, conflict, disintegration and coercion.

	ORDER	**CONFLICT**
SUBJECTIVIST	social relativism	neo-humanism
OBJECTIVIST	functionalist	radical structuralism

Table 1: Sociological Paradigms according to Burrell and Morgan's Scheme

While such a classification may serve a heuristic purpose it is, unfortunately, seriously misleading and oversimplifying of the nature and range of available sociological positions. The 'order' view stresses the degree of order, stability, etc., relative to a Hobbesian 'state of nature' but this is hardly the same as saying that it denies the change and conflict in actual human societies. Similarly, the 'conflict' view can hardly sensibly claim that there is *exclusively* conflict, coercion, etc., in society.

Nor does the 'objective/subjective' dimension provide a satisfactory characterisation of sociological options. Ethnomethodology, for prominent example, precisely takes exception to this choice. Its emphasis is upon the intersubjective character of social reality which dissociates it from 'relativist' inclinations associated with individualistic subjectivism. The beginning fact for ethnomethodology is the availability of a social world that is a social-world-known-in-common and which has 'order-through-and-through' (see for example, Garfinkel, 1967; Button, 1992). Its concern is not with the degrees of cooperation and conflict between or within social units, but with the orderliness involved in the pragmatic conduct of the affairs of daily life, the orderliness involved in 'getting things done', the orderliness that is involved not in 'reproducing the social order', as Goguen and Linde (1993) have it, but simply in carrying things out 'once again', doing them for 'another time', and so forth — activities which may be cooperative, but may also be conflictual, for ethnomethodology does not see social relations as having any essential nature, be it reciprocally beneficial or unilaterally exploitative. Talk of 'social reality' in this context refers to the ways in which members of society go about their affairs and the ways in which such conduct is based on presumptions about the *bona fide* socially sanctioned facts of life which are the same for 'everyone'. Ethnomethodology specifically contrasts its concern with social reality as 'intersubjective' with one which treats it as 'subjective', not least with the purpose of dissociating itself from the kind of polarisation built into the Burrell and Morgan schema (cf. Bittner, 1973).

These matters apart for the moment, Hirschheim and Klein undertake an ingenious application of the 4-Paradigm model to different conceptions of system development, setting these out as a set of 'stories' emphasising the implied role of the developer. The four conceptions are:

- the analyst as systems expert
- the analyst as facilitator
- the analyst as labour partisan
- the analyst as emancipator or social therapist

Each story is composed out of the ontological and epistemological underpinnings and then applied to system development in terms of conceptions of 'key actors', 'narrative' and 'plot' that are entailed. Here we will select two for illustration and discussion: the 'analyst as expert' and the 'analyst as facilitator'.

The 'analyst as expert'

The 'analyst as expert' approach aspires toward an 'objective' specification of system requirements using 'instrumental rationality'. This serves the purposes of management who are assumed to be the source of legitimate demands. The approach assumes that there is one identifiable reality. The key actors are managers, system developers and users. The managers specify the objectives, the system developer realises those objectives and users operate with the system to achieve the organisation's objectives. The 'narrative' associated with this approach is that information systems are "developed to support rational organisational operation and effective and efficient project management" with requirements specification building on the "notion of a manifest and rational organisational reality" and employs, it is claimed, a "naïve realism". It is positivistic and functionalist in its assumptions.

Rationality in this approach relates to choosing the best means for achieving given ends and the "systems development approach suggested by this story attempts to follow the scientific method". However, it makes the "issues of power, conflicting interests, and system goals appear to be largely outside the domain of the system developer". The authors go on to point out that a large number of systems have been built using these tenets. Against this they also point out that "there is an implicit assumption that ends are agreed" and, through this, "legitimation can become little more than a hollow force or thinly concealed use of power". Its major failing is its association with functionalism.

The 'analyst as facilitator'

The 'analyst as facilitator' approach "recognises that knowledge about human means and ends is not easily obtained because reality is exceedingly complex and elusive". There is no single reality, only different conceptions of it. Business, for example, does not deal with an objective reality but one that evolves. The attempt to discover economic laws is one way of trying to make sense by imposing a possible order on experience. No one has a privileged source of knowledge and the role of people in shaping reality is unclear. What they experience as a free choice may simply be "a reaction induced by enculturated habits or circumstances". The role of the developer is to interact with management to find out what type of system

"makes sense, but there is no objective criterion that distinguishes between good and bad systems. It all depends upon what parties come to believe to be true". The developer must work from within the user's perspective to help them find their preferred views, easing the transition from one view to another and alleviating possible impedances to change. This means that "any system that meets with the approval of the affected parties is legitimate". Achieving consensus or acceptance is critical. "Systems cannot be designed in the usual sense, but emerge through social interaction."

'Key actors' are 'users and system developers'. The former are the organisational agents who interpret and make sense of their environment. The 'narrative' is of information system development as creating new meaning, and its effectiveness resting on its "ability to help users better understand the currently accepted conventions and meanings". However, the narrative has no plot because the "social environment is under continuous evolution, no particular rational explanations can be provided to 'explain' organisational reality". The assumptions are anti-positivistic and that reality is not given and immutable but a 'social construction'.

The major claim of this approach is its assertion that reality is too complex and confusing to be captured by simplifying models. The involvement in social interaction "produces unique experiential knowledge" in which the notion of rationality plays little role. Developers act rationally if they simply accept the prevailing attitudes and values, remain consistent with existing opinion and implement any changes in ways that do not disturb or threaten social harmony. The role of the system developers becomes that of 'facilitator' "who helps stimulate reflection, cooperation, and experiential learning". Systems development itself is not as important as the way in which it is achieved. The kind of systems this approach produces stimulate creativity and sense making though not as a means to achieve any specific or wider benefits. Finally, "this story suggests that all is relative: acceptance is the only thing that matters" and, therefore, "because of its relativist stance, it is completely uncritical of the potential dysfunctional side effects of using particular tools and techniques for ISD. Different products of different systems development are simply viewed as the result of differently socially constructed realities".

As indicated, it is not our intention to present accounts of all four 'stories' identified by Hirschheim and Klein, but use the ones we have just summarised to locate what are some fundamental problems about this way of proceeding.

The first point we want to make is one which is highly germane to the kind of issues discussed earlier, namely, the association that is established between certain kinds of system development techniques and presuppositions about the nature of reality, including social reality. The practical deployment of any one or another of the requirements techniques does not depend upon the adoption of the associated epistemology or ontology. The techniques, for one thing, have been regularly used in conjunction with one another and, as we have suggested before, there is no

reason to suppose that the problems of system development represent a unified set to be resolved by the adoption of a single philosophy of approach.

Further, even though a particular technique may have been inspired by certain assumptions, it does not follow that the application and effectiveness of that technique is only possible as an implementation of those assumptions, or that the practical benefits of the technique necessarily require its use in conjunction with the framework it was initially designed to fit. There is no reason why the system developer should be either an expert or a facilitator to the complete exclusion of one or other function. With respect to many of the technicalities of system design the developer is likely to be more expert relative to those for whom he, or she, is designing the system. The designer may even be an expert in the ways and arts of facilitation, but with respect to the domain for which he or she is designing, it may be that those for whom the design is being provided are the experts. They will certainly be interested parties and some facilitation expertise on the part of the designer required. The techniques of formal modelling may, in certain phases of the design process, be indispensable whereas in other areas they may be inappropriate.

Much more fundamental, however, is the issue concerning the attempt to derive direct practical application for what are general arguments about epistemology and the nature of social life. There are some difficulties explaining clearly and simply what we mean here, but the example of the 'social constructions' approach associated with the 'analyst as facilitator' might help clarify some of the obscurities.

Design as Social Construction

The account we are given of the 'analyst as facilitator' makes it seem that a particular kind of activity comprises 'systems development as sense making'. That there are certain specific procedures in system development which constitute 'sense making'. However, the notion of the 'social construction of reality' in the conduct of social affairs is comprehensively general. It claims to provide a way of understanding the organisation of courses of action, whatever these are. In which case, the conduct of the 'analyst as expert' would constitute 'sense making' every bit as much as that of the 'analyst as facilitator'. The claim about the 'social construction of reality' is a claim about a generic social process such that any instance of social behaviour can be understood in these terms. Its analytic objective is to understand the way in which persons go about the construction of social reality in the course of whatever affairs they are carrying out.

What we are raising here, and an issue which is relevant to the need for 'real world' descriptions of work activities for CSCW, is the perpetual difficulty in sociological theorising, namely, the failure to recognise that the 'epistemological' and 'ontological' stories are redescriptions of activities already described in other terms. Though not the only person to put this argument, Schutz (1962) talks of sociology as a 'second order discipline' in that the apparatus it employs differs from that employed in the natural sciences. Sociology's theoretical constructs describe phenomena which are, themselves, self-describing, phenomena which are already described in terms of socially constructed types, phenomena which consist,

in part, of the very system of socially developed and organised typologies which are used to describe them. Thus, and for example, one of the other stories of Hirschheim and Klein concerns the 'analyst as partisan', a story about the conflict between 'labour' and 'management' and told in terms of the relations between 'managers' and 'workers'. Neither of these notions are the contrivance of social theory, in this case Marxist theory which the 'labour partisan' view embodies. They are notions which have arisen as part of the organisation of work in societies such as ours, and not notions which are only incidentally used to describe the organisation of work but which might otherwise be substituted for by other, more theoretically preferable notions. They are notions which are used to describe the organisation of work as part of the doing of that work. In other words, the notion of 'manager' and that of 'worker' have developed as part of the practice of organising work in our society and is, indeed, used in the carrying out of such work. The members of organisations orient to each other as 'managers and workers', as 'fellow workers', have expectations about each others' respective behaviours, and so forth. The Marxian story, as do many sociological stories, revolves around the supposition that these categories work in just these ways, that workers come to perceive themselves, for example, as exploited, seek to oppose the attempts of their managers to control them, and so on.

It is important to note that this is not a complaint against Marxist theory that it depends upon such presupposed categories. The Marxist case is an example of the general point which is that all sociologies, if they want to speak of the phenomena of social life, are compelled to draw upon the 'typologies' which are employed within the society and to formulate their phenomena in those terms and, hence, the need, as we have noted earlier, to attend to the phenomena of daily life.

This argument does not deny that the sociological theorist can construct specialised 'sociological' terminologies, but merely expresses a concern that the character of sociological typologies involve the reconceptualisation of the socially supplied typologies current in the society and which are constitutive of the phenomena being characterised. And it is just this problem of keeping clear the first order descriptions of what developers do and sociological, second order descriptions which is a bugbear of so much of sociological description and which afflicts Hirschheim and Klein's account in fundamental ways.

Finally, such an approach does not actually involve, nor does it encourage the study of design in practice for it is based on a reading of the literature about design. If one were to take Hirschheim and Klein's scheme seriously one would have to test out the adequacy of those text based descriptions as portrayals of actual designers and as analyses of the dynamics of their practice. In fact, as our own studies of designers show in Part III, many designers are 'methodological pragmatists' in their orientation to the day-to-day tasks they are faced with in the course of design and development.

Summary

One of the things we have been trying to demonstrate in the above is how some of the criticisms of the Waterfall Model have, to a considerable extent, sown confusion rather than clarity. Part of the responsibility for this is the attempt to incorporate social science, particularly sociology, into the design process. The result has been to bring in many of the problems of sociology to system design without very much improvement in clarity and without giving much in the way of specific and practical assistance.

One of the main problems is that much of the social science effort seems relatively unaware of the problems of designing and building large scale systems; unappreciative, that is, of the software engineer's problems.

This is not to say that sociology is irrelevant to system design; on the contrary. What it does argue is that finding an effective and appropriate place for sociology so that it informs system design has yet to be worked out. Indeed, in this respect we discuss at some length, in Part III, ethnography which has recently been emphasised as one of the more important methods through which sociology can inform system design. However, as we shall argue, it is not a panacea though it surely has a role to play in the system design process.

Nor are we saying that some of the issues identified in the above critiques are entirely without point. Some of the complaints against formalism and representation, for example, do have relevance to the requirements process and what designers hope to get from it. We also think that there is some mileage to be gained from appropriate sociological studies of the design process itself (Sommerville *et al,* 1993) and we report some materials derived from such studies again in Part III.

Sociology is certainly not the answer to the problems of system design, for whilst its general characterisations of the process of design and its problems may be imaginative and challenging, they are very far from having direct, unambiguous and practicable implications for that process and its problems. It is important to bear in mind that preferred sociological approaches should not be taken too much at face value for, within their home discipline, they are invariably controversial and their empirical and practical applications are debatable. It is not, we emphasise, our assumption that the approaches we predominantly draw upon here — ethnomethodology and ethnography — are themselves exempt from the above strictures.

In the following section we attempt to provide some perspective on the Waterfall-type Model by relating it more closely to issues in the evolution of system design and development in an effort to bring software engineering problems into sharper focus.

Rehabilitation of software development models

What we propose to do in this part of the report is review some of the contexts and the issues which informed the development of the Waterfall Model and allied approaches. Our aim is to provide a perspective in which to understand the nature of the problems it was intended to solve in order to obtain a clearer sight of what the issues of system design, and especially CSCW system design, might be.

It is important to recognise that many of the inadequacies of the Waterfall Model were already recognised by software engineers and did not originate in the kind of critiques we have reviewed earlier. Social scientists do not have a monopoly on criticising the design process. As Pressman (1992) remarks, "over the past decade criticism of the paradigm [of the classic life cycle model] has caused even its active supporters to question its applicability in all situations". However, the immediate purpose is to review some of the circumstances which led to the development of the Model.

Industrialisation and system development

Machine design and development has long ago moved from the era of the 'inspired tinkerers' (Landes, 1972) who built the first machines of what became the Industrial Revolution**.** As was indicated in the Introduction, the needs of industrial scale development and production for a more effective and efficient coupling of the human and the machine, as well as the sheer complexity of the machinery itself, have all served to make design a more explicit and self-conscious, not to say essential, stage in technological development; a stage involving no longer just the single-minded visionary but teams of highly trained professionals.

Computer system design is no exception to this trend. Computer system design as now practised is certainly remote from the stereotype of the 'inspired tinkerer', the solitary obsessive programmer who was the only person who 'knew what was going on'. Nowadays, it is commonly the case that the programmer is a professional member of a large team of highly skilled professionals of varying disciplines who are involved in the design, the development and the production of the hardware and software of such systems. As Bannon (1993) points out:

> "The simplistic idea of designs materialising 'out of the blue' more or less completely formed in the head of a lone designer has given way to understanding the work of a design team who engage in a fine-grained analysis of the work needs of specific people in a specific context, and the slow and laborious process of developing a design from a tentative first sketch through to a fleshed out design model that can be prototyped, tested and refined in an iterative fashion, before the process can become fully industrialised."

In the initial phase of computing, software development had much of the character of a 'cottage industry' (Buxton, 1978). To the extent to which programmers and system developers became part of corporate enterprises they tended to be managed much like academics or researchers by a kind of "responsible autonomy strategy which may be identified as 'management by neglect'" (Friedman, 1992). High priced, low capacity and low reliability hardware

dominated, and to the extent to which managements had requirements these tended to be for systems which were fast and efficient. Programmers developed their own tricks and techniques for taking short cuts to improve performance. Creativity was encouraged and, for many, it was programming's 'golden age' The approach was to take a small group of highly qualified people who wrote largely undocumented code and who maintained the system once built (Bergland, 1981). At the time this arrangement worked reasonably well but as the hardware constraints fell away and the cost/performance ratio of CPUs and memory increased, it became less and less appropriate.[20]

Around the late 1960s and early 1970s the key issue was the software productivity gap (Brooks, 1975). The automation of software production failed as a realistic option as more complex and large-scale system development came on stream. The best known strategy was to attempt to increase productivity by "creating finer divisions of labour by separating analysis from programming and dividing programmers and programs further so that most ended up concentrating on coding small modules using structured programming methods" (Friedman, 1992) with objected-oriented development methods as the apogee of the solution of management problems through design methodology. The point is that the production of software had to face a situation where it was no longer satisfactory to rely upon the 'local guru' but had to be provided with an organisation where there was little or none before.

It is in light of this that one can better understand the point of the Waterfall Model which was an attempt to cope with the changed circumstances of software production and use. It attempted to provide a structure where there was none before, organise very large projects with a large staff, over long time scales, with the extreme likelihood of relatively high rates of staff turnover, provide records where, again, there had been none before, all in an effort to 'industrialise' what had previously been a highly individual activity. Brooks (1975) expresses the point well:

> "Many of the classic problems of developing software products derive from this essential complexity and its non-linear increases with size. From the complexity comes the difficulty of communication across team members, which leads to product flaws, cost overruns, schedule delays. From the complexity comes the difficulty of enumerating, much less understanding, all the possible states of the program, and from that comes the unreliability. From complexity of structure comes the difficulty of extending the program to new functions without creating side effects. From complexity of structure come the unvisualized states that constitute trap doors.
>
> Not only technical problems, but management problems as well come from the complexity. It makes over view hard, thus impeding conceptual integrity. It makes it hard to find and control all the loose ends. It creates tremendous learning and understanding burden that makes personnel turnover a disaster."

---

20 During the 1960s a number of studies showed that the cost-effectiveness of central processing and memory in new computers increased by a factor of 10 every 4 years over the period 1953-1965. See Friedman (1992).

The point we are making is not to defend the Waterfall Model come what may. Rather it is to try to understand what it was that it was trying to do in the circumstances when it was proposed. In sum, it arose when system design and development had to move toward industrial production with its own requirements, such as:

- the need to integrate various skills and activities over a relatively long period of time;
- the need to coordinate skills and activities so that they are efficiently directed toward the same goal;
- the need to cope with turnover of staff, many of whom may not have been involved in the project at its inception;
- the need produce the system within time and budget constraints.

The Waterfall Method was a prescriptive device for co-ordinating large teams and in this sense was very much a managerial method for organising effort in a situation which required organisation. In which case it is misguided to criticise the model because it does not provide an accurate description of the design process or, for that matter, that the limitations of representation can be transcended so that it could become an accurate depiction of the design process in general. The model is not intended as an empirically oriented description of the design process. It is a prescriptive procedure for the rationalisation of design work.

Furthermore, it was a model which was itself designed to meet the problems encountered on large projects, but it is often debated as if it is generalisable to all projects regardless of scale when, in fact, it is much more specific in its application. For small teams, for example, it would be over rigid, over elaborate and involve considerable organisational overheads. In its time the Waterfall Model was an effort to solve engineering and production problems within specific circumstances of system development. Critics, however, have tended to see it as a solution to problems it was never designed to solve with the result that debate has all the flavour of the aftermath of Pandora's wilfulness.

Design constraints

Another point, and perhaps an obvious one but it nevertheless needs stressing, is that design takes place within numerous constraints, many of them beyond the control of the designer and the software engineer.

Among the many factors that can affect the success or otherwise of system design are the many imponderables other than those to do with engineering narrowly conceived, important as these are.[21] To briefly summarise these we can begin with the fact that system design will almost inevitably involve the redesign of traditional work practices and, depending on how resistant these are to change, they can have a significant bearing on the success of a system no matter how well

21 A standard finding in the history of technology is that the technological innovation is infused through and through by social, political and economic factors. See, for example, the collection MacKenzie and Wajcman (1985)

engineered it might be.[22] Further, a system's operating environment is never constant with the consequence that it is difficult to determine with any accuracy what effects are due to the system and what due to other factors. A poor system, for example, may well provoke a serious deterioration in effectiveness but, depending on a host of other factors, such as the dedication and craft pride of its operators, may have a negligible effect on operating efficiency.[23] A good system may well be so welcomed by its users that its design capacity is quickly exceeded, so turning it into a poorly anticipated system.

Systems development also takes place within a particular socio-economic-cum-legal environment. Many European countries have legislation which requires that the representatives of different interests be involved in projects. For example, Danish employers initiating a large scale system development must first obtain the employees' acceptance of the project goals and the expected consequence for work organisation (Grønbæk *et al*, 1993). Similarly, many European countries place restrictions on software products that use databases containing personal information. Labour supply, market conditions, the availability of investment finance, and many more, are among the continually changing economic factors which bear upon system development and its successful outcome.

Similarly, and again an obvious point that we are stressing, by far the great bulk of design takes place within commercial organisational settings which means that other constraints are inevitably brought to bear, including financial, organisational, the availability of the right skills at the right time, marketing considerations, among many others; ingredients whose admixtures can produce very different recipes despite the effort that goes in to anticipating and controlling the uncertainties that they generate.

Among the uncertainties, given the pace of development within computing, is technological innovation itself. The popular myth that technology is simply applied science has been disposed of long ago and whatever truth there may be in such a picture it does not apply very much to computer science. Pure and applied science, with the possible exception of chemicals and pharmaceuticals, are independent if related endeavours and the nature of the relationship is complex and varied (cf. Layton, 1978 and Gibbons and Gummett, 1984) The implication of this is that it becomes extremely difficult, if not impossible, to predict, let alone control, what impact basic discoveries in the relevant sciences as well as those in technology will have upon the best laid plans of corporations. While the history of technology is replete with stories of how the direction of technological innovation and development was manipulated by business corporations, it equally well demonstrates that they can make mistakes, back the wrong technological horses, as

---

22 See, for example, Ackroyd et al (1992) for a discussion of a variety of police information systems which befell this fate. Of course, the archetypal example of this are the Luddite disturbances in Britain in 1812.

23 A good example is the case of UK air traffic control where the introduction of poorly designed equipment had little or no effect on performance. However, there are specific features of this occupation which enabled them to resist using it. See, for example, Harper, Hughes and Shapiro (1991), Harper and Hughes (1992) and Hughes, Randall and Shapiro (1992).

well as being caught short by the completely unexpected. Hughes (1987), for example, speaks of the 'reverse salient' in technological development; a concept which points to the leads and the lags in technological development which, while having a bearing on technological innovation, make the business of prediction in this context hazardous.

As Shapiro (1993) reminds us, 'design is not an exact or an absolute matter, but is simply about "doing one's best..." '. Design is, to borrow a notion from Simon (1960) though with different implications, a 'satisficing' process concerned not so much with achieving perfection, but getting the best outcome within the constraints of functionality, cost, time, context, and so on (Shapiro, 1993). And even with the best will in the world, it is not always guaranteed that even the best designed of systems will achieve their aims. The productivity of North American office workers, for example, has been growing very slowly in the 1980s during a period of intense computerisation (Dunlop and Kling, 1991; Kling and Dunlop, 1992). Although it would be hazardous to infer too much from an instance such as this, what is clearer is that system design is by no means a straightforward business and it is equally difficult to judge and assess the worth of such systems even when well designed.

Managing the development process

The foregoing remarks are, of course, relevant to any domain of design, from toys to photocopiers, from cars to aeroplanes, from games computers to large scale mainframes, and generally express, as we have indicated, some platitudes about the conditions under which design and development has to operate in the world beyond the research environment.[24] Nevertheless, despite their being platitudes they are important reminders about the importance of the requirements capture and specification process as crucial in the system design process.

In addition, it also raises the important issue of the relationship between requirements and implementing them in a large scale system development process. Complex system development has become a process that requires effective management if it is to be done at all. However, life cycle modelling is not entirely, and not just, a management device. It is very much a design and development methodology that emerged to deal with the very practical problems of dealing with large-scale system design within a specific constellation of technical and human resources. The methodology was developed to obviate the largely unstructured, even bordering on the chaotic, approach to system design and development in which everyone threw themselves into the work without checking or co-ordinating their contributions with those of others, without providing any documentation of what had been done and who was doing what, tackling problems in non-standardised ways without any clear conception of what the team was about and the part that particular tasks played in the overall structure, and without any procedures

[24] This is not to say that the research environment is without its problems, only that they are not those of industrial system development.

to check whether the project was on schedule, which problems had been solved and which were awaiting solution, and so on.

None of the above should suggest that the Waterfall Method, or any of its variants, is beyond criticism. Our objective in 'rehabilitating' the model, albeit in a minor way, has not been to advocate its use but simply to achieve a better understanding of its nature and force, in order that we be better able to see what kind of problems it was designed to solve and, through this, better able to see what the problems of any large scale system engineering involve.

As we have already indicated, the Waterfall Model has come in for practical criticism in the sense that it has been superseded by more flexible development processes more suited to the variety of engineering needs that software development engenders. The model's own attractions may, themselves, have been shaped by practicalities, for in a time prior to 1970, when (according to Agresti, 1986, Musa, 1983) there were more programmers than computers, it made sense to husband the machine resources by careful paper and pencil planning, which would also permit less problematic and expensive revision than would the reworking of software. What the Waterfall Model recommended was the need to proceed systematically deferring implementation as long as possible in the development process. The injunctions can be summarised as follows:

- analyse and design before becoming prematurely committed to solutions which may prove expensive to rectify if wrong
- 'front load' as much of the design work as possible to make sure that things are right before making firm commitments which could ramify expensively

As Boehm's (1976) original article suggests, the rationale for the life-cycle model was economic in that the relative costs of fixing errors on large projects increased exponentially depending on the stage at which the error was detected. Thus, it was argued that successful software development required the successful achievement of sub-goals at each phase of the process. The Waterfall Model was directed toward organising the design team, using the resources and tools of the time, by disciplining the work process. Since the claim about the cost of fixing errors is not generically true, in the sense that a high degree of modularity in the design may greatly reduce the extent to which errors propagate through the system, it may even be argued that the Waterfall model was also directed toward solving problems that arose from the use of those programming languages which did not scale up well.

The Waterfall Model reflected the time in which it evolved and, since then, there have been changes which enable a reassessment of the method. There are now many more computers than there are programmers and access to them is no longer limited to those with highly specialised skills (Musa, 1983). A wide array of support tools are also available and capable of spanning several of the phases identified in life-cycle models, so challenging the usefulness of its partitioning into stages of design. Since then there have emerged a number of different versions and methods for the design and development of large scale systems most of them

aiming for greater flexibility in the design and development process, not least because of the variety of constraints under which the process takes place. The availability of such computing resources on the scale that Musa mentions means that the development model provided by the Waterfall can, in practice, effectively be 'flipped on its back', that the process can begin with the production of prototypes. But this does not mean, however, that the design and development process can cease to be organised; on the contrary. Many of the problems of large-scale system design are generic**.** The Waterfall Model, after all, does identify the main *logical elements* of the design process, in the sense that one will, in the process of system development be involved with 'analysis', 'design', 'implementation', 'testing' and so forth, even if these are not undertaken as distinct or separately phased activities.

Alternate Models of the process

As we have pointed out, software engineers have themselves been critical of the Waterfall Model as a solution to the problems posed by large scale system development. There have been ways in which the classical Waterfall Model has been modified (in practice, if not always in theory) almost beyond recognition and also attempts to get away from that model. DeGrace and Stahl (1990) have argued that from a software engineering point of view the Waterfall Model has maximum efficiency with respect to certain kinds of projects, namely those that,

> "...put the computer at the center of things: the payroll system, the airline reservation system, the planning system etc. In these systems, human serve the machine, providing it with the input it needs to produce the results. But we are now encountering problems of a different nature where the computer is no longer at the center of things — the human is — and the machine is now acting to provide or organize information the humans need to produce results."

They argue that the projects which put the machine at the centre of things involve problems which can be defined, data can be collected and analysed, and a solution constructed. The movement toward developing systems in which the machine is now operating in service of human work activities means that the problems involved are 'wicked' ones where "the problem is only fully understood after it is solved". Wicked problems "have no stopping rules. Work stops for external reasons such as lack of time, money, or patience. Also, the problems are not solved, but only have degrees of sufficiency".

Since solutions to wicked problems are not true-or-false but good-or-bad, there are no objective, unambiguous criteria for determining definitive solutions. For example, they say,

> " "Make or buy" decisions are frequently like that. Not only is there the problem of whether or not you have enough information, but also whether or not the information you have is appropriate. Do we really have the skills to do the job or is that what we advertise? Can we really buy it for less or is that smoke and mirrors from the vendor? Do we make our decisions just to get along with other members of our team or do we make them based on the technical merits?"

Decision making with respect to the clearly defined problems, DeGrace and Stahl go on to say, is typically in the hands of a few people, and they are the ones who evaluate whether a solution is good. But with wicked problems, by contrast, "any solution after being implemented, will generate waves of consequences over an extended — virtually — unbounded period of time."

The notion of 'wicked problem' is one which was initially formulated by Rittel and Webber (1973), and identified as characteristic of planning activities: system design being, of course, very much a form of planning activity. The characteristics of the wicked problem include:

- Every solution to a wicked problem is a 'one shot operation' because there is no opportunity to learn by trial-and-error, every attempt counts significantly.
- Every wicked problem is essentially unique.
- Every wicked problem can be considered a symptom of another problem.
- The existence of a discrepancy representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem's resolution.

De Grace and Stahl conclude that the 'Waterfall model' did achieve an improvement in system development, certainly over what they term the 'code and go' shops which preceded it. However, they continue, the success of the model has been less than its enthusiasts claim for it, for, very often

> "...success with the Waterfall model is not really success at all. Either the model used wasn't really the waterfall model, or the notions that the defenders of this model have of success aren't what we developers would accept as success...a "bastardized" version was employed to make the project work."

The problem in criticising the Waterfall is, they argue and one we have come across in connection with sociologically inspired critiques, that of dissociating the evaluation of model itself from the allegedly dogmatic attitudes surrounding it. However, included among the practical problems with the model De Grace and Stahl enumerate are:

- The stacks of documentation that nobody (except a lawyer, perhaps) reads, but which, nevertheless, have to be produced. There are complaints about having to create and maintain current and new physical and logical models of the system; complaints about how tiresome it is to draw and redraw all the data flow diagrams and other graphs; all of which encourages a feeling that these are done "for show — not know".
- Formal methods, depending on how they are enforced, make it more difficult to take advantage of opportunities for improvement that may come along, either by restraining exploration or by constraining it to one small part of one phase.
- For users the Waterfall Model is not real to them in that they are not attuned to the extent to which the model requires the fixing of decisions about requirements, and hence do not see changing their minds as a problem. Nor are they aware of the consequences that such changes can have.

- Requirements are incomplete
- Its procedures take a long time
- It is very expensive to use
- It has many variations that really do not map into one another
- It encourages a gap between end users and developers
- It assumes that the 'what' can be separated from the 'how'
- It does not lend itself to proper error management and access to appropriate emerging technology.

Whilst it may be a logically valid assumption that you can separate the problem description from the problem solution it is, they argue, unfortunately, the case that "real people do not behave that way. Ordinarily, real people consider a range of possible solutions when they consider a problem".

Further, there is an error management difficulty inherent in all top-down models, including the Waterfall. One cannot predict in advance how to handle all error modes and must, therefore, have an incomplete detailed design. Difficulties here reflect all the way back up the chain. Not only error management, but certain characteristics of the hardware, such as response times, overhead for backup and accounting, traffic control, etc. cannot be known in advance if consideration of the solution is deferred until the last — and these have implications for requirements.

Thus, as can be seen, the criticisms which arise from practical involvement with the Waterfall converge with some of those originating from a sociological angle, for example, those pertaining to the bureaucratic character of the method, and the separation between developers and end users.

The Waterfall model (and/or the methods of structured analysis which are often closely associated with it) is one which is, notionally at least, in very wide use in industrial practice. But if the arguments given about the practical shortcomings of the method are correct, then it presumably must be the case that the Waterfall Model (and the procedures of structured analysis), *strictly considered* must be honoured as much, or even more, in the breach as in the observance. So much so, DeGrace and Stahl argue, that the practice which is actually involved disqualifies the operations from being considered to exemplify the Waterfall at all. They are better termed 'The Whirlpool'.

The Whirlpool

This model of the development process involves 'clumping' of the phases of the classical Waterfall model. The composition of these clumps changes several times as the Whirlpool is being used until all the phases of the Waterfall appear in the Whirlpool but do not have the same timing relationship to other phases as they do in the Waterfall model.

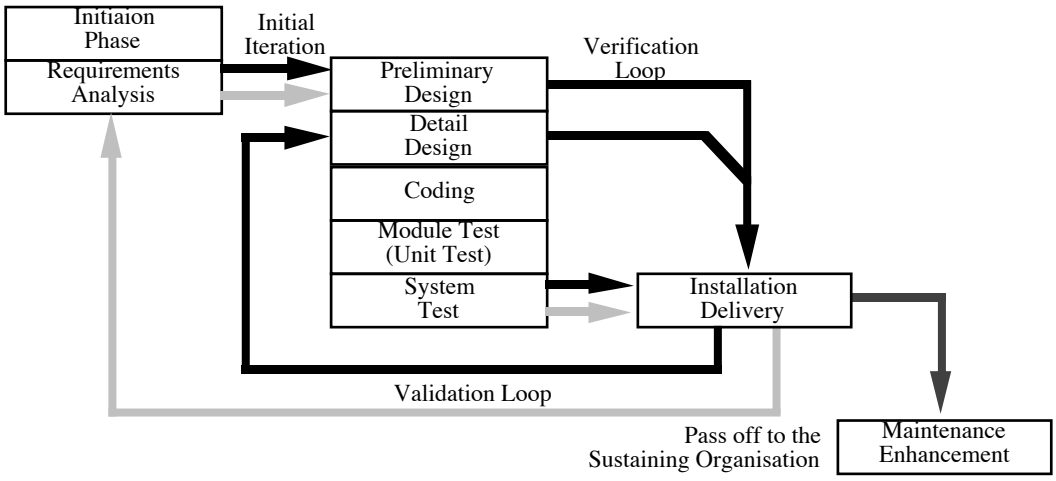

Figure 2 — The Whirlpool Model of Development

The Whirlpool can involve four major steps, each composed of one or more traditional phases of the Waterfall.

- The Initiation and Requirements phase is staffed mainly by users. There is much more interaction between the initiators of the project and the staff actually doing the requirements. The ‘what/how’ gap occurs between this step and the next.
- The Preliminary Design, Detail Design, Code and Test phases. The staff is mostly programmers and analysts, and there is much interaction between them with the various activities being conducted largely concurrently.
- Installation and Delivery.
- Maintenance and Enhancement.

However, under this regimen, and unlike the classic Waterfall Model, frequent testing allows for the identification of bugs in modules, interface problems, and error handling. Frequent iterations are a ‘verification loop’, a kind of ‘reverse engineering’ where the developing system is mapped onto the requirements in an effort to reconcile the system with the expectations of those who initiated the project. Some design and construction steps are also moved to the Installation and Delivery phase. In other words, the difference between the phases collapses as developers do different operations simultaneously. The whirlpool allows

> “...more iterations than the orthodox model would allow. And it was what people *had* to do to be effective and get the job done.’

Evolutionary Approaches

The kind of movement away from the Waterfall model in the ‘Whirlpool’ adaptation is not one which is necessarily perceived as such by those who have made the transition to it through practical adaptation. There have, of course, been more self-conscious attempts to revise the fundamentals of the Waterfall Model, particularly with respect to the reduction of its commitment to planning. The Waterfall, in relatively strict interpretation, involves the attempt to work out the design and plan

the implementation process as an early phase in the process, but there is a strong inclination to create alternatives to the Waterfall which will allow the actual system to be developed in a more 'evolutionary' way, and which might, even, recognise that the system will continue to 'evolve' even after it has been put into use (Belady and Lehman, 1976).

Thus, for example, there have been incremental models, such as that developed by the Boeing Corporation which abandons the idea of a complete system of requirements, and starts with general objectives and proceeds by translating some of those objectives into requirements and then implementing them. The cycle then repeats. Starting the development process with general objectives rather than a complete set of requirements is important because it allows requirements to adjust to changing conditions. When the requirements and the design are closely related in time, "the what/how gap is significantly reduced in size" (DeGrace and Stahl, 1990)

One of the major alternatives to the Waterfall which places great emphasis upon allowing the finished system to evolve, and to do so through interaction with the user, has been the use of 'prototyping' (Sommerville, 1992). In its most full blown form the 'prototyping' approach is intended to provide a comprehensive alternative to the Waterfall though even this does not involve dispensing with phasing altogether, for the method does require that there be a preliminary phase of requirements identification, but, the emphasis then falls upon the production of something that will operate as a model or mock up of the system being built, with the purpose of enhancing user involvement in the design/development of the system. The prototype provides a concrete realisation of the system in a form that the user can recognise as at least approximating toward the system with which he or she will have to work. The prototype may vary in the realism with which it is constructed; it may be simply a paper or screen representation, or it might be something which implements a design or maybe an existing system which is to be enhanced. The prototype can then be evaluated by the user and requirements further identified in an iterative process. The extent to which prototyping does involve the user in development can, of course, be variable, even allowing involvement to the extent envisaged by Participatory Design (cf. Bødker and Grønbæk, 1991).

Prototyping does not necessarily resolve the problems, however, for the development of the prototype may generate unrealistic expectations about the time scale on which the system will be available (the prototype having to be 'rebuilt' in order to make it genuinely functional) and the production of 'rapid' prototypes can, in fact take some time. There are, further, pressures on the developer to make

> "...compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability. After a time, the developer may become familiar with these choices and forget all the reasons why they were inappropriate. The less-than-ideal decision has now become an integral part of the system.
>
> The key for overcoming these problems is to define the rules of the game at the beginning; that is, the customer and the developer must both agree that the prototype is built to serve as a

mechanism for defining requirements. It is then discarded (at least in part) and the actual software is engineered with an eye toward quality and maintainability." (Pressman, 1993)

The Spiral Model

The spiral model was contrived by Boehm (1988) in order to combine the strengths of the classic Waterfall model with those of the evolutionary 'prototyping' approach, linking these through the use of 'risk analysis' procedures. This model is meant explicitly to accommodate a range of development models and to provide a means of choosing between them (or of combining them in a 'mix' on the basis of their appropriateness to the project's problems. The decision is to be made in terms of a risk analysis:

- "If a project has a low risk in such areas as getting the wrong user interface or not meeting stringent performance requirements, and if it has a high risk in budget and schedule predictability and control, then these risk considerations drive the spiral model into and equivalence to the waterfall model."
- "If the software product's requirements are very stable (implying a low risk of expensive design and code breakage due to requirements changes during development) and if the presence of errors in the software product constitutes a high risk to the mission it serves, then these risks considerations drive the spiral model to resemble the two-leg model of precise specification and formal deductive program development."
- "If the project has low risk in such areas as losing budget and schedule predictability and control, encountering large scale system integration problems, or coping with information sclerosis, and if it has a high risk in such areas as getting the wrong user interface or user decision support requirements, then these risk considerations drive the spiral model into an equivalence to the evolutionary development model."
- "if automated software generation capabilities are available, then the spiral model accommodates them either as options for rapid prototyping, or for application of the transform model, depending on the risk considerations involved." (Boehm, 1988)

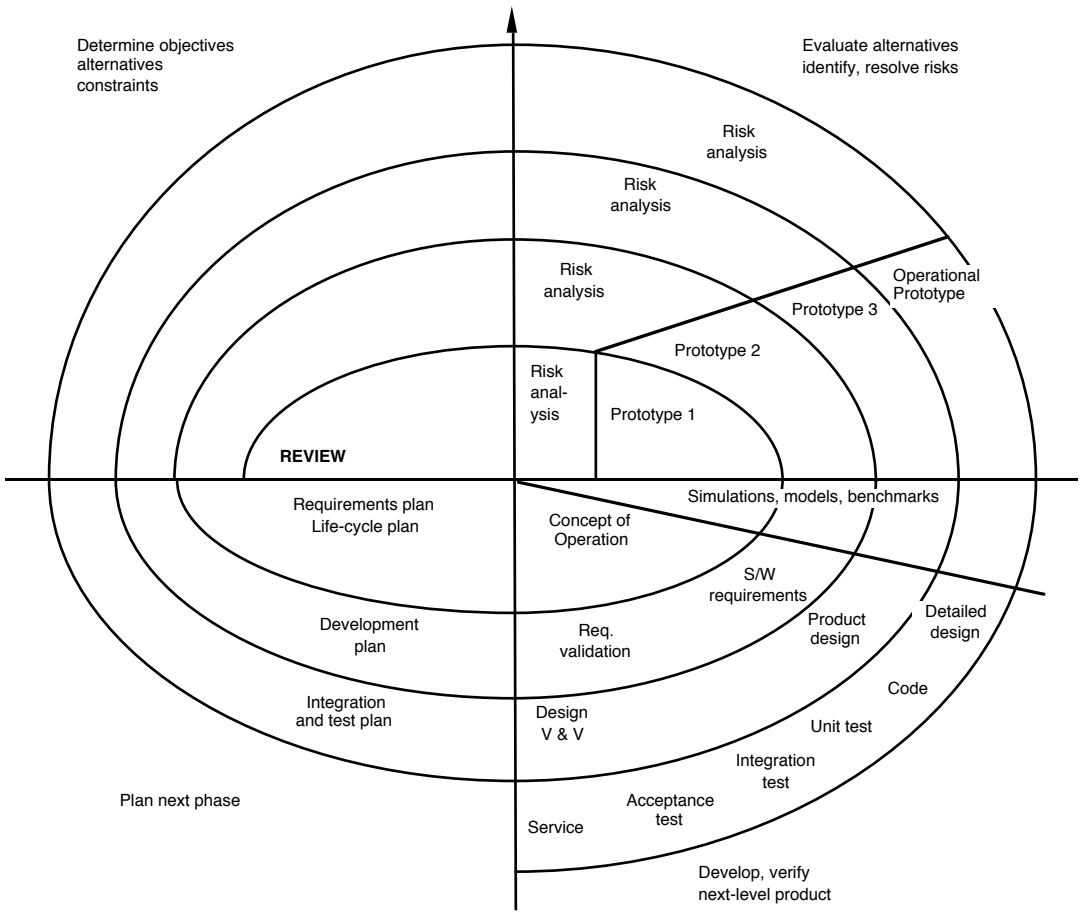

Figure 3 — Boehm's Spiral Model of Software Development

The spiral approach identifies 4 main activities within the development process:

- planning
- risk analysis
- engineering of 'next level' product
- customer evaluation

Development 'evolves' through iteration of sequences of these four activities. The underlying structure of the spiral is, in fact, the classic life style model, proceeding through from initial requirements, planning and design phases, through the elaboration of the design, validation, verification and test, through to implementation and acceptance testing. But if risk analysis identifies the identification of requirements as a critical consideration then the use of prototyping at various points along the course of the development project allows for interaction between the project team and the customer/user, with the prototypes being of increasingly elaborate kinds, eventually approximating to the operational system. Risk analysis is an important component in the model, but this cannot be thoroughly systematised and must be substantially a judgmental matter, and the capacity for judgement must be a function of familiarity with the problems of

project development. Risk analysis is undertaken does not ensure that major risks will necessarily be identified.

Object-oriented design

Object-oriented design (OOD) arose out of object-oriented programming (OOP), and thus out of developments in the design of programming languages. As so often, there are debates about whether people were doing objected-oriented programming before there was a name for it and whether or not there is really anything distinctive about it, and what its precise origins and history were. SIMULA is often cited as the language in which the concept of 'object', which is the central element in 'object-oriented' work first appeared, though it was not fully developed there. In the years since then object-oriented programming has become a widely popularised mode of programming, one which, it appears, many programmers like to use.

Advocates of such approaches often claim that their appeal lies in the fact they depend on what people in general find it natural to do, namely, to conceive of situations as composed of objects. If one conceives of a situation as comprised by distinct objects, then it is natural, and therefore easier, to decompose that situation into its constituent objects or elements The claim can, of course, be contested. Sommerville (1992) asserts that, at least as far as systems analysis is concerned, "the identification of the appropriate system objects is difficult. Our 'natural' view of many systems is a functional one and adapting to an object-oriented view is sometimes difficult. A particular problem is object identification". He argues that there is no simple formula which allows the identification of the objects or the specification of the attributes and operations which comprise them. It is, in fact, something which requires the skill and experience of the programmers and designers.

The 'object' in OOP is a combination of data and software, and it is the combination of attributes and procedures within a single self-contained unit which holds out the promise of much more effectively modularised and, therefore, reusable software. If objects are relatively self subsistent units which do not need to share information with each other, then the successful design of them will make them highly cohesive units in the program, with well defined characteristics and behaviours which ought, then, to be abstractable from the program for which they were initially designed and insertable into other programs.

The concept of 'information hiding' is commonly strongly associated with object-oriented programming. The fact that the 'object' is comprised of data and of the software which performs operations on the data means that the there is a need, from the point of view of the program's structure, for an awareness of the internal structure of the object. All that needs to be known, from that point of view, is how the object will interact with other objects in the system, and this means that both the interfaces and interactions between the objects in the system can be simplified, with clear implications for economy and correctness of design. Additionally,

simplification in the code that has to be written can be expected from the operation of 'inheritance'.

The cohesive modularity of the 'object' means that corruptions within the object do not propagate throughout the system and that redesigns of it are also contained. The simplification of the interface also means that exchanges between modules are minimised, making the structure of the system more perspicuous and errors in the design of interaction between its modules easier to detect, thus reducing the risk of unintended and incorrect interactions.

The concepts of object-oriented programming are modelled upon the modularising practices of engineering (Berard, 1993). Engineering design involves decomposition in terms of objects, the assignment of functionality to parts, and, with the reuse of parts in mind, the functionality of the parts may be modified or extended. Since the parts must interface with each other, care is taken in defining the interfaces. Objects that are well defined in terms of function and interface can be taken off the shelf. The design of the whole system proceeds in terms of the specification of the connections and interactions between its constituent objects. In engineering, the objects are instantiations of classes, and the identification of classes therefore involves the specification of the characteristics common to all members of the class, with the possibility then of 'specialising' objects within a class, as one can have a class of 'wheeled vehicles' and can then specialise amongst different kinds of wheeled vehicles, such as 'three wheeled', 'four wheeled' and 'twelve wheeled' for example. Because there are characteristics shared by members of a class, these can be 'inherited' by instances of the class. If we design a new object, which is a sixteen wheeled vehicle, this will inherit the characteristics common to wheeled vehicles, and thus, there will be no need to write code to provide it with these characteristics.

Object-oriented programming and design are distinct. There is no requirement that object-oriented design be implemented in object-oriented programming languages. The features of OOP which we have enumerated, however, are ones which obviously were inspiration for the work of OOD.

Object oriented techniques have provided tools for use through much of the software life-cycle, through requirements analysis, domain analysis, design, and programming. The pervasion of object oriented methods throughout the life-cycle is intended to increase the coherence of the development process, to ensure that those working on different aspects of the division of labour or at different places are oriented to a common and full definition of a given object, rather than operating in terms of partial and discrepant conceptions. An object-oriented life cycle is therefore required insofar as,

- the use of object oriented methods with other life-cycles and their associated tools (such as structured analysis) will merely result in incongruities and
- the life-cycle is essentially intended to deal with the problems of scale and complexity which are involved in large projects. There is no need for a methodology for a small project, but the increase in size does mean a

disproportionate increase in complexity, both in terms of the code to be written and the problems of communication and co-ordination within the project.

A life cycle model proposed on the object-oriented basis is the recursive/parallel life-cycle, which is characterised (Berard, 1993) as "analyse a little, design a little, implement a little and test a little". This recognises the reality of software development activities, in their iterations and 'overlapping', with activities at different levels of abstraction being carried on in parallel. There is no need to attempt to do all the analysis, followed by all the design, followed by all the implementation. Indeed, there is no need to think of design and analysis as starkly distinct and entirely independent operations, conducted with different tools and procedures. Nor is there need to make all the decisions about design at once and prior to implementation; design decisions are of different significance and some can be deferred, though the capacity to do this hinges upon the capacity to keep components of the design well defined, and stable, with clearly specified interfaces.

The above are not, of course, the only approaches to requirements specification and software development that are available, as we shall see in Part II. What all of them represent, just as did the Waterfall Model, is a means of organising the design and development process to achieve consistency in its production, stability in its activities, and a system that meets the specified services it is intended to provide. That there is still a lively debate within software engineering including the relatively frequent proposals of new methods and approaches, suggests that the issues are still very much open ones: issues that are not well-served by the rhetoric and offering of panaceas that too often surrounds the debates.

A Pragmatic Attitude to System Design

For us, and one of the rationales underpinning the work of Strand 2, is the need to avoid polemical polarisation of the debate over methodologies and to make pragmatic appraisals of them, ones which presume that the problems of system design are not to be solved 'at a stroke'. An approach which recognises that in system design there is a dense complex of problems involved, and that these problems are of different kinds and order, and are, therefore, unlikely all to yield to the same strategy.

The approach we will be using in the course of considering methods and tools is very much a 'horses for courses' one which recognises that the purposes they are intended to serve are diverse, and that their respective merits will, therefore, tend to be relative to only a few of the needs which must be met even within the 'requirements identification' activities in the design and development process. As things stand within the state of system design, the decisions over tools and methods will typically involve 'opportunity costs', for the alternatives are seldom clear cut, and the attractions of one way of doing things typically involve foregoing the attractions of some other way.

Taking a pragmatic attitude, or to borrow a phrase from Schmidt (1993) a 'reformist stance', to system design is not one which ignores the very real problems which it faces, some of which have been voiced within the critiques reviewed above. Instead, what it is intended to open up are what we refer to as 'calibration issues'; that is, determining how and in what ways a method can deliver what is required of it. Determining, to put it another way, what its limitations and its potentialities are. While recognising that this is by no means a straightforward task, the least it will involve is comparing methods across a range of desired objectives and circumstances and coming to some judgement as to which serves which needs best. This may well mean developing new methods as well as rejecting others.

As indicated above, such an approach does not mean denying the substance of many of the complaints raised by the kind of critiques reviewed earlier. Indeed, in some cases they raise important issues which need careful consideration, but which are in danger of being confused because of the rhetoric which sometimes overwhelms discussion. Much of this has to do with the stance taken toward the issues. One does not have to see Participative Design, to choose but one example, as a visionary vehicle for the democratisation and humanisation of work to find it an appropriate candidate for dealing with some of the problems of the relationship between designers and users.[25] The point is not that treating a method as a vehicle for the realisation of broader political goals is illegitimate, though it may encourage an uncompromising attitude to other methods, but that it opens a debate that goes beyond the practicalities of system design and development.

In some respects what we call a 'pragmatic attitude' could equally well be described as taking a 'methodological attitude' to matters of system design and development. That is, attempting to evaluate within the problems set by system design and development, how best to calibrate and organise its process in order to make it more effective. This assuredly will involve argument and debate but will avoid the drift to metaphysics which has, so far, characterised much of the debate in this regard.

In many respects the current problems of systems engineering are the outcomes of solutions to previous problems. Much more is now possible as very powerful machines become more widely available to users to who need have no special expertise. This provokes new problems of design as does the increasing variety of the settings in which computers come to have a place. If the problem is simply about the solitary individual user then much has already been achieved. Once the computer moves into the place of work and within cooperating ensembles of people, this gives rise to new problems of design as well as new challenges for the kind of systems that can be envisaged (Grudin, 1990). We have, as it were, a new 'satisficing curve' to move along.

---

25 We are not implying here that Participative Design is a direct response to the Waterfall Method or its variants. Its roots have much to do with the political culture of Scandinavia. Within HCI, however, it is often counterposed to the positivistic, formal and bureaucratic approach to the organisation of work and system design that life-cycle models represent.

The development of software engineering, elements of which we have briefly reviewed in the earlier discussion, has now reached the point where software engineers have reasonable expectations as to what can be achieved and how to achieve it. This is not to say that they are always successful. Engineers are as capable of making mistakes as the rest of us are, but it is also worth a reminder that many of the problems of system design, and the production of 'unsuccessful' systems, lie beyond engineering itself.

Nor is it the case that software engineers are complacent in the face of the problems that design poses. Their day to day work — as we shall develop in Part III — consists in dealing with such problems, trying to get things right, making the necessary compromises, dealing with the barely tractable, all within a context which is, as it is for many others, unpredictable and uncontrollable. This, after all, was very much the point of the Waterfall Model.

What needs to be avoided, in keeping with the pragmatic attitude to these matters that we are recommending, is to imagine that there is a single solution to the problems of design. That is, we need to resist the temptation to seize upon panaceas which offer the false promise that all the problems of design can be resolved. For example, there is a current enthusiasm for the idea that 'ethnography' will make a decisive contribution to the problems of 'requirements elicitation'. While we certainly see that ethnography can have value in design, we take the view that its potential should be explored warily, for the technique of ethnography brings many problems and complications of its own, as will be discussed below in Part III. Even the name 'ethnography' can be a source of difficulties, for it might suggest that it refers to a unified pursuit, but it is, in fact, the title for diversified activities, and ethnographic studies can be made on very different terms and for diversified and divergent purposes. It is not, therefore, the case that all the ways of approaching ethnographic studies can play the same role in system design. We do want to stress the point that methods are only as good as their calibrations; that is, how well they shape up to meet the tasks demanded of them in the system design process (Sommerville *et al*, 1993). What should be avoided is erecting them into ideologies of requirements engineering.

However, one of the prime targets of many of the criticisms mentioned earlier is what can be referred to as the 'engineering mentality' which is seen as a product, to put it generally, of the rationalistic, analytical, scientific, modern culture and all that this represents as the antithesis of the humanistic, holistic, synthetic culture more suited to human need. Be this as it may, more prosaically and pragmatically, there is a germ of a point of this critique which is worth some attention.

Whatever the extra-design considerations which motivate the critique of the 'engineering mentality', it suggests that many of the engineering solutions to the problems of system design are themselves the source of many of the problems of system design. Bucciarelli's (1988) depiction of engineers at work, for example, describes the ways in which, when confronted by a purely administrative problem, they attempt to resolve it as though it were an engineering problem. Thus, in the context of system design, the 'engineering mentality', to use it in a less grandiose

sense, would tend to see all aspects of the design process as susceptible to engineering solutions. It presumes, in other words, that analysis, the subdivision of elements into their smallest components, abstraction, the systematic planning of operations, and so on, will resolve all the problems of system design. Accordingly, what is required are more sophisticated formal devices for capturing and specifying requirements. The possibility that such formal devices can themselves be a source of problems is a key inspiration for CSCW (Sommerville *et al*, 1993).

However, and above all, it is important to avoid thinking that one must be either for or against formal methods as a whole. It is not necessary to be totally opposed to formal methods in order to make severe criticism of their use in any given instance. The essence of the criticism may be that the manner of their use is, at least in that case, inappropriate. The problems with 'formal' approaches do not have to be seen as originating in the inherent deficiencies of the approaches as such, but the likely result from the problem which Kenneth Burke long ago identified as 'Occupational Psychosis ' (Burke, 1935) and which is indicated in the just cited example from Bucciarelli; that is, the possessors of a skill or technique become so enamoured of it that they indiscriminately seek to answer all problems by its means. It is, of course, an affliction which can, and does, strike social scientists too.

There are generic problems in building large scale systems, problems which belong to those of engineering: disposing of tasks among personnel with the relevant expertise, keeping track of tasks and their completion, seeing that all the components fit together, making one's own work and that of other's reciprocally visible and intelligible, recording what has been done and what is yet to do, and more. That is, structuring the whole process so that the overall task of building the system can be completed within the relevant schedules and constraints. While this is a generic engineering problem, how it is achieved is, of course, an open one depending on many factors not all of which have to do with engineering itself.

However, there are issues which are alluded to in the heading for this section which are of special reference to CSCW as a design activity, and it is these we now turn.

CSCW and Requirements

In this part of the Deliverable we want to begin bringing the discussion closer to some of the issues which arise in connection with CSCW and which constitute important aspects of the research in this Strand of COMIC.

The problems which system designers are required to solve are often immensely complex, and understanding them is difficult, especially when the envisaged system is new and there are no existing systems which can serve as an initial model. The latter condition is one which, for the immediate future, is likely to obtain in CSCW systems. A principle difficulty is that many of the problems that arise in establishing large scale software system requirements are usually 'wicked' ones (Rittel and Webber, 1973); that is, problems for which there is no definite formulation. Further, and as already indicated, 'requirement's capture' is really a

collection of very different kinds of problems, many of which can be affected by non-technical factors impinging on the design and development process.

We suggest that the generic problems of system design and development can be characterised as follows:

- understanding the domain of application
- relating this knowledge to designers
- managing the design and development process

Again, it is important to stress that these are not only generic they are also generally stated in that what they actually involve in the design and development process are highly varied. As we shall review in Part II, system designers have available a variety of methods and techniques for solving, or attempting to solve, the problems that these pose, methods which are, in the context of CSCW, in need of calibration.

In requirements elicitation for CSCW there are three sets of related issues here which can be stated as follows:

- the development of support for cross disciplinary working in CSCW systems development
- the augmenting of existing approaches to requirements to incorporate an understanding of cooperative work settings
- the development of tools to support the effective transfer of knowledge of the work setting to the CSCW design and development process

As we say, these are related issues and their interconnections are not always easy to see clearly, not least because many aspects of them have yet to be worked out. Nor is there widespread agreement as to what the problems are let alone agreement about what the solutions might be. Accordingly, what follows is more in the nature of a scene setting exercise in an attempt to get a better sense of what the problems might be.

It is important to note what the issues assume. It is accepted, for example, that CSCW design is multi-disciplinary in nature. It acknowledges the fact, to put it another way, that involved in design will be a number of specialised skills, experiences and expertise, including importantly, those who will use the system itself. This may involve creating new methods, it may need new methods to augment existing ones, it may mean abandoning some, but this has to be a reasoned and well grounded judgement.

Finally, what is also accepted is that system design needs some means of meeting the needs of engineering large scale systems and that this will require communicating knowledge of the work setting to developers by some effective means.

The Multi-disciplinary Nature of CSCW

In our earlier discussion of Hirschheim and Klein, we made the point that one of the implications of much of that type of critique of conventional design and

development methods is that the designer is required to be the proverbial Renaissance figure able to exhibit immense competence in all the expertise required in system design. Multi-disciplinarity is also the source of another Pandora's Box of issues to do with 'communities of practice' which make up different disciplines and the problems — often severe — of 'translation' which can be consequent on relations between such communities (Robinson, 1991). It cannot be expected that system development, if it were to seek close, systematic and principled dependence upon these sources would be able to evade these problems, problems which are by no means conclusively resolved within the disciplines in which they make their initial appearance, as we have seen in connection with sociology.

The idea of the designer as a Renaissance figure is unrealistic or, as one of our colleagues put it, 'arrant nonsense'. The fact is that CSCW system design and development, as is the case with other types of system, will involve a variety of skills and competencies ranging from those of software engineering (itself a manifold assemblage), psychology, sociology, project management, to mention but a few. Undoubtedly this poses a number of problems including those of managing heterogeneous teams, communicating between different design disciplines, organising the respective lessons, making the right choices in the light of differently auspiced analyses, among others.

Among the strategies that have been proposed for coping with multi-disciplinarity are the following contrasting views:

- the need to develop a conceptual scheme or a common theoretical framework for CSCW which can serve as a bridge between social scientists and designers (Schmidt, 1991a) so making design a more inter-disciplinary endeavour.
- the need to encourage more effective dialogue between the various contributory disciplines because no satisfactory theoretical framework is likely to be forthcoming in the near future.

There is nothing decisive to recommend either of these strategies as superior to the other except that the former is likely to prove long term if it can be done at all. For one thing, it is likely that the parties from the different disciplines will have difficulty in concurring on what constitutes the common theoretical framework. Meanwhile, it is the second strategy which, even if only by default, commends itself for the immediate future.

However, there is the prospective need, and this is germane to the first strategy, to devise means of communicating to designers especially some of the important aspects of the analysis of work domains in ways that can inform the design process. This was one of the major rationales, and the source of the kind of complaints discussed earlier, of formal methods which represent, for the designer, what needs to be done. As we shall see, there are problems with existing formal methods in characterising and representing important features of likely CSCW domains, but such problems do not remove the need to communicate these features to designers. This is an aspect of the work of Strand 3 as well as being one of the

major concerns of this Strand and the development of the Designer's Note Pad (DNP).

Understanding Work Domains

A central issue in all systems design is simply, what does the designer need to know about the domain in order to build a suitable and effective system?

As we have mentioned above, the unqualified enthusiasm for one way of tackling problems can itself create problems. Some have criticised traditional methods for allowing too rigid an attachment to an 'engineering mentality' to obscure their view of the 'human dimension' of system design. In much the same way, it could be argued that HCI has often exhibited a comparable, unrestrained enthusiasm for the automation of activities, an enthusiasm which has simply precluded the question as to whether any or further automation is, in any given case, the optimal solution.

However, and familiarly, it has become clearer that this strategy has left a great deal to be desired. Although automating many of the operating functions of the machine can improve its efficiency immensely, in human-machine systems this is by no means a guaranteed or, indeed, a desirable outcome. For example, to the extent to which CCS (Command and Control Systems) require human intervention, it is vital that the human operator remain 'in the loop' rather than reduced to a passive monitor of automated processes.[26]

Perhaps of more relevance to the bulk of potential CSCW applications are the lessons to be learned from such as the Office Automation tradition (see Part II). Whereas the intention was to develop a methodology for use on non-procedural domains (Zisman, 1977) it presumed that clerical staff did not *decide* what to do but 'reacted' to 'inputs' from the organisation. However, this is grossly oversimplified as a number of field studies have pointed out. Describing office procedures as stimulus-response patterns simply removes office work from the descriptions. As Suchman and Wynne (1984) comment:

> "The operational significance of a given procedure or policy on actual occasions is not self-evident, but is determined by workers with respect to the particulars of the situation at hand. Their determinations are made through inquiries for which both the social and material make-up of the office setting serve as central resources".

In other words, the claim is that understanding work domains cannot be adequately achieved by describing and analysing the domains using methods and models which, in effect, make the actual work activities invisible or distort them in ways such that they cease to inform system design. Much routine work is, in fact, the skilful accomplishment of human actors who, using their common sense and largely tacit knowledge of the work and its context, locally produce that routineness. What the Office Automation methods failed to capture was how the

---

26 Matters are more complex than this since, of course, CCS vary as do the risk magnitudes of failure. In air traffic control systems it is advisable that the controllers remain involved in the work in order to be more able to respond should things go wrong. (Shapiro *et al*, 1991; Bentley *et al*, 1992). In entirely automated CCS, such as in missiles, this is not an issue.

work was actually accomplished in respect of, and for example, handling 'routine' discrepancies, bending the 'fixed' rules, contextualising aspects of the work, and so on. Although office work is bureaucratic work, its routineness is an accomplishment of parties to that work (Zimmerman, 1970). The office is not a place where people perform a set of well-structured tasks according to prescribed procedures in the way that the 'automated office' model suggested that they did. Rather, the work is accomplished through locally situated activities and interactions which produce work-according-to-rules-and-procedures. Ethnographic studies of office environments have documented aspects of this rich interactional world (Suchman, 1983; Gerson and Star, 1986; Anderson *et al*, 1989), one which will not be found through the examination of the official office procedures as recorded in the manuals, no matter how detailed these may be. The way in which such procedures are operated in the organisation of work, and of the way in which they are embedded within the interactional environment of routine office, is not something which is captured or described in the earlier attempts at portraying office organisation. This means that the understanding and characterisation of these matters is, therefore, something which must figure significantly on the CSCW agenda.

Bannon (1991) characterises the human factors view which has traditionally informed much of the analysis of activities for system design as clinging to a view of the user as a "passive, fragmented, de-personalised, unmotivated individual" rather than as an active controlling and motivated person who lives and works with others. He goes on to propose that,

> "System design...should start out with the understanding that workers/users are competent practitioners, people with work tasks and relationships which need to be also taken into account in the design of systems, and with whom they must collaborate, in order to develop an appropriate computer system."

It is dissatisfactions such as these which have played a part in the rise of CSCW and the turn toward the social sciences and place the idea of cooperative work, and the support for this, on the system design agenda.

There are two principles which emerge from such dissatisfactions:

- the importance of attending to the 'real world' circumstances of system use.[27]
- the 'real world' of system use is a social world.

The first of these encapsulates the prescription that systems should not be designed in abstraction from their situations of use. The second goes further to say something about the nature of the situations of use, namely, that they are social. It is this latter which invites social science into system design and is potentially another Pandora's Box for although it can be readily acceded that the 'real world' of system use is a social world, just what this involves and how it may be

27 Although the term 'real world' is in widespread use within system design it has some unfortunate connotations. Here we use it to contrast with laboratory and experimental studies to refer to naturalistic studies of work activities.

understood as cooperative is a moot point. This is an issue we will take up later in the Deliverable in Part 3.

For now we want to consider, briefly, how and in what ways the social nature of work is cooperative.

The Nature of Cooperative Work

Bowers (1991) identifies two senses of cooperation that have appeared in the CSCW literature: cooperation as opposed to chaos or disorder, and cooperation as the opposite to competition. Others write about CSCW as if workgroups were primarily cooperative and that groupware will improve their performance (Bullen and Bennett, 1991). In the 1988 CSCW Conference, Ehn took the position that all group work required cooperation and that cooperation was a dominant element in work groups. But as Kling (1991) points out:

> "Most workplaces are much less coercive than chain gangs. But the primary alternatives are not limitless cooperation, as if these are all-or-nothing characteristics of group relations...In practice, many working relationships can be multivalent, mix elements of cooperation, conflict, conviviality, competition, collaboration, commitment, caution, control, coercion, coordination and combat..."

What this draws attention to is the option of seeing cooperation as a special quality of certain kinds of groups, such as those which involve synchronicity, small numbers, a common single task, or to see cooperation as equivalent to interdependence of activities, as argued by Bowers (1991). The former would direct the attention of CSCW toward a specific and very limited subset of human ensembles while the latter would considerably broaden its scope to include almost any work setting. It is toward the latter conception that we lean in COMIC. Work activities, though performed by individuals, typically involve levels of integration with the activities of others. This is the kind of organisation which is pointed to with the notion of a 'division of labour'.

Our argument, which we will develop later in Part III, is that such interdependencies of work activities, a division of labour if you will, is socially organised and is not to be described as some abstract, decontextualised disembodied pattern of activities but needs to be understood and analysed as the activities of persons who work within 'real world' natural settings. The effect of this shift of attention has also been to modify considerably the traditional notion of the user by acknowledging in the situation of use, the actual operator of the keyboard, for example, is not the only person to take into consideration in the design of the system. It also recognises that systems need to be designed with much more domain specificity than has, perhaps, been the case previously.

Eliciting Requirements

The injunction that CSCW attend to the 'real world' character of domains of application, is to place the nature of work first in order to better see how systems can be designed to support that work. This shift can be seen in the OIS field where there has been a change in terminology from 'automating' the office to 'supporting'

office workers with appropriately designed information systems (Woo and Lochovsky, 1986; Hewitt, 1986; Hughes and King, 1992b). The main issue for CSCW is how to design systems so that they articulate with work activities and support rather than disrupt or replace them, especially those tasks which are conditional to the work rather than its operational substance. This is also true of innovative systems for facilitating cooperation where this was not previously possible, such as in widely distributed networks. Such systems require an understanding of how they can be designed to support distributed work activities in their relevant domains. Once again, to put it simply, the emphasis is upon supporting activity rather than automating it.

This shift of focus in CSCW toward 'support' is not merely a change in terminology but a radical change of attention toward the nature of system design. One major consequence is a dissatisfaction with, and even a rejection of, the traditional methods of requirements capture methods as unresponsive to the complexity of the phenomenon of cooperative work. A point we have already discussed at some length. However, this still leaves the problem of relating studies of 'real world' domains to the needs of designers while acknowledging that computer use is a social act embedded within a socially organised world of work.

This constitutes one of the main areas of CSCW research, that is, the attempt to devise and evaluate effective ways of analysing work settings in such as way as to inform system design. One of the prime candidates for this, as already indicated, is for sociological analyses of socially organised work settings. However, and as Schmidt (1993) reminds us, there is yet much to do before sociology can deliver on this promise. One of the main prior problems here is determining what kind of sociological attention needs to be developed; for sociology is, as so many of the social sciences are, not exactly noted for its widespread agreement upon how the discipline should proceed with its inquiries.

Augmenting Traditional Requirements Methods

We have already reviewed some of the general arguments raised within CSCW, though their origins do not necessarily lie within that field, about traditional requirements methods and also tried to make clear the attitude that we intend to take to such arguments and to the methods, 'traditional' or not.

For CSCW one of the major steps is the requirement that a much larger role be accorded in system design to analyses of the 'real world' character of relevant domains. In which case, it inevitably makes system design a multi-disciplinary endeavour which dispenses with the presumption that the system designer is the authority on all matters of system design and best suited to determining what it is that users want. In other words, there is a reluctance in CSCW to leave the development of applications open to the intuition of developers. In fact, as noted by a number of people (for example, Grudin, 1989) given the poorly understood nature of cooperative work the intuition of designers about useful software for groups is likely to be poor. Uncovering the properties of useful software and its effect on the nature of the group work being supported is a central concern of

CSCW systems development. Consequently, it is important that CSCW systems development is strongly rooted in an understanding of the nature of group work and in the social setting in which it is placed.

However, this is not to argue that system development requires a whole new apparatus of methods. It does imply that many of the traditional requirements methods may need augmentation by other means.

Clearly, one of the large problems in requirements capture is a crucial element in the relation between designers and users, and that the role that users are to play could be considerably more prominent than it has been in the past.[28] One important element in CSCW is to direct more attention to end users as a vital element in design.

Tensions at Hand in CSCW System Design

We speak of 'tensions' in the design process because, as we have already noted, the decisions which have to be made within the design process involve 'opportunity costs.' In other words, there are practical dilemmas in that both pros and cons can be given for each of the alternative possibilities before the designer, and in electing for one of them means that advantages distinct to the others must be sacrificed. The choices in design are not, on a practical level, clear cut, and some of the tensions may be severe. It is not, of course, the case that the dilemma which arises in a particular case cannot, *in the long run*, be overcome, but the fact that the need to make a difficult or even painful choice may subsequently be obviated is no consolation to designers in the present where, within the scope of their resources, no reconciliation of the alternatives is practically possible.

The position taken here is not that requirements capture is dispensable but rather one of addressing the question of how do you organise requirements capture into design sequence and what kind of methods of analysis of work do you provide it with? Requirements are not given in nature — one slight problem with the term 'capture' — but must be assembled, analysed, developed from, the investigation of setting and the problems the system is intended to solve. This raises a number of issues which need to be addressed with some thoroughness and are issues which are relevant to any design approach. Among the practical questions the system design and development process poses are the following, though in no special order:

- how is the process to be resourced?
- how long is the process going to take?
- how are users to be identified?
- how is their participation in the process to be assured?
- how is the information they provide to be evaluated?
- how is it to be communicated to the design team?

---

28 Friedman (1992) identifies this as the more recent stage in software development.

- how is the system to be designed so as to be sufficiently flexible in use?
- how to balance flexibility and commitment in the design process given that, at some stage, some features of the design will have to be fixed?
- how to formulate the specification of the system in ways which support an effective balance of flexibility in the design process and yet meet the developer's need for formal and precise expression?
- how to accommodate to financial and legal constraints?
- how to achieve practical design outcomes under conditions which are less than ideal?

Each of these is, of course, a collection of problems for which, as we have been stressing throughout, there is no 'silver bullet' or panacea.

These particular problems are manifest as a set of explicit tensions which emerge as part of the design and development process. These include:

Small scale v large scale

What this tension draws attention to is the appropriateness of the methods and approach for the scale of the system being designed and developed. One might also add here, whether or not the system being devised and developed is a prototype to demonstrate a principle or a commercial product. In many respects, the role of requirements specification becomes especially pertinent in the design of large scale systems as a means of communication and planning among the various and numerous members of the design and development team.

Domain specific v generic

A traditional tension in systems development which is compounded in CSCW is that between meeting the specific needs of a situation and achieving more general and reusable software solutions. Bespoke systems while meeting the specialised needs of a single situation are costly and difficult to re-use, and subsequently to maintain. In contrast, generic applications offer lower cost solutions by reusing software. However, this is achieved at the cost of the developed system having a less precise fit with the particular needs of the work situation.

Strict decomposition v dynamic restructuring

This tension is between the desire to plan and control the work of analysis, design and development as clearly and as firmly as possible and that of supporting design in ways that fit most comfortably with designers' actual practices. It involves methods which allow for the revision of ideas in the course of design, allows for effective interaction between participants in various phases of the process, and which permits the extensive, exploratory iteration of the design.

User informed v user participation

Given the enhanced importance of the socially situated user, the tension is between involving users closely in design, effectively abolishing the designer, and informing design with appropriate knowledge of users and their activities. Neither of these extremes is likely to prove satisfactory. The issue is one of balancing the

necessary details of user activities with the practical constraints, not least that of time, of system development.

The problem at hand v maintaining the system

The problem is, at one level, to achieve a recognition of maintenance as an integral part of the product life cycle and, as such, a design problem, instead of seeing design as something which ceased at the point at which the system was handed over to users and, accordingly, maintenance as 'someone else's problem'. Incorporating techniques such as Design for Maintenance into life cycle models is one device for making maintenance part of the design process. As we shall see in Part III, in practise designers are often dominated by the 'problems to hand', that is, the immediate concerns of providing a working solution to the current problem. It is this that typically takes priority over providing solutions which will ensure ease of use.

Systems correctness v flexibility of use

System correctness is often a key feature of most developed and installed software. Normally this is ensured by fixing a high degree of determinacy which prohibits alteration of the system. Introducing flexibility into these systems also introduces a greater degree of indeterminacy and uncertainty. Highly flexible systems tend to deprive designers of ways of ensuring the correctness of the finished system. Yet flexibility of design will be an important feature of CSCW system development.

Automation v support

In many respects this is the standard tension in HCI, namely, what aspects of the human-machine system need to be automated and what left to the human users. In CSCW, the emphasis is on design for support which is likely to involve elements of automation though these need to be assessed in light of detailed knowledge of the relevant domain.

In many ways a particular tension exists in current considerations of the uncovering of requirements and the development process. In essence this tension can be viewed as balancing the need to understand the work to be supported against the pressures of managing the development of large scale systems. This tension represents the core set of issues to be addressed within this deliverable.

Structure v responsiveness

The construction of systems to support rather than automate complex patterns of work, for example, those which involve 'knowledge workers' and 'high performance work groups', has involved a significant shift away from adjusting the individual worker to the technology, to adjusting the technology to the worker. Much work, though not all, requires that the technology support be adaptive to the needs of those working with it, even to the extent that the user can shape (or 'tailor') the technology to serve purposes not always anticipated in the design. This raises the tension between the needs of the designer to achieve some 'control' over the activities of the user by designing for its use, and providing the user with

powerful yet unstructured resources giving them the freedom to devise uses for their own purposes.

As we have indicated, there are no clear cut ways of making the choices here except in the context of specific systems and the work settings in which they are to be placed. It may be, for example, that in some types of system 'tailorability' needs to be restricted, whereas in others allowing for this feature could be a positive virtue. In any event, this is likely to be a feature which differentially affects different levels of the system.

Addressing Methods

The argument that we've been developing in this part of the deliverable recognises the realities of CSCW system design, arising from the needs of engineering and managing the production of large-scale systems and also the need for a more directed social science input into this process. In other words for the forseeable future, design in CSCW will be a multidisciplinary endeavour which must acknowledge both the needs of software engineering as well as a better understanding of the work settings into which system support is to be placed. As we have indicated, CSCW design, like system design more generally, is inevitably a compromise and a matter of making as best informed choices as one can among the kind of tensions noted above. We have also indicated that, to date, some of the social science input to design has few, if any, practical lessons or consequences for the very real problems of designing large scale systems. Design and development as a collective and collaborative endeavour will need to be a managed process though the most effective way of doing this is very much an open question. The fact remains, however, that systems do get built. For CSCW design, one of the major problems has to do, as we have said repeatedly, with melding social scientific studies of the social organisation of work with the requirements of software engineering for a methodical means of effecting the technicalities of the system. Within its relatively short life, software engineering has evolved a number of methods and techniques for systematising the design process which will, in practical terms, constitute the context for CSCW design and development and it is to an evaluation of these that we now turn to in the next part of this deliverable.

Part II
Techniques to Capture System Requirements

Part II: Techniques to Capture System Requirements

Requirements and CSCW Systems

The development of real world CSCW systems requires the construction of large complex software systems which need to meet the needs of a large number of cooperative groups users and which support the working practices exhibited by the community of users. This challenge is not in itself unique to CSCW system development. As computers become more prominent in the workplace, they run the risk of becoming increasingly intrusive within work settings. To avoid the problems of impedance generated by a poor 'fit' to the nature of work, these interactive systems will need to resonate with the work activities being supported.

In many ways the problems central to the effective development of cooperative systems are a special case of the problems facing the next generation of interactive systems development. However, the central role of work within cooperative systems ensures that the problems are even more acute for the future of CSCW development. It is essential that future CSCW systems are built from a foundation of understanding the cooperative nature of the work being supported.

As we have already seen in Part I, the nature of system design and development is open to debate within CSCW. Indeed, a considerable amount of current research within CSCW can be characterised as "involving a social perspective within systems development" (Grudin 1989). Much of this debate has called into question the issues surrounding an overly abstract consideration of the development process and a simplistic consideration of the nature of work. Indeed, in a consideration of the nature of the redevelopment of early CSCW systems, Bannon (1993) states:-

> " when refinements to the initial model were presented, often these refinements were also based on abstractions rather than on any clear empirical evidence for the relevance of these features in actual work situations."

Many of these debates have highlighted the problems associated with leaving the features of CSCW applications open to the intuition of developers. In fact, as noted by a number of people (e.g. Grudin, 1989) given the poorly understood nature of cooperative work the intuition of designers about useful software for groups is likely to be poor. Consequently, it is important that CSCW systems development is strongly rooted in an understanding of the nature of group work and in the social setting in which it is placed.

Before considering the means by which developers can gain an understanding of the nature of work and how this can be used to support the construction of computer systems it is worth highlighting some central characteristics of the problem being addressed by this strand of the project. Rather than consider system development as a purely academic concern we wish to investigate the issues

surrounding the realistic development of CSCW systems within an industrial context. In this sense we wish to advance from the problematic position highlighted by Bansler and Bødker (1993) in their investigation of how structured methods are used in practise.

> "... researchers attach too much importance to tools, methods and principles, and pay too little attention to the behavioural and social aspects associated with systems development."

To suggest that all systems have a similar need for the representation and communication of requirements is naïve. Software development projects vary greatly, this may include a diversity in the development approach they adopt, the role of the computer system and the knowledge of the situation in which the system will be placed. Only in some of these development scenarios is there a need for an abstract and explicit representation of the requirements of the system. Such representation is often needed as a means of communication between and with developers. In fact a notable use of existing structured analysis techniques is as a communication tool to convey what is to be built to developers: a conclusion offered by Bansler and Bødker's (1993) study of the application of systems analysis.

Before considering any of the different approaches to systems development and construction in any detail it is worth revisiting the particular form of development of interest to the COMIC project. We would also argue that these are the also the very occasions in which some form of explicit representation of the requirements is most likely to be used. Properties of note include:

Interactive systems

The systems being developed are interactive in nature. Thus, rather than existing as embedded software, the computer systems form a significant component within an interactive work setting.

Specialised systems

The developed systems are intended to meet a particular set of functionalities for a particular community of users. That is to say systems are built to meet the needs of a client. They can be constructed either from scratch or from some core set of functionalities. These systems are distinct from "shrinkwrap" software which is intended to service a mass market.

Large scale

The size of the development task at hand is of a similar scale to those requiring the attention of software engineering approaches. That is to say, future CSCW projects are such that it is unlikely that it will be completed in a reasonable time by a handful of developers.

Managed and Visible

Development is set within an industrial setting and is open to commercial constraints and the pressures associated with market forces and budget limitations. Consequentially, it is important the development process offers facilities to allow it to be both manageable and visible.

This section considers the approaches to requirements which have emerged from a variety of different traditions. Rather than attempting to be exhaustive in each of these traditions the particular methods highlighted are illustrative of the different approaches. A more complete investigation of the different methods and approaches are given in the set of supporting COMIC documents.

Systems and the workplace

The distinguishing feature of CSCW systems is their potential impact on the work place; an impact which is often difficult to determine partly because of a lack of understanding, partly because system effectiveness is open to influences beyond those of software engineering and partly because many CSCW systems will offer new means of cooperative work.

As we have seen in Part I, many of the traditional approaches to requirements and system development have come in for no little criticism not least from software engineering itself. An important element in these criticisms has been the need to understand the work setting in which systems will have to play their role; a need which, it is argued, has often been ill-met by traditional methods. As a consequence there has been much effort within software engineering to develop alternative and more effective methods for the design and development process; so much so that it has become difficult to assess how and in what ways each method can usefully play a part in system design and development.

This Part of the deliverable examines the various candidate techniques which can contribute to understanding cooperation in the workplace and conveying that understanding to system developers. We examine a number of methods and techniques which have emerged from a variety of different backgrounds. The intent of this examination is to highlight the particular emphasis of each approach and the particular view of cooperation embedded within the approach.

Alternative Development Philosophies

A significant problem in assessing different approaches to requirements development is that often we are attempting to compare techniques which reflect widely different perspectives on what constitutes software development. Many of the existing methods associated with software development take a general and holistic approach to the development process. Consequently, it is worth reviewing the different methodological stances which have been taken to the development of software systems. Each of these characterise the requirements problem in different ways and put a different emphasis on both the nature of requirements and their place in the development process.

This section briefly reviews a number of distinct approaches to software development and the particular positions they embody. In doing so we highlight each perspective's consideration of requirements and the corresponding influence on the methods which have been developed. We do not consider this list of development perspectives to be exhaustive but consider those which characterise

the different approaches prevalent in debates of the development of CSCW systems.

The software engineering perspective

Software engineering models of development have traditionally suggested a linear sequence of activities to be carried out step-by-step, as we saw in Part I in the discussion of the Waterfall Model. The idea of the 'life-cycle' is intended to capture the overall process of system development, from the inception of the idea through to the eventual elimination of the system from service. In significant respects, and as its origins suggest, software system development is treated as an instance of project organisation and management. Crucial to the model is the importance attached to the early phases of determining system requirements for the design and holding off implementation of the design in software until late in the whole process. There have been a number of variants proposed since the original version appeared in the early 1970s, though all follow much the same logic.

Variants often emphasise different phases. Thus, for example, the development of Structured Methodologies is often treated as an advance on classical methods by providing a more elaborate articulation of the middle stages of the life cycle and a more systematic working out of the design and its implementation. Other approaches place their stress on 'front end' loading and postponing detailed design until the requirements and specifications are more assured.

As indicated, the requirements phase is a critical first step.

> "The hardest single part of building a software system is deciding what to build...No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later." (Brooks, 1987)

In the case of traditional software engineering approaches the difficulties surrounding the development of requirements are particularly problematic for a number of reasons. From the software engineering perspective the central difficulties in constructing requirements are enumerated by Sommerville (1992) as:

> "1. Large software systems are usually required to improve upon the *status quo* where either no system or an inadequate system is in place. Although difficulties with the current system may be known, it is hard to anticipate what effects the 'improved' system is likely to have on an organisation.
>
> 2. Large systems usually have a diverse user community who have different and sometimes conflicting, requirements and priorities. The final system requirements are inevitably a compromise.
>
> 3. The procurers of a system (those who pay for it) and the users of a system are rarely the same people. System procurers impose requirements because of organisational and budgetary constraints. These are likely to conflict with actual user requirements."

An important distinction in the software engineering perspective on requirements is the difference between *system goals* and *system requirements*. The distinction is that a requirement is something that can be tested whereas a goal is a more general characteristic which the system should exhibit. For example, a goal might be that

the system should be ‘user friendly’. This is not testable since ‘friendliness’ is a subjective attribute. An associated requirement might be that all user command selections should take place using command menus.

This is an example of the different interpretations of what ‘requirements definition and specification’ actually means. Depending on particular organisations, the system requirements specification can be anything from a broad outline statement in natural language of what services the system should provide to a mathematically formal system specification. The line between requirements and design specification is a tenuous one and sometimes, the terms are used interchangeably.

Much of the focus of work in the software engineering tradition of requirements is on the need to document a system so that it can be understood by potential users and, at the same time, produce a system specification (often called the functional specification) which can act as a basis for a contract between a procurer and a software supplier.

This is an obvious point of tension in the process. Generally, users prefer a higher level of abstract description than that which can be provided in a specification which is sufficiently detailed to act as a contract. Furthermore, it is probably impossible to construct a detailed specification without some design activity so further blurring the distinction between requirements and design specification. To alleviate this a number of distinct specifications can be highlighted, according to Sommerville (1992):

- “A *requirements definition* is a statement, in a natural language, of what user services the system is expected to provide. This should be written so that it is understandable by client and contractor management and by potential system procurers and users.
- A *requirements specification* is a structured document which sets out the system services in more detail. This document (sometimes called a functional specification), should be precise so that it may act as a contract between the system procurer and software developer. It should be written so that it is understandable to technical staff from both procurers and developers. Formal specification techniques may be appropriate for expressing such a specification but this will depend on the background of the system procurer.
- A *software specification* (design specification) is an abstract description of the software which is a basis for its design and implementation. There should be a clear relationship between this document and the requirements specification but the important distinction is that the readers of this are principally software designers rather than users or management.”

These different forms of specification correlate with particular activities within a software process model. The software engineering community recognise that formulating outline requirements for a project is difficult as the application domain is poorly understood. Rather than expect a definitive requirements definition before system development, a process model based on prototyping is considered more appropriate. In the cases where a process model which does not involve system

prototyping is used, a number of steps in the derivation of requirements are outlined, again according to Sommerville (1992):

1. "*Feasibility study*. An estimate is made of whether the identified user needs are satisfiable using current software and hardware technologies, whether the proposed system will be cost-effective from a business point of view and whether it can be developed given existing budgetary constraints.
2. *Requirements capture and analysis*. This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, etc.
3. *Requirements definition*. A system model is formulated and this is used as the basis for an abstract description of the system requirements. This is primarily a document which describes the system from the user's point of view.
4. *Requirements specification*. A detailed and precise description of the system requirements is set out to act as a basis for a contract between client and software developer. The creation of this document might be carried out in parallel with some high-level design (indeed, this is often essential) and the design and requirements activities influence each other as they develop."

Of course, the activities in the requirements process are not simply carried out in sequence but are iterated. The requirements analysis continues during definition and specification and new requirements arise during the process. In the previous section of this deliverable we examined the various criticisms which have been levelled against this "traditional" approach to software development. Irrespective of this criticism it is worth noting that the associated methods based on the work of Yourdon, and DeMarco are by far the most widely used methodologies for analysing and designing computer based systems.

The methodology is based on a systematic abstraction of the existing implementation which exploits a number of models of the system being constructed. First a "*current physical model*" of the existing field of work is developed, that is, a model of the system as it is implemented containing persons, departments, routines and procedures, machinery, formulas, files etc. (cf. Yourdon, 1982). Based on the physical model a "*current logical model*" is derived, that is, a model of the system concepts and their relational dependencies "expressed in abstract, logical terms without any reference to the physical means by which the user actually accomplishes his present business policy" (Yourdon, 1982).

Much of the effort used in the analysis phase is used to identify the concepts in the 'real world' which must be included in the computer system. The first models can easily be verified by the users. Based on these early models a model called "the new logical model" is constructed illustrating the functionality of the system and the concepts it will be based on. This process has, in principle, no relations to the physical model, and it becomes the basis of constructing a "new physical model" which is used to guide the construction of the final computer system. In line with the traditional Waterfall Model of development which motivated much of this work, the relationship between these models is predominantly sequential and transformational (figure 4)

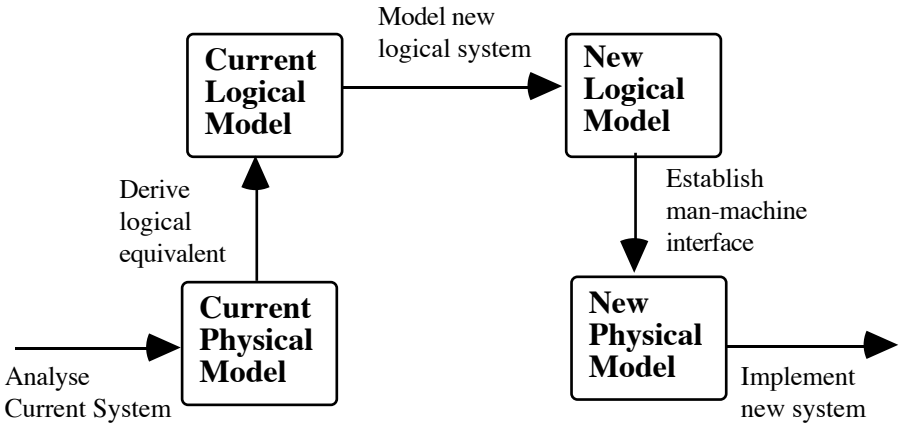

Figure 4 — The relationship between the four types of models in structured analysis

The methodology is based on the assumption that a system can be described as a process receiving and producing data flow. As a result the modelling approach is oriented towards software development rather than understanding the nature of work. The different distinct development methods which are based on structured analysis are reviewed later. The techniques which are derived from a software development perspective are also considered within the development oriented section.

The HCI tradition

Human–Computer Interaction and User Centred Design (Norman and Draper 1986) have become substantial areas of research in recent years. In many ways their central focus has been upon establishing who end-users are and what their tasks and goals are as a basis for understanding the requirements for computer support tools.

The origins and motivations behind these lines of thinking have been charted in different ways by numerous authors. A definitive history is not attempted here but it is interesting to note certain aspects of the approach. Bannon (1991), for example, has highlighted the range of conceptual treatments given to people as users, as operators, as human factors and as human actors. He comments on how Ergonomics, Human Factors, Human–Computer Interaction and Participative Design have ranged in the implicit views of people they adopt. These approaches vary from treating people as passive, fragmented, depersonalised, unmotivated, and as another system characteristic (with faulty memory and limited attention), to accounts that see people as active, controlling, involved with more than using a computer and involved in situations with multiple tasks and users. The elements of HCI to be considered here lie somewhere in the middle between, on the one hand, a concern for the surface characteristics of displays (font sizes, colour options, etc.) and, on the other, a concern for the full social complexity of work.

Bannon (1991) has also noted how the increase in computer use and the production of novel applications, motivated attempts to understand how to improve the 'learnability', 'usability' and 'understandability' of systems. Both the machine operator and the dedicated programmer became replaced by a focus on casual users.

Others have considered the basis of some of the conceptual notions deployed in HCI and how it positions itself with relation to other disciplines. For example, Cooper and Bowers (forthcoming) have attempted to examine the 'internal logic' of HCI as a new discipline and consider its dependency on the notion of 'users'. Woolgar (1991) has examined the process of identifying putative users and their requirements suggesting that this is a matter of debate in which users and their desires are constructed or 'configured' alongside the building of a machine to meet them. The concepts of 'users' and 'designers' are also to be distinguished from each other according to HCI. Each has different views upon the same system. Designers are said to be concerned with data structures and processes acting upon them, while users attempt to understand the system in order to use it in performing tasks. HCI almost stands between the user and the designer. It represents the needs of users and their desire for 'usable' tools while attempting to convey user requirements in ways that some suggest are the only ways that designers will understand them. At a theoretical level HCI has also on occasions seen itself as between Cognitive Psychology and Computer Science, bridging the gap by informing design.

Aspects of this separation of concerns can be seen in the Seeheim model below (Pfaff, 1985). This model arose from a workshop concerned with computer architectures. Considering this model as a programmatic statement (rather than considering its impact as a practical architecture) it can be seen to permit the creation of HCI from a computer science perspective in that it provides for the separation of 'interface' from 'application' and the communication between them.

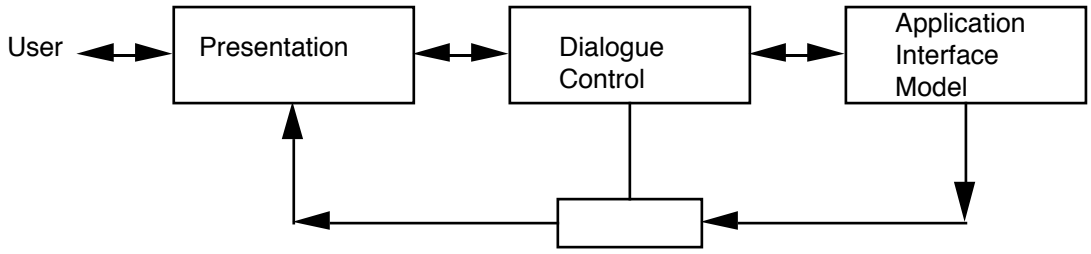

Figure 5 — The Seeheim Model

Despite these aims the actual nature of systems design and development has only relatively recently started to emerge as a significant consideration within HCI. The focus within software engineering has traditionally been on the problems of constructing large software systems and providing techniques which support the effective management and control of the development process. In contrast, the focus within HCI has been on the smaller scale with a particular emphasis on the

detail involved in the development of systems which closely match the tasks of users.

A similar distinction between user tasks and artefact is envisaged by Norman (1989) in his consideration of systems design within HCI. He stresses the central role that the *conceptual model* a user has of a system (computer or otherwise) has to play in determining the effectiveness with which that system can be used. He argues that a correct conceptual model can transform difficult, confusing tasks into simple, straightforward ones. Norman highlights the differences between a user's goals, expressed in psychological terms and a system's mechanisms and state, expressed in physical terms. Norman's consideration of design focuses particularly on the role of the user interface and he argues that for a computer system the mechanisms can be represented by the input devices provided, while the state is determined by the configuration of the output devices, including the display screen.

Norman presents these differences as two uni-directional *gulfs* between user goals and the physical system (figure 6). Transformation of user goals into a sequence of actions with the input devices is necessary to 'bridge' what Norman terms the *Gulf of Execution*. Similarly the *Gulf of Evaluation* is bridged by the interpretation of system state from the output devices and comparison of this with intended goals. Appropriate representation of application entities can help in bridging both these gulfs.

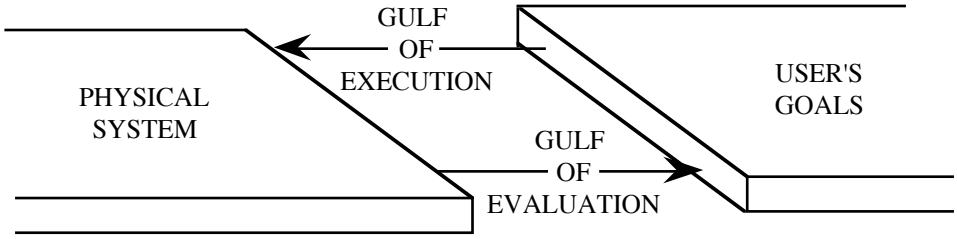

Figure 6 — The Gulfs of Execution and Evaluation[29]

Norman argues that the usability of any system, computer or otherwise, is directly related to the size of the Gulfs of Execution and Evaluation. The task of design is thus to reduce the size of these Gulfs and, hence, provide more usable computer systems. Much of the emphasis within HCI in bridging this gap is in the development of appropriate means of task analysis and techniques to represent these tasks to systems developers. Some of these methods are reviewed in our later consideration of task analysis. The general approach adopted is to develop a clear model of user and tasks which can then be used to convey the requirements of users.

We have seen that as a means to achieve its purposes HCI has looked to the techniques, theories and perspectives of Cognitive Science. In many respects, the field of HCI as a design discipline is constituted by this use of cognitive theory and

29 From Norman (1986): figure 3.1

method. Hence efforts have been oriented towards understanding the development of users' mental models of the system by drawing on psychological theories of skill acquisition, expert/novice performance and knowledge representation. Users are said to progress from simple command-function mappings to a deeper understanding of the system's conceptual structure allowing them to predict, reason and plan the operation of the system. In this way HCI has offered to attempt to improve the design of interfaces such that they convey an image of the underlying conceptual model of the system quicker and more accurately to the user.

In considering the user centred development of computer systems and the implications for design, Carroll *et al* (1990) stress the importance of iteration and evolution. They argue that the introduction of a computer system into a domain to support a particular activity very often changes the nature of that activity. To understand and support this Carroll *et al* (1990) have formulated a model of innovation based around a *task-artefact cycle* which assumes that innovation of any kind continually involves building artefacts to support tasks then modifying these artefacts as the tasks are changed by their introduction.

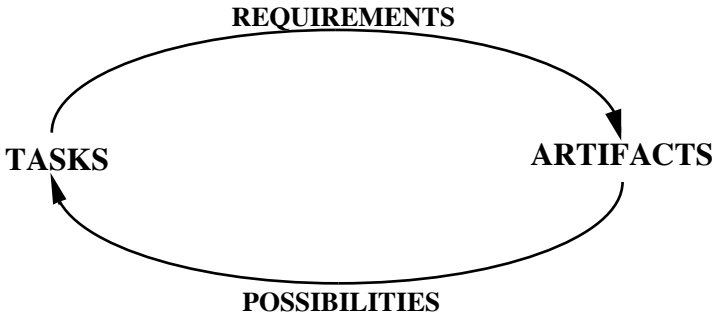

Figure 7 — The task artefact cycle

In proposing an evolutionary process as the basis for design the Task-Artefact Cycle argues that the evolution of HCI technology is a co-evolution of HCI tasks and HCI artefacts. In discussing the nature of the cycle Carroll *et al* (1990) make the point:-

> " a task implicitly sets requirements for the development of artefacts to support it, an artefact suggests possibilities and introduces constraints that often radically redefine the task for which the artefact was originally developed".

In proposing the task artefact cycle Carroll *et al* (1990) make an explicit separation between user tasks and the behaviour of the artefact. Carrol also suggests that the psychology of HCI tasks will form the basis of understanding the tasks within the cycle. In doing so he stresses the importance adopting a richer consideration of tasks which allow some contextual representation. A number of candidates are proposed including Activity Theory (Bannon and Bødker, 1990). A central feature of the cycle is the analysis and representation of tasks to inform the development of the artefact. He stresses the importance of "information flow" between the two and looks to close coordination between them. The means of

effecting this coordination is the use of some form of design rationale (MacLean*et al*, 1989) to support the communication between the two domains.

An alternative approach of iterative development which sits better with the task artefact cycle outlined by Carroll does not attempt to formally model the data, tasks or users. Rather, evaluation of prototype designs by end-users engaged in real tasks allows iteration towards task- and user-specific systems. Such an approach, as illustrated in figure 8, has long been advocated within software engineering for developing systems where the understanding of the problem is poor.

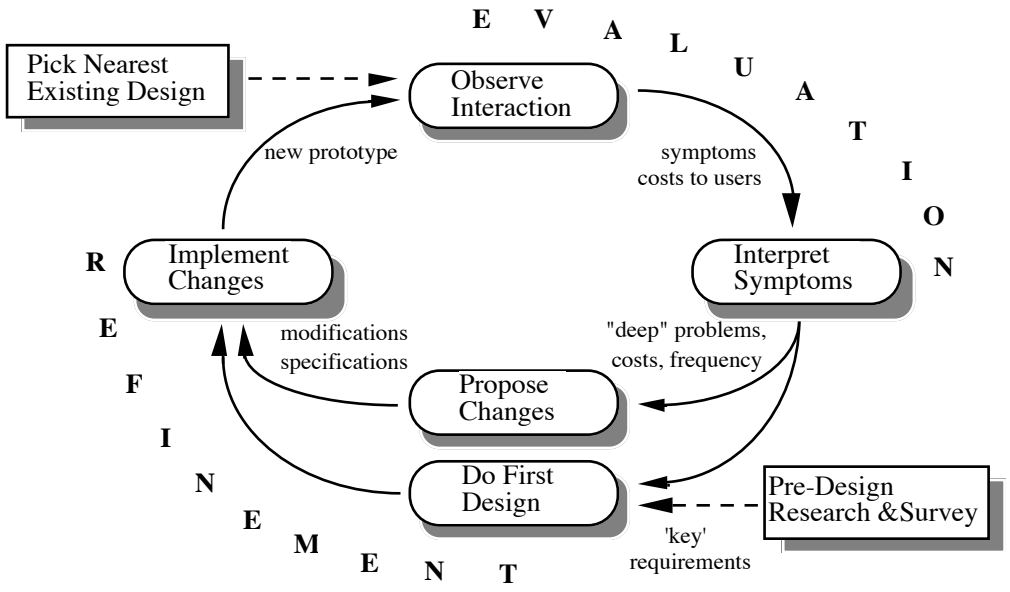

Figure 8 — A HCI prototyping cycle[30]

In addition to utilising qualitative rather than quantitative data, an approach based on rapid prototyping has the advantage that a range of different designs can be developed and evaluated. Some of these designs may initially seem unsuitable given current understanding of what makes a user interface effective, but evaluation trials may demonstrate their suitability for end-users who have extensive knowledge of their domains of expertise.

Participative Design

Participative or *participatory design* (PD) is an approach to information systems design which has centred on the Scandinavian countries and which originated in the early 1970s. Over this period it has naturally developed in various directions so that it is not a single unified approach with unambiguous adherents. Its most distinctive points of departure are:

(1) it is politically committed to workers and direct producers, and to the enhancement rather than the destruction of their skills, autonomy and quality of working life. It has sought to realise this through working

30 Based on Draper and Oatley (1991).

closely with trade unions at various levels, and enhancing their capacity to devise and to achieve alternative technology policies.

(2) it is action research, intended to make a direct and immediate difference to the working conditions of their research 'subjects' and to foster industrial democracy.

(3) it proceeds by a direct and unmediated partnership between designers and the users of systems, and has devised a range of methods for rapid prototyping of systems in collaboration with users.

Exponents of PD have identified at least three generations of their work.[31] The pioneering project was that of the NJMF (Norwegian Iron and Metal Workers' Union) initiated in 1970 (Nygaard and Bergo, 1975). Ehn (1993) describes how this evolved from a conventional research project to one in which local unions chose the topics for study and action and were supported by the researchers. A planning and control system, an on-line production information system, and the reorganisation of a main assembly line were investigated in this way. A key outcome was a local and subsequently (in 1975) a national 'data agreement' regulating the design and introduction of computer-based systems.

Several further projects were inspired by this. DEMOS (Democratic Control and Planning in Working Life) was supported by the Swedish Trade Union Confederation from 1974 to 1979 (Carlsson *et al,* 1978). It explored the possibilities for the unions to influence the design and use of computer-based systems at a daily newspaper, a locomotive repair shop, a metal factory, and a department store. In the case of the locomotive shop, it largely succeeded in displacing a projected Taylorist system for planning and supervising repairs, with a system based on the autonomy of repair teams and flexibility in planning and executing the work (Ehn, 1993a). A comparable project DUE on democracy, education and computer-based systems was carried out in Denmark (Kyng and Matthiassen, 1982). In this phase of the work information packages about workplace technologies were developed, making courses and training materials available to workers and their representatives.

The second generation can be represented by the UTOPIA project (Training, Technology and Products from a Quality of Work Perspective) started in 1981 (Bødker *et al,* 1987; Bødker, 1990; Ehn and King, 1984). It focused on graphic workers doing newspaper page makeup and image processing. It involved the 'tool perspective', that computer-based tools should be designed as an extension of the traditional practical understanding of tools and materials within a craft or profession (Ehn, 1993a). This necessitated a mutual learning process on the part of designers and graphics workers. A breakthrough in the course of this was the development of a design-by-doing approach using mockups, wall charts and organisation games or

31 Expostitions and analyses of PD at various stages of its development and from various perspectives include Bjerknes, Ehn & Kyng (eds) (1987), Ehn (1988), Bansler (1989), Floyd *et al* (1989), Greenbaum & Kyng (1991), Whitaker *et al* (1991), Bødker (1990), Clement & Van den Besselaar (1993), Shuler & Namioka (1993), and the proceedings of the 'IRIS' conferences on Information Systems Research in Scandinavia (annually since 1977).

future workshops for prototyping. Many of these techniques have now become widely used and accepted in the systems industry. While a promising and skill-enhancing graphic system was devised, lack of resources and some conflicts of professional interests prevented its full development (Ehn, 1993a). Now under the label of the Collective Resource Approach (CRA) (Ehn and Kyng, 1987), the key PD concepts include:

- The desire to enhance worker control over technology
- The centrality of the labour process
- The need to build on the tacit knowledge and skills of labour
- The use of computers to augment worker skills, not replace them
- The utility of the tool rather than the machine metaphor for the computer
- The poverty of formal models and abstractions of the work process
- The need and desire for 'user' involvement in the design process
- The need to develop concrete representations of designs — prototypes
- The awareness of systems design as a political and social, as well as technical process

Recently, Kyng (1993) has cited the Århus AT project with the National Labour (safety) Inspection Service as an example of a 'third generation' of PD. This involves cooperation with both workers and managers in the design of applications to support organisational development, with local cooperation in design supplemented by negotiations. What makes it a 'third generation' project, according to Kyng, is that it is the first occasion on which CRA techniques have been used over the whole life cycle of a design, and the first project which has involved all of the interest groups in an organisation.

Some commentators suggest that presentations of PD have been over-optimistic and that it has not, even on its home territory, achieved the real impact on working practice which is sometimes claimed for it (Kraft and Bansler, 1992). Defenders of PD counter that it has not been naïve about the prospects for realising its objectives and has reported problems clearly (Kyng, 1993). As indices of its success, Kyng points not only to the specific research cases but to such things as the very large number of worker and union participants who have been through PD training courses and seminars, and the institutionalisation of PD in many information systems curricula. Undoubtedly, the vast majority of systems development work in commercial settings continues to be done using structured design methodologies. But it is arguable that even here, the failure of systems analysts and designers to pay serious attention to users' needs is becoming an increasingly embarrassing gap (though at present they are as likely to turn to other HCI perspectives as to PD for assistance).

PD has recently enjoyed a surge of interest in information systems research in North America and in the rest of Europe beyond Scandinavia. A recent issue of the Communications of the ACM was devoted to this topic (Kuhn and Muller, 1993), and a series of participatory design conferences was begun in 1990 (Schuler and Namioko, 1993). There is certainly no doubting the strong influence which this

school has had on a number of leading research groups. It is perhaps ironic that its wider success in terms of academic research is occurring at a time when political changes have further undermined the prospects for realising its original trade-union oriented and industrial democracy strategies. Having its origins in a particular political conjuncture where the possibilities for the development of social democracy seemed most open (Scandinavia in the late 1960s and early 1970s), it is now reorienting to the fiscal and ideological retreats of socialism, and the reinsertion of the Scandinavian countries into the mainstream of competitive capitalism (Ehn, 1993b).

In the light of these changes, some commentators suggest that the main rationale for PD has shifted from democratic participation to effectiveness in design (Procter and Williams, 1992), as evidenced, for example, in Greenbaum and Kyng (1991). It might be more accurate to say that the concern with effectiveness has always been there, but that its grounds have shifted. In the earlier formulations, the capitalist context and its labour process implications were themselves seen as the main fetters on development, so that a thoroughgoing industrial democracy was the most obvious way to unleash more effective systems. As confidence in this straightforward solution has weakened, more complex partnerships are being sought as a means both to the quality of working life and to effective systems.

This could be judged, on the one hand, to show continuing developments in subtlety and openness of approach; or, on the other hand, to reveal some indecisiveness and loss of direction. Whitaker *et al* (1991) critique both the melange of theoretical inspirations espoused by some in the PD school (ranging from Heidegger to Marx to Wittgenstein) and the lack of a clear research and evaluation methodology. It is arguable that the mainstream in PD agrees that a broader interdisciplinary cooperation is necessary, but is not quite sure which or how; and equally that a broader collaboration with different parties and interests in production is necessary, but is not quite sure which or how (Knudsen *et al*, 1993). Of course, it is not alone with this problem.

Participative Design And CSCW

The PD approach has various affinities with the incorporation of a sociological perspective in design which is being attempted within CSCW. Partly for political and partly for technical reasons, PD has followed the same course and employed much of the same literature as the sociology of work, representing in both cases a shift from a sociotechnical to a critical basis. Both received a great impetus from the intense debates on the labour process which followed the publication of Braverman's *Labor and Monopoly Capital* in 1974. In a later phase, which he describes as emerging from his experiences in the UTOPIA project, Ehn (1993a) has espoused a Wittgensteinian language-game perspective in which entering into users' intersubjective practices is a prerequisite for design. This obviously brings those who follow this development into close alignment with ethnographic studies of work undertaken from an ethnomethodological perspective. As in sociology, we may expect some tensions in trying to reconcile these different aspects of the field.

In practical terms too, paying close attention to users' practices and their own articulation of their needs has required participative designers to encounter the social organisation and interdependency of that work, and so to try to support its cooperative aspects.

So for a range of reasons, PD and CSCW have arrived at a position with several crucial constituents in common. In terms of personnel too, PD practitioners were among the first to take up CSCW in Europe, and have exerted a corresponding influence on its development. While there are many different possibilities, a continuing convergence is quite likely. A key remaining difference is whether the relationship between designers and users is to be an unmediated one, with any necessary interdisciplinary connections being mainly realised in the person of designers themselves; or whether multidisciplinary teams are to collaborate in relating to users and analysing their needs for support. But here too a probable line of development is a convergence of these positions. Participative designers have sometimes worked in interdisciplinary teams, and seem increasingly ready to recognise, for example, the contribution of ethnographic studies; while within CSCW, the view may be gaining ground that there is no substitute for direct contact between users and all of the partners to design, through all the stages of development.

A related key issue which both communities need to address, and which they might resolve differently, is their approach to the 'industrial' character of systems design. Are the developments of recent decades in software engineering — involving a refined division of labour and management structure, and the formal specification of requirements — simply necessary in a world of large-scale systems produced to contract? Or are they seriously incompatible with a processual view of the development of systems to support socially-organised work activities in context (Grønbæk *et al,* 1993; Bickerton and Siddiqi, 1993; Goguen and Linde, 1993)? As indicated in earlier sections, we can expect that different kinds of compromise will prove suitable for different kinds of systems and parts of systems. Some of these issues are taken up again later in this report.

The Information Systems Tradition

As in the case of structured analysis the basis of the Information Systems approach is in the construction of models. However, these models differ considerably from those in structured analysis. The models used in structured analysis treat people as objects and take no account of the setting of a system in terms of resources, training or management. In contrast, the Information Systems tradition focuses on extending these models to incorporate the organisational setting of systems. While the content and limits of these models are of major interest to the work of organisational context within the COMIC project they are included here because of their use to inform developers of the organisational setting of computer systems.

'Information systems' as an academic discipline and a practical profession is based on modern computer technology. As computers invaded business

applications such as management and in particular unstructured decision-making (Keen and Scott Morton, 1978), it was soon understood that the major problem of IS development is the specification of what the system should do rather than the technical implementation. Young and Kent (1958), and Grindley (1966) were among the first to recognise the importance of the specification problem. The Scandinavian infological tradition pioneered by Langefors (1966) proposed a distinction between the infological problem of defining the information to be provided by the system in order to satisfy the needs of its users, and the datalogical problem of designing the structure and operation of the system and exploiting current information technology (Langefors, 1974).

It was gradually observed, however, that extension of the problem area of IS design to cover the infological problem is not enough, largely because of the problem of implementing the organizational changes implied by an information system. It was understood that without any changes in the adopting organization, its structure, activities, work, etc., one cannot expect any benefits from the system. Reflecting the above trends, there is increasing agreement about the fruitfulness of distinguishing three levels of abstraction or modelling for an information system (Langefors, 1974; Kerola and Järvinen, 1975; Welke, 1977; Falkenberg *et al*, 1983; Iivari, 1983; Essink, 1986; Lyytinen, 1987):

1. The organisational level, which defines the organisational role and context of information systems.
2. The conceptual/infological level, which defines the "implementation-independent" specification for an information system.
3. The datalogical/technical level, which defines the technical implementation of an information system.

The latter two levels are usually called 'analysis' and 'design' in the IS literature. Consequently, the distinction between the conceptual/infological and datalogical/technical level can be exemplified by the distinction between the conceptual schema (described using an Entity-Relationship (ER) model, for example) and the internal schema in the DB tradition. This review focuses on three approaches within the Information systems tradition which focus on the organisation level of information systems and a consideration of what is modelled within these approaches.

User Oriented Techniques

A key feature of CSCW systems is their interactive nature. CSCW systems exist within and support communities of users working together on common endeavours. Consequently, an important feature of developing CSCW systems is gaining insight into the needs of users. This section reviews a collection of methods and techniques which have emerged to develop an understanding of the relationship between IT systems and the needs of users. These techniques represent a wide variety of backgrounds and traditions. However, each supports a different contributing need in building a set of requirements for CSCW systems.

The Needs of Organisational Work

The relationship between cooperative work, computer systems and organisational context is central to CSCW systems. This relationship provides the theme for Strand 1 of the COMIC project. The relationship between the organisation being supported and the cooperative systems placed within it is manifest in a number of ways:-

- The cooperative system is based on some theoretical perspective of what constitutes an organisation and organisational work.
- The cooperative system makes explicit use of some computational model of the organisation within an organisational knowledge base or memory.
- Changes in the organisation or the work which occurs within the organisation.

The nature of organisational work and the needs reflected by organisations play an important part in determining the requirements of cooperative systems. Here we wish to reflect on information systems development techniques which have focused on reflecting the organisational needs as part of the development process.

In particular, we wish to focus on two different techniques for understanding organisations which have provided the basis for a number of methods, Soft Systems Methodology (SSM) and sociotechnical systems development. These are important in terms of reflecting the requirements of users which emerge from the nature of the organisation within which it exists.

SSM Based Methods

Soft Systems Methodology (SSM) is a general systems approach developed by Professor Peter Checkland and his colleagues at the University of Lancaster since the early 1970's. It has been published in a number of books (Checkland, 1981, Wilson, 1984, Checkland and Scholes, 1990). Although SSM was originally a general systems approach, without any specific orientation towards information systems, its developers are increasingly perceiving it to be well suited to IS development (e.g. Checkland and Scholes, 1990). The following synopsis, by von Bulow (1990) is mainly based on the most recent material (i.e. Checkland and Scholes, 1990):

> "SSM is a methodology that aims to bring about the improvement in areas of social concern by activitating in the people involved in the situation a learning cycle which is ideally never-ending. The learning takes place through the iterative process of using systems concepts to reflect upon and debate perceptions of the real world, taking action in the real world, and again reflecting on the happenings using systems concepts. The reflection and debate is structured by a number of systemic models. These are conceived as holistic ideal types of certain aspects of the problem situation rather than accounts of it. It is taken as given that no objective and complete account of a problem situation can be perceived."

The original cycle of the inquiry process of SSM consisted of seven stages (Checkland, 1981; Checkland and Scholes, 1990):

1. The problem situation considered problematic.
2. Problem situation expressed.

3. Root definitions of relevant systems.
4. Conceptual models of the relevant systems.
5. Comparison of the models and the real world.
6. Systematically desirable and culturally feasible changes.
7. Action to improve the problem situation.

Stages 1-2 and 5-7 are 'real-world' activities whereas stages 3 and 4 are 'systems thinking' activities. More recently, Checkland and Scholes (1990) have described SSM in the IS development as an eight step process :

1. Worldviews, Weltaunschauungen, in the situation
2. Meanings attributed to the perceived world
3. Real-world action
4. Models of purposeful activity systems
5. Information-flow models
6. Information categories embodied in information flows
7. Data structures which express the information categories
8. Design of appropriate data manipulation system (information system).

The authors do not explain the process in detail, but in terms of Checkland's earlier discussion (Checkland, 1981) steps 1-3 can be interpreted to concern modelling the real world and step 3 systems thinking about the real world in terms of 'human activity systems'. Step 4 includes the evaluation of the relevance of the human activity systems from the viewpoint of the real world action. Once a truly relevant human activity system has been agreed upon, one can proceed to step 5, asking in the case of each activity, "What information would have to be available to enable someone to do this activity?"

Soft systems methodology was not developed to support the construction of information systems. However, a number of different approaches to systems development have emerged which use variants of the Soft Systems Methodology to investigate and represent the real world. This model can then be used to inform the development of new computer systems. The most notable of these approaches have been MULTIVIEW and FAOR (Functional Analysis of Office Requirements) MULTIVIEW is, in fact, a framework to join together a number of different development and analysis methods.

MULTIVIEW

MULTIVIEW (Wood-Harper *et al,* 1985; Avison and Wood-Harper, 1990) is a synthesis of features of several methods. It distinguishes five stages or views in information system analysis and design

1. Analysis of human activity
2. Analysis of information
3. Analysis and design of sociotechnical aspects
4. Design of human-computer interface
5. Design of technical aspects

Analysis of human activity is based on SSM (Checkland, 1981), and examines the activities of the organisation concerned for its major objectives. Analysis of information takes place in the sense of functional modelling (functional decomposition and events triggering the functions) and entity modelling (entities and entity lifecycle). It results in the creation of a model of the information processing required to satisfy the objectives identified as a result of the previous stage. The stage has clear similarities with Information Engineering in particular. Stage three is based on the ETHICS method (Mumford, 1983), and is concerned with how the information system to be developed is to be fitted into the organisation. Design of the human-computer interface is concerned with the technical requirements of the user interface to the information system. Specific decisions are made at this stage on the various technical system alternatives available. Finally, the technical aspects of the system are turned to in stage five, which aims to create a technical solution to the various human and system requirements arrived at as a result of stages one to four.

Multiview is intended to be flexible in use, allowing the various stages to be turned to as and when necessary during the analysis of the information system. The various methods incorporated within Multiview are integrated in such a way as to ensure that the different perspectives of the people involved in the development and ultimate use of the system can be considered. Therefore, the views of end-users are concentrated on during stages one and three; the technical concerns of system developers are the focus of stages two and five; and the interface between user and technical system are turned to in stage four. Whilst stages one and two should be followed in sequence, stages three to five can be followed in whichever order is appropriate to the situation, and can be returned to as and when necessary during the development process.

Functional Analysis of Office Requirements

The FAOR approach is a specific application of SSM to the analysis of office information systems. It includes an Activity Framework, Generic Office Frame of Reference (GOFOR), Benefit Analysis Framework (BAF), and a number of instruments (Function Analysis Instrument, FAI, Communication Analysis Instrument, CAI, Information Analysis Instrument, IAI, User Needs Analysis Instrument, NAI, and Benefit Analysis Instrument, BAI). The Activity Framework is depicted below in Figure 9 (Schafer *et al,* 1988)

The general activity framework is based on SSM. In accordance to SSM, problem solving in FAOR is seen as a learning process. Each of the four major activities of the framework is further divided into four subactivities which correspond to different stages in Checkland's Model (figure 10).

Soft Systems Methodology offers different support for the analyst during different activities of the FAOR framework. As set out in figure 9, in the exploration activity (A1), SSM assists an analyst in getting to know the client organisation and in developing a rough model of the future office system. In the method tailoring (A2), SSM is primarily a thinking tool for constructing a method

out of the existing instruments. In the analysis (A3), SSM helps to relate the results to a coherent set of requirements, and in the evaluation (A4), SSM is used to control the results towards the fulfilment of the study objectives (Schafer *et al,* 1988).

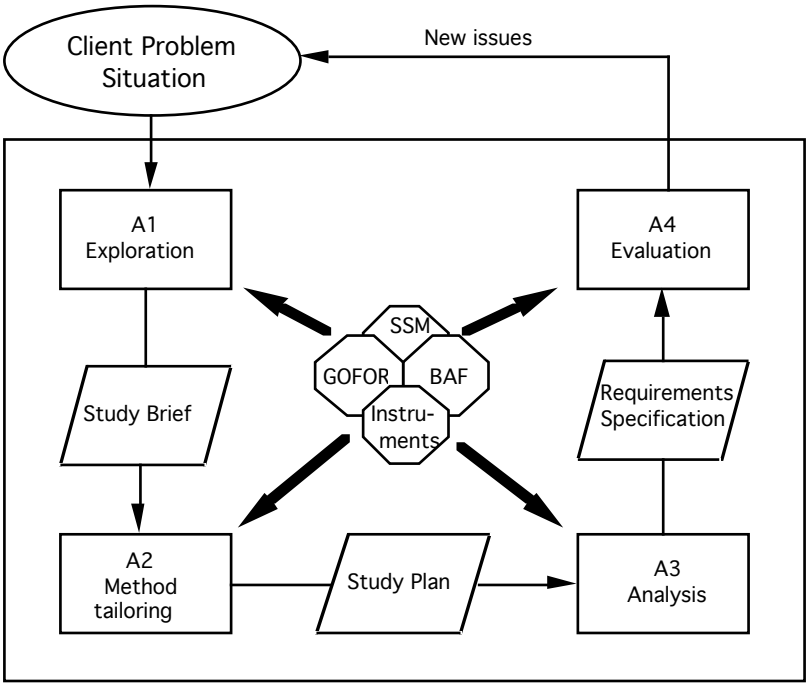

Figure 9 — The FAOR activity framework

SSM provides only general problem-solving and learning heuristics in FAOR. The more substantive support for office IS development is provided by the frameworks and instruments which can be applied flexibly in the process. From the viewpoint of modelling organisational action, the *Generic Office Frame of Reference* (GOFOR) and the *Function Analysis Instrument* (FAI) are most relevant. In addition to some general systems thinking (based on SSM) and modelling techniques (based on PETRI nets).

FAOR Subactivity	**Stage in Checkland's model (Checkland, 1981)**
Situational Analysis	**1.** The problem situation considered problematic **2.** The problem situation expressed
Systemic Analysis	**3.** Root definitions of relevant systems **4.** Conceptual models
Debate	**5.** Comparison of models and real system **6.** Systematically feasible and desirable changes
Action.	**7.** Action to improve the problem situation

Figure 10 — Matching FAOR subactivities to Checkland's Stages

GOFOR distinguishes seven perspectives (Schafer *et al,* 1988):

A function perspective

Three implementation perspectives

information processing, resource, and personnel perspectives

Three management perspectives

coordination, control, and re-organisation perspectives.

These perspectives are supported by a number of instruments. Which are used from each of the different perspectives. The different perspectives allow the system under study to be viewed in the following ways.

The function perspective: This deals with the 'purposiveness' of office work. It focuses on the work the office is intended to perform, pointing out that even though office work is normally heavily dependent on information processing, this is only a means, not the function of office work. As part of the method, Schäfer *et al* (1988) suggest a number of features to be analysed from the function perspective in the investigation of any particular office.

The implementation perspectives: The *information processing perspective* is interested in information flows between office units, i.e. the exchange and processing of office information objects (e.g. letters, reports, documents, data files, messages, instructions, commands, etc.). The *resource perspective* focuses on the required resources of office work. Schäfer *et al* (1988) suggest that resources are considered in line with a number of questions concerning the nature of resources and their relation to users in the organisation. Similarly, for *the personnel perspective*, Schäfer *et al* suggest a series of questions concerning the relation between people and the jobs they hold.

The management perspectives: The GOFOR framework includes three management perspectives: the *coordination perspective* which involves decision making and action under the goal of keeping the system coherent, the *control perspective* involving decision making and action under the goal of maintaining the functionality of the office during the day-to-day running of the office through supervision and instruction, and the *(re-)organisation perspective* involving decision making and action under the goal of building, changing or abolishing an office unit in response to a changed functional specification for that unit (Schafer *et al,* 1988).

Summary

The methods developed from SSM unsurprisingly adopt a similar approach to the uncovering of the needs of an organisation. However, given that SSM was not initially intended to make explicit the requirements of an organisation each have adapted the basic principles somewhat differently. An interesting feature of all of these methods is the general acceptance of the need to manage more than one perspective on the needs of organisations. This theme of multiple perspectives emerges through our later considerations of requirements techniques and mirrors many of the consideration of organisations within strand 1. In particular, a number

of recent development oriented techniques have focused on the explicit definition and resolution of multiple viewpoints.

Sociotechnical based approaches

Another approach to systems development which takes an essentially organisational perspective is what is often termed sociotechnical systems development (Cherns, 1987). The sociotechnical approach to IS development has its roots in the sociotechnical systems design initiated by the Tavistock Institute in the late 1940's. Rather than consider sociotechnical systems design in general, we wish to concentrate on the use of sociotechnical design approaches in information systems development. Fok, Kumar and Wood-Harper (1987) identify and compare three IS development methodologies having a strong socio-technical component: ETHICS developed by Mumford and her colleagues (e.g., Mumford, 1971; Mumford and Henshall, 1979; Mumford and Weir, 1979; Mumford, 1983), the methodology of Pava (Pava, 1983, Pava, 1986) and the methodology of Bostrom and Heinen (1977). This provides a useful starting point for our consideration of sociotechnical systems development.

The following sections examine ETHICS as an example of a traditional sociotechnical approach, and a more recent sociotechnical design approach to IS development proposed by Eason (1988).

ETHICS

ETHICS is the first and most influential of the three most prominent sociotechnical development methods. Olerup (1989) has written an excellent review of the sociotechnical design of computer-assisted work, concentrating on ETHICS but also including the Tavistock approach as a background to which interested readers are recommended. The development of ETHICS has been based from the beginning on three major ideas: sociotechnical design as the joint design ("optimisation") of the technical and social systems, a framework for job satisfaction, and the recommended participative approach.

ETHICS is largely based on the traditional sociotechnical view of a work system as a linear conversion process of routine work. The acronym ETHICS comes from Effective Technical and Human Implementation of Computer-based Systems. The method involves 15 major steps which are summarised below in Figure 11 (Mumford, 1983).

ETHICS is a highly rational approach in the sense that the specification and evaluation of the organisational and technical options is based on articulated goals and objectives. Organisational options (step 11) are explicitly derived from the key objectives and key tasks defined (steps 4 and 5). The reconciled efficiency needs, job satisfaction needs and future needs (steps 7-10) are used in the evaluation of the organisational options (step 11), technical options (step 12), merged organisational/technical options and the final implemented system in use (step 15).

More concrete modelling of organisational action in ETHICS is based on three major components: the horizontal input/output analysis used in step 3, the vertical

hierarchy of activities used in step 3 and 11 and at the more detailed level allocation of the responsibilities and distribution of the tasks used in step 13. Figure 12 summarises and illustrates the hierarchy of activities in the ETHICS method.

1. Definition of why change?
2. Identification of system boundaries
3. Description of existing system
4. Definition of key objectives
5. Definition of key tasks
6. Key information needs
7. Diagnosis of efficiency needs
8. Diagnosis of job satisfaction needs
9. Future analysis
10. Specifying and weighting efficiency and job satisfaction needs and objectives
11. The organisational design of the new system
12. Technical options
13. Preparation of a detailed work design
14. Implementation
15. Evaluation

Figure 11 — Summary of the ETHICS method

ETHICS expresses, in accordance to the sociotechnical tradition, a clear preference for the "autonomous" work groups or "self-managing" groups, even though the detailed steps of ETHICS summarised in figure 11 do not explicitly

Operating activities
What are the most important day-to-day tasks?
Passing receipts for payment
Dealing with rejected goods. This often involves asking suppliers to send a credit note
The payment of cheques
Problem prevention/solution activities
What are the key problems that must be prevented or quickly and easily solved?
Incorrect accounts being passed for payment
Incorrect payment for goods received
Co-ordination activities
What activities must be co-ordinated within the system?
Passing of receipts for payment of cheques
What activities must be co-ordinated with other systems?
Receipt of goods received notes from stores
Notification of rejected goods by stores or production units
Development activities
What activities, products, services etc. need to be developed and improved?
Better relationships with suppliers
Control activities
How is the system controlled now? (Targets set, progress monitored, etc.).

Figure 12 — The hierarchy of activities in the ETHICS method[32]

32 This figure is taken from Mumford (1983)

refer to them except in the context of the example of a Purchase Invoice Department used to illustrate ETHICS throughout the exposition of the steps. In the detailed work design Mumford describes the design process based on autonomous groups and unit operations, a unit operation being an integrated set of tasks separated from other sets of tasks by some kind of discontinuity. She points out that having autonomous groups based on carefully thought unit operations provides a number of advantages:

1. It gives individual support of a small, helpful team of colleagues.
2. It enables tasks within the unit operation to be allocated to meet personal preferences.
3. It provides excellent training for new staff who can begin with the routine work and progress to the more complex.
4. All members of a group can learn all tasks and eventually become knowledgeable and multi-skilled.

The emphasis of the ETHICS method is on establishing the nature of the work taking place and designing the way in which work is done to fit with IT systems. For example, it is stressed that each unit operation should be checked that the following good job design principles apply:

1. There are clear boundaries between the different operations so that each work group has a feeling of identity of its work.
2. The sets of tasks contained in a unit operation provide a good mix of simple, intermediate and complex activities.
3. The work group is able to solve a majority of its own problems without having to refer them to a supervisor or another group.
4. The work group is responsible for the internal organisation and co-ordination of its own activities (who does what).
5. The work group is responsible for developing improved methods for carrying out its tasks. It may also be given other development responsibilities.
6. The work group can set many of its own targets and monitor its own performance in achieving these.
7. The work group can easily identify the targets which it has to be achieve.

ETHICS pays little attention to the software development or construction problem associated with the realisation of software systems. Basically, ETHICS' strengths are in establishing the needs of an organisation rather than a specification of what needs to be build. However, the view of an organisation as autonomous work groups and the need to support these separate groups provides a useful means of considering the nature of organisations for CSCW systems development.

Eason's Sociotechnical Design Approach

Eason's sociotechnical design approach is based on ten propositions expressing the objectives of applying information technology and the conditions which must be

met in order to achieve the objectives (Eason, 1988). These are summarised in figure 13 below.

1	The successful exploitation of IT depends on the ability and willingness of the employees of an organisation to use the appropriate technology to engage in worthwhile tasks.
2	The design target must be to create a sociotechnical system capable of serving organisational goals, not to create a technical system capable of delivering a technical service.
3	The effective exploitation of sociotechnical system depends on the adoption of a planned process of change that meets the needs of people who are coping with major changes in their working lives.
4	The design of effective sociotechnical system will depend on the participation of all relevant 'stakeholders' in the design group.
5	Major benefits will only result if the sociotechnical developments are directed at major organisational purposes where there are opportunities to be taken or problems to be resolved.
6	The specification for a new sociotechnical system must include the definition of a social system which enables the people in work roles to cooperate effectively in seeking organisational purposes and provides jobs which incumbents perceive as worthwhile.
7	IT systems must be designed to serve the functional needs of the organisation by serving the functional needs of individual users in a usable and acceptable way.
8	The effective exploitation of IT requires a major form of organisational and individual learning.
9	The exploitation of the capabilities of IT can only be achieved by a progressive, planned form of evolutionary growth.
10	To be successful, sociotechnical design concepts must as far as possible complement existing design procedures and organisational change practices

Figure 13- Basic propositions of the Eason's sociotechnical approach

The last of Eason's propositions suggests a 'toolbox' approach to sociotechnical design which can be applied in different ways in different contexts to complement existing design procedures and organisation change practices. Eason's detailed exposition of the IS design process does not strictly follow the 10 propositions outlined as the basis of his approach, but he divides it in five interacting parts. These are summarised in figure 14.

For the purposes of this review we will focus on the second and third activities outlined by Eason.

Specifying the requirements

Eason divides the specification process in six stages depicted below (figure 15). He emphasises that it is essential that the analysis of organisational needs and opportunities does not begin by and is not concerned with examining current ways of undertaking tasks, existing information patterns and needs, but should use these as evidence of goals being sought and the problems to be overcome. He also points out that the fundamental objective of approaching the business needs via a sociotechnical framework is to ensure that the role of the social structure in mediating the information technology contribution to the business needs is not overlooked.

1. **Designing the design team and its design process**
   to plan who is involved in the design process and the responsibilities they take in the process
2. **Specifying requirement**
   identification and evaluation of requirements for a sociotechnical system
3. **Creating and designing solutions**
   creating social systems and designing technical systems that meet the requirements specified
4. **Implementing systems and supporting users**
   introducing the new system
5. **Evaluating the impact.**
   monitoring the progress made by users, individually and collectively, in exploiting the new capability and feeding back to a reviewing authority

Figure 14 — The activities of the IS design process

For the analysis of the organisational needs and opportunities he proposes a framework that makes a distinction between a functional view of the system (what the organisation is required to do) and the work performing system (how it currently does it) (Eason, 1988). The outputs that have to be produced constitute the primary goal of the organisation and the necessary transformations from inputs to outputs are described as functional transformations.

1. **The analysis of organisational needs and opportunities**
   to develop a statement of organisational needs and opportunities that is in a form which facilitate the search for sociotechnical systems, including those making use of new forms of information technology
2. **Specifying sociotechnical options**
   to produce broad conceptual proposals for potential sociotechnical systems
3. **Impact analysis:**
   Assessing the consequences of sociotechnical options
   a cost-benefit assessment performed separately for different stakeholder groups to provide feedback at the earliest possible stage before any major investment decisions are taken and changes are relatively easy to make
4. **Analysing the user/task requirements of a sociotechnicalsystem**
   to detail the requirements of the users and their tasks for the social and technical system to be designed
5. **The specification of a prototype system**
6. **The evaluation of the prototype system**

Figure 15 — The specification of user requirements

When most of the evidence available to the analyst is about the current way of undertaking the tasks he proposes the following approach:

1. Use a top-down approach; ask senior people about their objectives and responsibilities and examine then how these decompose to give objectives and responsibilities of their subordinates.
2. Concentrate on 'what' and 'why' (i.e. goals and requirements) rather than 'how' (i.e. current modes of operation).
3. Examine relations between functions such as sequential interdependences, reciprocal interdependences and pooled interdependences.
4. Look for variations in demand, inputs and environment.

Even though Eason puts the major thrust on the work requirements in terms of the functional transformations, he points out that it is necessary to create an outline view of the current sociotechnical system, because it forms the basis from which the organisation moves forward and the members must be able to see the advantages of any proposal over the existing structure.

In considering the specification of sociotechnical options Eason makes two points:

- the solution to the problem or opportunity that has been identified may not involve information technology.
- in addition to a range of application types being identified, there are also likely to be a variety of technical ways of serving the application and a number of possible social structures.

For the analysis of the user/task requirements of a sociotechnical system Eason suggests a technique called 'Open System Task Analysis' (Eason, 1988). It is based on the same concepts as the sociotechnical framework discussed above but is more focused and detailed (Figure 16). The analysis starts with the statement primary task of this particular work system, taking into consideration the possible hierarchy of goals, short- and long-term goals, task performance goals and system sustaining goals. The input analysis pays attention to the character, variation and predictability of inputs which largely determine the way the system has to behave in order to achieve its primary goals. Since the boundary of the open system task analysis is limited to tasks undertaken by individuals or work groups, the relevant environment may lie within the larger organisation which protects it from much of the external world turbulence, but if the work system lies at the boundary with the outside world, it may have to cope with the full range of external conditions. Eason suggests an action/object framework for establishing the functions that have to be undertaken to transform the inputs into required outputs, i.e. what are the objects that have to be transformed and what are the actions at each stage that can bring this about? In addition to the relations between activities (e.g. sequential, reciprocal, pooled and time interdependences), he advises to pay special attention to variations in transformations because of variations in inputs or goals.

The social system analysis comprises role analysis, indicating the current allocation of tasks between individuals, and user analysis, exploring the characteristics and qualities of the people who will ultimately become the users of

the new system. These include physical characteristics, skills/qualifications and sources of stress/satisfaction.

The analysis of the existing technical system takes place for two reasons. Firstly, because the new system may replace it in part, it may be necessary to analyse the existing one in order to ensure the compatibility between the new system and the remains of the old system. Secondly, Eason proposes that the current human–computer allocation should be recorded, since it indicates the degree to which the technology currently supports the users and will show the degree of change that will be necessary if the new system is to be adopted.

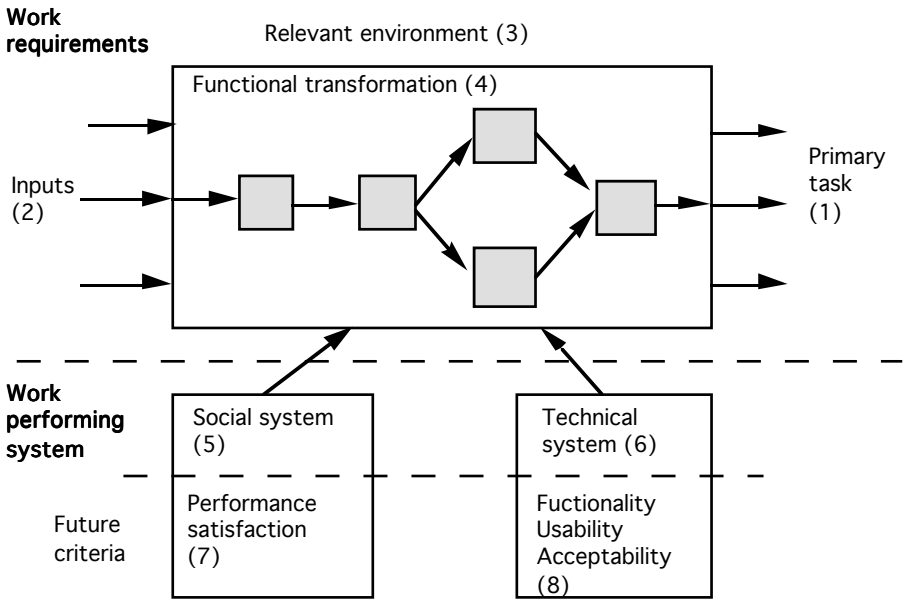

Figure 16 — The framework for an open systems task analysis

The analysis of the user/task requirements of a sociotechnical system also includes specification criteria for a new social system and for the new technical system. Eason points out that the criteria for the new social system should embrace criteria about the performance goals of the work system and the needs of the workforce for satisfying work. In the case of the criteria for the new technical system, he identifies functionality, usability and acceptability as three broad criteria. The usability criteria cover ease of use and ease of learning whilst the acceptability criteria cover the autonomy, control and responsibility issues.

The specification of user requirements is concluded by the specification of a prototype system and evaluation of it. In the case of the specification of the prototype, Eason points out that before proceeding to the development of a technical prototype it is important to specify the allocations of the functions of the proposed sociotechnical system between various members of the social system and between the users and the technical system.

Cooperative Processes and Procedures

The previous section considered organisational perspectives upon systems development. Some of these approaches considered the organisational context in which the computer system will ultimately exist as a structure or system in itself. Other sociotechnical approaches focused on the political and social issues in computer development within an organisation. Clearly the concepts of organisation, of work, of computer systems and the relationships between them is central to the COMIC research. This section explores traditions of research and development that share a common orientation around concepts of 'process'. The four approaches we shall consider — office automation, software process modelling, business process reengineering and coordination technology — each develop and apply the notion of process. Furthermore, in considering these approaches together we shall explore how the process construct has been developed in its application to computer systems and to cooperative work, to the process of developing computer systems and to modelling organisations in which computer systems exist.

While the four approaches share a common focus on the process concept, from the perspective of CSCW, the approaches in general (and individual projects within these approaches) can vary widely on points of central importance. Many issues explored within CSCW are also becoming much debated topics in process modelling communities. Additionally, while the four approaches are described here in a serial order, research in many of the approaches (such as Business Process Redesign and Software Development Process Modelling) are being developed in parallel.

We shall highlight significant aspects of these prominent approaches throughout the section and suggest influential themes which have emerged from these different traditions. Comments upon one approach are often relevant to discussions of the others. The approaches do, however, differ somewhat in their relation to IT system development and in how they assist developers in establishing the support requirements for cooperative work. Additionally the approaches vary in their domains of application, the scope of modelling and the levels at which the process construct can be made to work.

These four approaches consider cooperative work as a set of processes which work in unison to realise an agreed aggregate task. This 'work as process' view is a procedural conception in which work is said to be made up of individual communicating processes, and some of the earliest work adopting this perspective is that of the Office Automation tradition.

The Office Automation Tradition

A range of different approaches have emerged from the Office Analysis/Office Automation tradition. The clerical work undertaken in office environments has on occasions been seen as unproblematically 'routine' and a predominant focus has been upon the development of systems which automate a large part of the work taking place. Focusing on the analysis and representation of the work has been seen as a means to directly inform the construction of the information systems. The

models play a central role in expressing the processes occurring within the work setting.

Office Procedure Specification Language

An important and influential perspective on office analysis and description is Office Procedure Specification Language (OPSL) (Zisman, 1977) developed by Michael Zisman. This early work introduces the scope of phenomena and systems being considered. It seeks to describe:

> "office procedures which center around a flow of written communications. It allows an analyst to describe the way in which the system should react to the receipt of messages from the environment, what messages should be generated and transmitted, when these message transmissions should occur, what to do if expected responses are not forthcoming from the environment, and how the user's electronic files should be managed. " (Zisman, 1977)

The methodology contains two specification languages; an "external representation" (the OPSL) describing office procedures as a collection of activities and documents and an "internal representation" describing and specifying the internal control structures. Zisman defines the facility of the external representation (OPSL) based on the notion of a procedure and the processing of messages:

> "The analyst will be encouraged to develop the specification in terms of messages and activities which process these messages"

The procedure concept has a narrow definition in Zisman's terminology:

> "An office procedure is composed of a number of steps or activities. Activities are event-driven and are characterised by what we shall call a local focus of attention. It is a series of actions which occur in a single attention span, and normally by a single individual."

Zisman (and the methodology) is mostly focused on identifying and describing the concurrency of the work flow:

> "The user [of OPSL] is forced to consider the conditions which must be met before a task can be initiated. By specifying sufficient pre-conditions for initiation, the user implicitly provides information about concurrent execution of tasks"

As mentioned above the methodology consist of two specification languages. The second — the "internal representation" — is called Augmented Petri Net (APN) and used to

> "facilitate the use of production systems for modelling asynchronous, concurrent processes. The production rules of the APN succinctly capture the event driven nature of office procedures." (Zisman, 1977).

Additionally the methodology offers a procedure for relating the external representation (OPSL) to the internal representation.

According to Zisman the methodology is intended for use in non-procedural domains (Zisman, 1977). However, it appears that it may be most useful in describing domains characterised by being well structured (an example may be a domain where procedures exist for, say, sending standard letters when very exact conditions are fulfilled). We shall discuss later how the views of the 'routineness' of office work have been changed by, for example, field-studies of actual work.

In this approach an understanding of the character of office work is based on "discussions and interviews with managers in large, communication based organisations" (Zisman, 1977). There appears to be little in the way of suggested procedures or techniques for supporting practical analysis work in the field. Accordingly, OPSL is really only a modelling language and specification technique for the operational part of the analysis phase. Consequently in this way it might be said to not constitute a full analysis methodology.

The method is restricted to modelling sequences initiated by events, and in these cases only modelling the document exchanges. While this could be useful in the construction of case-handling systems and archives (and for this purpose it might be very useful indeed as it focuses on relations between events, activities and concurrency), it is consequently limited to specific parts of the typical office domain.

Information Control Net

Another, even more influential approach is Information Control Net or ICN, developed by Clarence Ellis and others at Xerox PARC in the late seventies (Ellis, 1979).

ICN is a mathematical based technique that focuses upon defining a model ("normal form description") of an office and using this model as a basis for deriving alternative organisational models of the same office. These alternative models or patterns of organisation of cooperative work may be more or less streamlined ("minimal forms"):

> "These minimal forms of the office description show the basic necessary information flows and the invariant information requirements that must be met in any realisation of a specific set of office functions" (Ellis and Nutt, 1980).

The perspective in the modelling techniques is very similar to that in Zisman's OPSL. Both are based on a flow (or stream) paradigm and describe the flow of information objects (e.g. formulas, messages or documents) among distributed procedural activities. The central concept in the methodology is defined as:

> "The Information Control Net defines an office as a set of related procedures. Each procedure consists of a set of activities connected by temporal orderings called precedence constraints. In order for an activity to be accomplished it may need information from repositories, such as files and forms" (Ellis and Nutt, 1980).

ICN, in comparison with OPSL, attempts to include some more of the 'complexity' of office work in the model. For example the ICN model has an explicit distinction between control structures and information structures. Additionally the analyst can express possibilities and expected processing times (Cook, 1980).

On the actual effectiveness of the methodology for reconstruction (or "streamlining") of office procedures, Ellis claims that ICN can be used:

> "to test the underlying office description for certain flaws and inconsistencies, to quantify certain aspects of office information flow, and to suggest possible office restructuring permutations. Examples of office analyses that can be performed via this model include detection of deadlock, analysis of data synchronisation, and detection of communication

bottlenecks. Restructuring permutations that can be performed via this model include parallelism transformations, streamlining and automation" (Ellis and Nutt, 1980).

and that:

"some of our mathematical office analyses which automatically detect potential parallelism have produced extremely useful results" (Ellis, 1983).

Further Ellis claims that ICN provides "a comprehensive description of activities" but in practice, defining offices as sets of related procedures tends to produce descriptions in terms of simple non-overlapping processes. Indeed, the later work of Ellis (Ellis 1983) includes a reflection upon some of the complexities of office work and of modelling this domain. We consider some of these points below, but it is worth noting that Ellis suggests elaborating ICN by introducing means-ends hierarchies into the methodology. The inclusion of structures for means-ends hierarchies may assist in the use of ICN for considering the operational and functional levels of analysis.

Office Analysis Methodology

Office Analysis Methodology or OAM was developed by Michael Hammer, Marvin Sirbu and others at MIT's Laboratory for Computer Science (Sirbu *et al,* 1981; Hammer and Kunin, 1980). In contrast to ICN, OAM is not dedicated to informing the development of office automation systems:

"Most earlier approaches, intended to result in a single highly-structured computer program, placed great emphasis on capturing operational details. We are not trying to directly specify computer systems, but rather the requirements of office functions, which might be implemented via numerous different man-machine systems." (Sirbu *et al,* 1981).

The objective for OAM is to develop a deep understanding of office work. The application area of OAM is aimed explicitly at 'non-trivial' parts of office work:

"The methodology proposed here is designed to address the semi-structured problems in managerial and operational areas that form the bulk of modern-day office work" (Sirbu *et al,* 1981).

In focusing on these aspects of office work, the approach proposes the development of "integrated functional support systems". In other words the approach looks to systems that, in an integrated manner, support the spectra of procedures and decisions carried out in order to fulfil a specific business oriented objective (Sirbu *et al,* 1981). Hence, one difference from previous office methodologies lies in the focus upon the functions of the office related to goals:

"Our purpose in studying an office is to develop a description of the functions that it performs in support of the business goals of an organisation" (Sirbu *et al,* 1981)

"a constant theme in our analysis process is the question 'Why?'" (Hammer, 1982)

The office work is modelled as a number of dimensions of knowledge and constraints in which an office worker is said to be navigating during the work. These dimensions are drawn in relation to the functions fulfilled and these functions are again related to the office systems context. The methodology results in descriptions of plans or strategies. The level of work analysis then becomes strategic or functional.

Interestingly, advocates of OAM suggest that one of the problems in this form of analysis can be the influence that existing organisational structures exert on the formulation of the analyst's definition of the (cooperative) work arrangement. They comment that, for many good reasons, it is often easiest to start the analysis in a certain organisational unit. However, they note that often a single unit does not reflect the performance of a function. It is important, they therefore suggest, to identify all organisational units that are relevant to the fulfilment of a function, and then analyse all of these.

In contrast to OPSL and ICN, OAM is a methodology. It is not related to specific modelling languages, although it is developed in connection with "*Office Specification Language*" or *OSL*, which is "a high-level, problem-oriented language for describing offices in a precise and non-ambiguous way" (Sirbu *et al,* 1981). OSL has a basic distinction between structure and process:

> "An office specification in OSL consists of two major components: a description of the application domain with which the office is concerned, and specifications of the procedures performed in the office." (Hammer and Kunin, 1980).

The model of the field of work ("office environment" in OAM terms):

> "effectively expresses a model of the world of the office; it describes the objects on which the office is focused, the organisational context of the office, the documents and forms that the office processes, and the information it needs to utilise." (Hammer and Kunin, 1980).

In this sense and compared to previous techniques for modelling office work, OSL is a useful development in capturing central aspects of the rich semantics of the non-trivial parts of office work in a problem-oriented, object-oriented language.

It should be noted that an OAM analysis not only models the "local office context" of people, their roles, responsibilities and authority together with entities and their attributes, inter-entity relationships, and entity-collections but:

> "Included in this environmental specification is a description of the offices in the organisation and the lines of communication and authority that connect them." (Hammer and Kunin, 1980)

Identifying the organisational context and central functions of a cooperative work arrangement is to be achieved via interviews of the management of the organisational unit and by structured observation (Sirbu *et al,* 1981). Furthermore, it is suggested that the analyst must identify the procedures that implement the functions and the "objects" processed by the procedures.

In a detailed analysis of the procedures OAM recommends a separation between the *intentions* of the procedure and the *exceptions* from the procedure. One aspect of this distinction is an abstract treatment of specific existing circumstances and actual implementations. We have seen how Sirbu *et al* (1981) talk of the "requirements of office functions, which might be implemented via numerous different man-machine systems" and Hammer plainly states that:

> "Our description of a unit's operations are expressed in high-level implementation-independent terms" (Hammer, 1982)

They suggest that a deep understanding of office work may be 'disguised' or 'confused' firstly by a focus on the "artefacts of current operation", and secondly

by focusing on the day to day activities of office work rather than its purpose to "realise a mission". Hammer and Sirbu (1980) remind us that:

> "It is the rare office whose goal and function is to produce letters"

A further aspect of OAM analysis is the identification of kinds of exceptions and kinds of necessary modifications to procedures. It is proposed that such identifications can assist the analyst in better understanding the decision situations actors are involved in. However, this does requires a consideration of the definition of the concept 'procedure'. Possible separations may exist between 'regulations' (expressing the contextual requirements), 'routines' (expressing existing practical experience) or 'algorithms' (expressing the "correct" logical method).

While Sirbu *et al* (1981) recognise:

> "the myriad of exceptions from procedures happening in daily practice (and that)... Classifying exceptions into basic categories is another important technique for imposing order on the unruly world of office operations"

Nevertheless, the separation and categorisation of exceptions (and procedural modifications) that is advocated can tend to presuppose that the number of exceptions and the number of kinds of exceptions are in fact limited. This is often not the case in non-trivial office work. In this way the methodology tries to produce not only descriptions of "typical" sequences but of typical exceptions to these sequences. We shall consider later how the development of CSCW systems may require additional consideration of exceptions that may be 'non-typical'.

OAM can be used in domains where the work is performed using explicitly formulated procedures and where it makes sense to distinguish between intention and exception. The developers of this approach also honestly admit areas where OAM will have difficulty producing a usable identification and description of the central aspects of the cooperative work arrangement. As phrased by the authors:

> "OAM is particularly appropriate for offices in which the execution of procedures is of primary importance, as opposed to the design and monitoring of office functions. It is not well-suited, for example, for studying an R&D laboratory, or a long-range planning office. In addition, the methodology, at this stage of its development, does not adequately deal with the 'meta-level' tasks associated with managing a procedure." (Sirbu *et al,* 1981).

As noted above we would wish to add to this that the approach often tends to presuppose that a domain may be entirely characterised as containing a limited number of exceptions from the explicitly formulated procedures. Nevertheless, the OAM approach contains a number of relevant and useful ideas on office analysis. It augments earlier procedural views of office work and it is appropriate to reflect on some themes contained within (what we have referred to as) the Office Automation tradition.

Summary

We have seen how early work focused mainly upon modelling representations and concerns over representational adequacy. Zisman (1977) concentrated on how to use two description languages to capture both semantics and the flow of control. This is an important issue, however, an understanding of the path from model to

informed design, and an understanding of the methodology of analysis, remain less developed in this research. Furthermore the work tended to assume a desire to automate work.

We have seen how Ellis and colleagues (Ellis, 1979) introduced the use of formal abstract representations as a basis for the automatic detection of certain patterns of work, such as parallelism. Furthermore they propose to offer the potential for suggesting alternative organisational forms and for improving the actual working practice. In particular they suggest ideas of streamlining or producing 'minimal forms'. In this way they suggest the use of metrics and diagnostic reasoning as a precursor and aid to redesign. In this notion of redesign, automation is but one possible form of change.

However, we may note that Barber *et al* (1983) when discussing office modelling techniques and languages such as OPSL, ICN and OSL, suggest that:

> "In all these systems information is treated as something on which office actions operate producing information that is passed on for further actions or is stored in repositories for later retrieval. These types of systems are suitable for describing office work that is structured around actions (e.g. sending a message, approving, filing); where the sequence of activities is the same except for minor variations and few exceptions. None of the above systems explicitly describe the goals of office work and how each action is related to the accomplishment of the overall goal of the work. Thus it becomes difficult to describe work where the goal can be achieved via several different methods or where the actions necessary to accomplish the goal cannot be known ahead of time. These systems do not deal well with unanticipated conditions.

In the OAM approach we see a greater discussion of notions of 'business goals', 'integrated systems' and 'support systems'. Hammer's later work in Business Process Redesign will be considered below, but in many ways the themes of this later work emerge from that described above. While the OAM approach is part of what we have called the Office Automation tradition, in that it still works within the modest scope of the office domain, nevertheless it considers what the office can do for the organisation, what functions it can offer to the context. We see functions rising in importance above organisational units such as offices; it is said that many such units may be required to perform one function. In this way we see talk of "overall" functions and offices carrying out "stages" of this function. We also see in both ICN and OAM attempts to produce an implementation-independent description abstracted from only the concerns of IT system development. Analysis and modelling becomes valued in and of itself.

A central issue to emerge from considering attempts to model and automate office work is in what sense may activities be 'routine'. Some have characterised offices according to the levels of routineness they display, suggesting that this may be associated with degrees of automation that are appropriate. Zisman distinguishes between management analysis-based 'responses' to stimuli and clerks 'reactions' to inputs:

> "By 'react' we mean that the action to be taken for a given stimulus is to a high degree predetermined by the organisation's management. Once a clerk is told about a situation, s/he can consult a predefined procedure (formally or informally) to determine what action should be taken by the organisation. The organisation does not rely on the clerk to *decide* what to do;

> instead the organisation provides a procedure which instructs the clerk how to react to the situation." (Zisman, 1977).

In OPSL there is also little room for judgement and interpretation as the model only implicitly represents non-procedural decision process. In contrast, early empirical studies of office work, for example Suchman (1983), have highlighted how the decisions and judgements necessary even in 'routine' office work play a central role in the functioning of an office.

In the OAM work there is some recognition of the problematic aspects of work and the existence of exceptions from defined procedures. However, it is those 'typical' exceptions, suitable for separation and categorisation, that are most comfortably admitted into the modelling. Unfortunately, predictable 'exceptions' appear to be more like 'options' than to be responses to unanticipated conditions and events. The approach also recognises its limitations in accounting for the articulation of cooperative work arrangements and that OAM "does not adequately deal with the 'meta-level' tasks associated with managing a procedure".

In many ways office automation systems have not had the impact or acceptance expected but the experience of attempting to model offices as processes has led to a greater awareness of the issues facing those hoping to model cooperative work and inform the development of computer support for such work. Ellis, for example, writing in 1983 and reflecting on his earlier work with ICN, suggests the following areas were the office domain is perhaps more complex than initially expected:

> [1.] People Systems — An office is a social environment to which any introduction of procedural change, goal changes, or automated equipment causes perturbations. Explicit consideration should be given beforehand to analysing likely effects of these changes. Many technologically successful systems have failed due to ignorance of human and social factors. […]
>
> [2.] Dynamic Systems — Change is frequent and expected in most domains of the office; an employee's vacation days, for instance, force others to change their routines accordingly. Change also results from promotions, employee turnover, competition's changes, sickness, changing government regulations, etc.
>
> [3.] Concurrent Systems — The office is a highly parallel, highly asynchronous system. […] As a word of caution, we have observed that it is not always easy to discern whether an activity falls into the category of unnecessary relic or necessary redundant checks and balances."
>
> [4.] Ill-Structured Systems — [M]ost of the people resource is within the 'knowledge worker' category. […] Much of the work performed by this group is only semi-structured if structured at all. These workers need augmenters and aids rather than the structured data processing systems which are prevalent in more structured parts of a business. For example, a sales manager […] may need to search diverse data, read between the lines of a report, and have a confidential lunch meeting with a colleague in order to track down the information necessary to salvage the account of a big customer. *A useful office model must be able to represent these types of unstructured activity and interpersonal relations.*
>
> [5.] Open-Ended Systems — Another important group of people is the 'clerical/secretarial' category. One might assume that the work of people in this category is all structured, but office studies [by Wynn, Suchman, Fikes etc.] have shown that even within this category, the amount of problem solving, exception handling, and customer interfacing (all three are unstructured activities) are quite high. […] *Thus systems and models must again be able of handling a*

diverse spectrum of activities with high proportions of semistructured and unstructured activities. Models cannot capture all of the exceptions, and furthermore should not be based upon the premise that this is possible. Keeping in mind that models are limited abstractions of reality, it must be expected that procedures are open-ended with inherent escape hatches to handle unanticipated exceptions and emergencies. Our approach at Xerox has gravitated toward the use of distributed systems and artificial intelligence techniques to provide automated knowledge based assistants, rather than toward notions of total automation of procedures and offices." (Ellis, 1983 — our italics).

Understanding the requirements for developing successful CSCW systems will require a serious consideration of the points outlined by Ellis. The notion of 'process' as a basis for modelling the work of groups is not unique to the office automation tradition. It has a prominent place within the software process modelling community and a number of startling similarities in approach have emerged.

Software Process Modelling

The intent of much of software process modelling investigation is to develop appropriate means of managing the software development process. It is interesting to note that the work of software development, as in the case of office automation, is highly cooperative and centrally dependant on the activities of people. As a result, members of the process modelling community have invested in modelling the activities undertaken to realise a software system. Much of this work has resulted in the construction of generic models of the software process which provide a framework for understanding and managing software development. These process models have traditionally considered the activities involved in software development in isolation of any particular application domain and as abstract tasks which need to be completed to realise a software system.

An historical focus has been to develop an abstract statement of how things should be done in order to manage the software development process effectively (Osterweil, 1989; Krzanik, 1989; Ambriola, 1992). (Many models include either an implicit or explicit understanding that some form of development method such as Structured Development has been applied. The assumption is that developers follow a set of prescribed activities to realise the finished software system). However, some authors have distinguished between modelling processes and supporting them:

"The purpose of process modelling is to facilitate understanding of the process and communication about it, and to provide a basis for analysing it and remodelling it. Process support involves the use of computer systems to improve the performance of the process within the organisation. The general requirements of a technology to assist process modelling are not the same as those for a technology to provide process support". (White, 1992)

White was writing about business processes, as we shall consider shortly, but such definitions can be applied to Office Automation and Software Development Process Modelling as well. Hence for some a process model provides a framework to understand, manage, etc. the work of software developers (discussed below and earlier in this Deliverable); while for others process models also provide a basis for

developing systems to support software development. We shall consider both these aspects starting with process modelling as a means to better understand software development work.

Understanding Processes

Defined "simplistically", by Krasner *et al* (1992), a software process model, in the most general sense,

> "defines how an identified process is to be performed in the context of the goals, objectives and constraints of a project or organisation (i.e. why). The model contains prototypical sequences of tasks that must be done in order to accomplish desired goals.... A process model provides the framework for producing plans that can be replicated for specific software development projects. It does not, however, provide the detail necessary for individuals to produce specific products of a desired quality for a specific cost or in a defined timescale."

The exercise is, of course, one of modelling, and is thus directed toward providing a representation of the software development process which is simplified relative to the processes that are to be modelled. The project of process modelling appears to be directed at an incremental improvement on available methodologies, seeking to provide a corrective to the familiar weaknesses of such models.

The objective of process modelling is to enhance the standardisation of understanding and practice within the software development process by providing those within the team with a shared, explicit and comprehensive representation of the process. One which is itself capable of accommodating the processes such as opportunistic design, learning, technical communication, negotiation and customer interaction which Curtis *et al* (1988) concluded are not normally acknowledged in, or adequately provided for, by existing methodologies. Though the modelling is intended by some to assist the automation of the design process, it is also required to acknowledge that the actual development process is one which involves people working with each other and using technology as a means for their collaboration, thus enabling that attention not be exclusively focused upon the technological system.

The process model will, itself, be an evolving construct that will be developed on the basis of project experience, and it is meant to facilitate the accumulation of such experience by getting it on record. The process modelling is thus thought to provide improved techniques of project planning and project management, for it will — when equipped with details of a specific project — generate prescriptions for the carrying out of the project and, through the identification of interdependencies, indicate potential sources of trouble.

The process model can be the basis for the automation of the representation of the development process, desirably one which is more accessible and intelligible than that which is now contained in extensive and unwieldy documentation, but also to provide a representation which is more effective in that:

> "capabilities such as levels of abstraction, multiple perspectives, and precision offer distinct advantages over representations based upon narrative text or a single type of diagram" (Kellner, 1989)

It should be added that the multiple perspectives are to be systematically interrelated.

The essential feature of such modelling is, of course, that it seeks to define development as a sequence of processes, rather than just as a set of objectives to be met, and thus to improve the understanding of developers and managers of the processes necessary in the achievement of the usual objectives as specified in scheduling and milestones. The understanding of the processes and their exigencies allows the possibility that these can be taken into account in the planning process in conjunction with the imperatives of cost and schedule. Much of the information necessary for this must, however, be available in quantified form but even qualitative representation will be useful for the development of the model for the organisation as another 'forcing device'.

The formal character of the model is said to ensure that the attempt to enter information into the model will reveal ambiguities, inconsistencies and uncertainties and thus will call for the clarification/resolution of these matters, which will involve the redesigning of the processes being modelled.

The inputs to these models (as identified in Kellner) are

(i) descriptions of current — 'as is' processes

(ii) prescriptions of desired future "to be" processes i.e., process definitions, and

(iii) restrictions imposed by regulations, standards and directives

and the capabilities for handling these systems would be

1. Use a highly visual approach to information representation
2. Enable compendious models i.e. comprehensive in scope, yet concise in presentation
3. Support multiple, complementary perspectives on a process, such as functional, organisational, behavioural and conceptual data modelling
4. Support multiple levels of abstraction for each perspective
5. Offer formally defined syntax and semantics
6. Provide comprehensive analysis capabilities
7. Facilitate simulation of process behaviour directly from the representation
8. Support the creation and management of variants, versions and reusable components of process models
9. Support the representation and analysis of constraints on the process
10. Enable the representation of purposes, goals, rationales etc.
11. Integrate easily with other approaches (such as PERT diagramming or E/R modelling)
12. Take an active role in process execution, and
13. Offer automated tools supporting the approach

Supporting Processes

Process modelling work has also considered software development as a set of procedures (Osterweil, 1987) which can be used to control software development.

Generally, the software process has been considered in terms of an abstract model which is instantiated for a particular setting before being enacted to manage the use of tools within a process enactment environment. These models have tended to consider the software process purely as communicating functions undertaking independent activities to realise effective software development. Common descriptions of the software process used within the literature highlight the procedural nature of these models:

> "Software Processes are software too" (Osterweil, 1987)

> "A software process model defines the activities performed in a software engineering environment in order to produce a software product." (Bandinelli *et al,* 1992)

Most projects within the process modelling community can be characterised as fulfilling one of a number of goals

- Adopting an abstract representation of the software process.
- Developing formalisms to express this representation.
- Constructing environments to enact the expressed abstract model of the process

As we saw with some approaches to developing office systems, much of the study of software development processes has given little consideration to the methodological aspects of the production of the abstract specification of software process. What is more evident is the development of appropriate formalisms to represent the cooperative work taking place as part of developing software. Indeed a wide range of formalisms have been used to describe process models, including Petri Nets (Bandinelli *et al,* 1992), Communicating Sequential Processes (CSP) (Greenwood, 1992) and logic based systems (Ambriola, 1992). Three general categories of formalisms exist:

- Descriptive or rule/trigger-based (MARVEL (Kaiser *et al*, 1988) and ADELE2 (Belkhatir *et al,* 1991))
- Network based (MELMAC (Dieters and Gruhn, 1990), Petri Nets (Bandinelli *et al,* 1991))
- Imperative or programmatic Process Modelling Languages, PML, usually interpreted (APPL/A (Heimbigner *et al*, 1990), IPSE 2.5 (Warboys, 1990))

Two of the most generally used examples of process modelling formalisms are SPADE (Bandinelli *et al,* 1992), which uses petri nets to represent the software process, and the process modelling language SPELL (Conradi *et al*, 1990). However in general little consideration or guidance in Software Development Process Modelling is given to the actual application of these formalisms.

Finally some, such as Finkelstein (1992), have suggested that a by-product of the seductiveness of the notion of software development processes as 'enactable' has been the dominance of 'environment' applications (see for example, Bruynooghe *et al,* 1991; Fernstrom, 1993; and Warboys, 1990). There are a class of problems (which in many respects relate to central issues within the CSCW field) associated with constructing environments to 'enact' the expressed abstract

model of the process and to 'instantiate' a software process. For example, we may ask in what way may models of processes be a basis for building process support tools, or how might one view 'enactment' and how might one see the relationship between a model and some activity. These are concerns for process modelling in general and are discussed below.

Summary

In many ways the Office Automation tradition and the Software Development Process Modelling approach are about the production of plans. Where there is an initial consideration of activities, as perhaps in the analysis of offices, modellers have sought to add goals and means-ends hierarchies. Where there has been consideration of goals and milestones, as perhaps in the analysis of software development, modellers have sought to add descriptions of the activities and processes performed to achieve those goals. In this way plan based representations of cooperative work are developed in both the Office Automation and the software Development Process Modelling approaches. They share many similar formalisms for these plans. These included Petri Nets (De Cindio *et al,* 1986) and Information Control Nets (Ellis, 1979).

We have also seen how the assumptions of 'routineness' evident in early office modelling were questioned by field studies of actual office work (Suchman, 1983). Such results apply to software development as well, where judgement and decision making are an essential part of even the most 'standard' tasks. However, the work of Suchman (1987), for example, suggests not only the interpretative nature of work but that such judgement is situated in the context of work and dependent upon prevailing circumstances. In this way the very relationship between plans and actions, as a relationship of 'execution', is questioned. It suggests that, for example, plans do not get instantiated themselves but are applied by persons within work settings. This application is dependent upon the skills, knowledge and understanding of the person within the domain-at-hand. In this sense, rather than being procedural instructions to be literally interpreted or 'enacted', plans are resources to be used by users to co-ordinate and manage their work in conjunction with their experience and expertise.

Underpinning much (though not all) work in process support for software development has been an acceptance that a procedural abstraction of the software process effectively describes software development. In this way it is thought that the representations of how software developers work together as a set of abstract processes could be unproblematically instantiated and enacted. Such a view has consequences, for example, in the development of support tools and as to whether these tools are expected to support anything other than the execution of the process as represented in the process model. In contrast, some, for example Robinson and Bannon (1991), have suggested that:

> "Rather than viewing the models embedded in designs as implementable accounts of work processes, they should be conceived as heuristic devices that provide resources on particular occasions for particular groups of people" (Robinson and Bannon, 1991)

Studies of software development need to pay greater attention to the kinds of day to day problems as they actually occur in software development and to consider the relationship between the generality of an abstraction and the actual nature of software development. Furthermore, the use of such a representation within a development team, perhaps even a small one, would involve considerable problems of coordination with respect to its own use.

A number of further points are worth restating from our consideration of software development process modellers. They have usefully identified the need for integrating multiple perspectives upon work and work situations. This is both in terms of relating formalisms that capture functional, organisational, behavioural and conceptual information and also in terms of recognising differing viewpoints of 'stakeholders', 'roles', 'agents', etc. Also the theme of redesign is continued together with a conception of a process model as an evolving construct that records the 'maturity' of the understanding about the development activities (Kellner, 1989). Such a representation is seen as potentially re-usable and as generative of plans for undertaking specific projects. We might also note how the modelling process itself is seen as a useful exercise in consensus generation, and this is a theme discussed below.

Business Process Reengineering

Business Process Reengineering (BPR) makes a call for boldness in a search for dramatic improvements beyond earlier forms of process rationalisation and computerisation in which old ways of organising business were simply automated and speeded up. Hammer (1990, *'Reengineering Work: Don't Automate, Obliterate')*, in a development of his earlier work on office systems, urges us to "stop paving the cow paths" and "merely rearranging the deck chairs on the Titanic". Such calls ground themselves in a mandate for change enabled by the new capabilities of information technology, associated reductions in particular business overheads, and a perception of a new age with a different competitive environment.

The old era is associated with a time "before the advent of the computer" when job designs, work flows, control mechanisms and organisational structures were geared towards efficiency and control. Today's environment is portrayed as rewarding innovation, speed, service and quality. It is seen as a potentially hostile environment requiring radical, indeed revolutionary, changes in order for traditional firms to survive in the face of competition from "sleek startups or streamlined Japanese companies". Present processes and standard operating procedures are said to be 'traditional' and rooted in the past of the industrial revolution.

The BPR approach draws upon work in Office Automation and Software Development Process Modelling, yet differs in ways we shall highlight. It is a more recent and on-going exploration of the potential to use some of the abstractions considered above, in modelling organisations. An initial orientation to the approach is perhaps best provided by a consideration of two stories from the business process reengineering folklore.

Cases of BPR

The Ford story

Stories are told of a reengineering exercise undertaken by the American car manufacturer Ford Motor Company that began with an intention to cut costs by reducing the 'Accounts Payable' department from 500 people to 400 people by means of introducing new computer systems and rationalising processes (Hammer, 1990). Ford was motivated towards a more radical reengineering by discovering that the slightly smaller but compatible company, Mazda, employed only 5 people in their accounts payable department.

Redesign involved looking outside of the Accounts Payable department to the whole process of purchasing. Originally, or so it is told, purchasing involved the Purchasing Department sending an order to an external vendor and a copy to Accounts Payable. Later Material Control received the goods and sent a receiving document to Accounts Payable. The vendor also sent an invoice to Accounts Payable. Accounts Payable matched these and spent much time sorting out mismatches. Ford paid when they received an invoice which matched a purchasing order and matched a receiving document.

The tale is completed with a description of how Ford introduced a mixture of enabling information technology and changed work practices and cooperation networks to improve efficiency and costs by 75%. Invoiceless processing now involves the Purchasing Department entering information into a database which is checked by Material Control when goods arrive at the receiving dock. Goods that match orders are accepted (others are returned). Matching is done by the computer which also prepares the check which is sent to the vendor. Ford has asked vendors not to send invoices to them (some vendors still generate them for their own purposes and throw them away when they have been paid).

The IBM Story

A further case much quoted in Process Modelling circles is that of IBM arranging loans for customers to buy their computer systems (Hammer and Champy 1993). In taking between six days to two weeks for a loan approval customers had time for second thoughts and to change their minds.

The sequence of arranging a loan included logging the deal, going to the Credit Checking Department, then to a Business Practices Department, then to the Pricing Department to calculate the charge rate, then to the Administration Department to produce an offer of finance.

Following one deal right through its course IBM discovered that actual work on a deal only took 90 minutes. Doubling productivity by improving the existing process would have only reduced the time taken by 45 minutes. The boundaries and interfaces between departments were seen as a cause of delay.

It is recounted how the approval process was re-engineered so that one person handled each application and the time was cut from six days to four hours. Applications for loans were no longer seen as unique and routine cases requiring no expertise were separated from more unusual instances. The redesign story ends with more deals being handled by less people and expertise being used more appropriately.

Clearly, these 'before and after' stories offer only a certain sort of description, remaining silent on other aspects and outcomes that perhaps do less to promote the positive effects of the consultant's intervention. Not surprisingly, failure in the form of increased overheads, breakdowns, increased loads on other agents, etc. are not much discussed. Some old themes of 'paperless office' are also evident in the stories. Nevertheless, features of these tales of redesign illustrate a new perspective

of process modelling found in Business Process Reengineering. A number of the features of this approach are worth highlighting.

In a development of notions we described above as part of the OAM approach, the BPR perspective seeks to go beyond boundaries of departmental or compartmental thinking and advocates a cross-functional view. It attempts to 'reverse' the products of the ideas of the industrial revolution about organisational structure,

> "For the most part, we have organised work as a sequence of separate tasks and employed complex mechanisms to track its progress. This arrangement can be traced to the Industrial Revolution, when specialisation of labor and economies of scale promised to overcome the inefficiencies of cottage industries. Businesses disaggregated work into narrowly defined tasks, reaggregated the people performing these tasks into departments, and installed managers to administer them" (Hammer, 1990).

BPR is, however, a significant extension of the ideas in OAM, where a limited consideration was given to the purpose of an office in terms of its organisational functions and goals. Here in BPR the analysis is said to begin with organisational goals; processes are to be primary and the infrastructure reconstructed around them. Furthermore, the source of organisational goals is identified in BPR. It is the customer-performer relationship, one perhaps orthogonal to traditional departments, that becomes the key to setting goals. Departments are demoted and processes that satisfy customer-performer goals are to be allowed to retain a coherence. It is these that are made to be central to the organisation. It is these that should be organised around and the infrastructure adapted to.

In the IBM story, the successful outcome is described as arising from approaching loan issuing as an end-to-end process rather than looking to improve separate departmental functions such as credit checking — to pave the cow paths. Similarly in the Ford case making dramatic improvements is said to have become possible, not by looking under the light, but by looking outside of the Accounts Payable Department and to the wider picture.

In addition, we have seen how in the IBM example the departmental boundaries were viewed as a cause of delay. BPR recognises possibilities for interfaces to be problematic, to cause delay, to be complex and demanding, to be 'incomplete' and to affect quality (De Michelis, 1993; Lappo, 1992; Kawalek, 1993). Hence one of the central concerns of process modelling, and one of relevance to understanding cooperative work, is its attention to interactions between people, processes and departments (e.g. waiting, checking, matching, distributing, etc.). The notations, diagrams and tools being developed for business process modelling provide overviews for understanding 'the wider picture' but additionally they focus upon representing identified interfaces.

However, as a note of caution, this call for process primacy and the removal of departmental divisions can cloud the fact that, rather than remove boundaries, it is often necessary to redraw them. Furthermore, before redrawing boundaries it is of course necessary to 'identify' these multiple interfaces and boundaries, an issue discussed later (see also Bowers and Rodden, 1993).

We see a reconception of an organisation's internal activities in which the term 'process' refers to end-to-end customer-performer inter-departmental threads rather than to activities within a functional department. (In other words, units — for example a 'Finance Department' — are not merely to be renamed — as say a 'Finance Process'). Modelling the process threads starts from the organisation's interactions with its external customer environment from which the process goals are located. The process perspective then duplicates and decomposes interactions, processes, boundaries and interfaces within the organisation as an internal market organised around the top-level processes. The input output transformation through the black box of the organisation is opened up into many such subprocesses and represented with multiple levels of abstraction — looking inside the organisation we find many more customers. Going the other way, the inter-organisational boundaries could be treated, at least from a process perspective, in a similar way to the treatment of traditional inter-departmental divides.

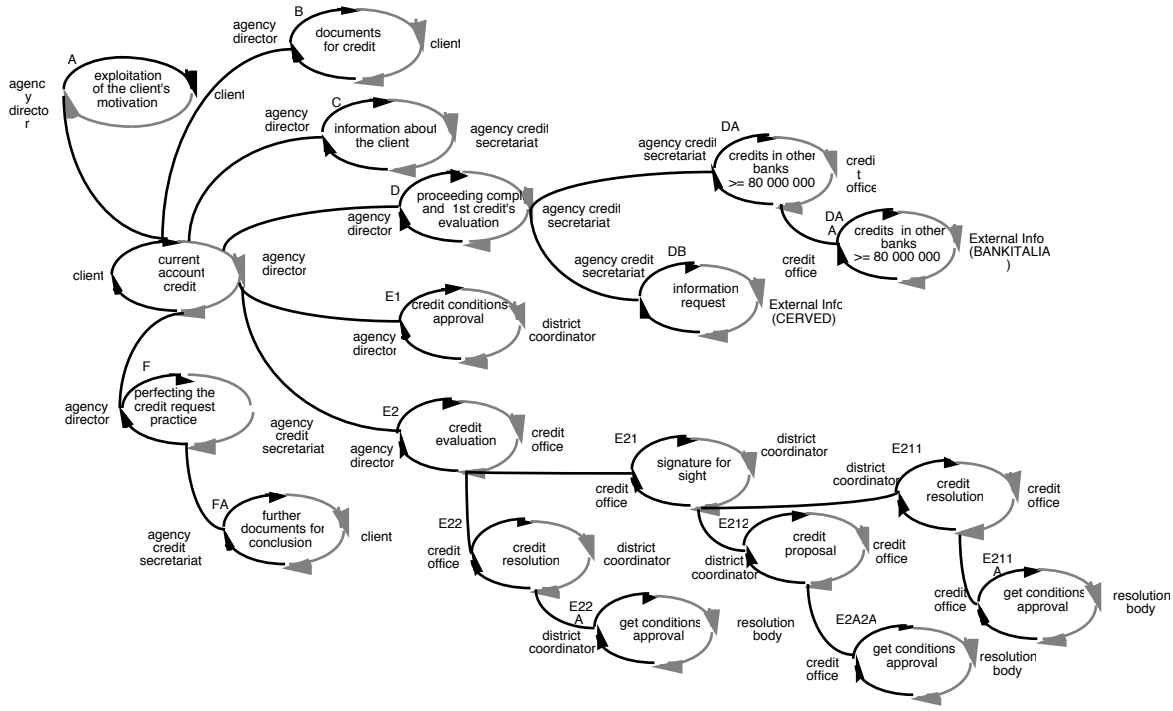

Figure 17 — Case study of Bank Processes[33]

In a development of the implementation-independent abstractions we saw in some Office Automation work, BPR suggests that computer systems, such as the shared data-base in the Ford case, should be viewed as part of an organisation and its processes. BPR permits roles and activities to be performed by a heterogeneous mix of human and non-human performers. In this way organisational design and the design of computer systems that exist in the organisational context are inseparable. As the notion of the 'system' takes on this wide sense then so does

33 From De Michelis and Grasso (1993).

that of 'design'. (In some ways this may move the designers' concerns from the 'effects' or 'consequences' of the introduction of technology and towards a more integrated conception of redesign in organisational contexts). The process model language becomes the basis of all design.

A process based rethinking of organisations and computer systems may result in the automation of processes but also the enabling of new working procedures. As well as requirements for further computer systems there may be a rethinking of existing interactions between separate computer applications and how to produce a coherent whole from the plethora of support tools located in pockets of IT.

The relational orientation of BPR has been linked to Total Quality Management (Bank, 1992) and a Human Relations perspective with end-to-end processes being delegated to and self-managed by workers with responsibility for the whole process, who act on information they generate, and work to customer objectives and outcomes. For example, the IBM loan scheme is redesigned so that case workers follow an application through from inception to completion. Individuals are said to be assigned goals and whole problems. For some this is thought to enable control to be built into the process and is contrasted with 'thinking departments' that control and monitor primary workers. As a result departments that collect and process information that has been generated by others come under particular scrutiny as potentially redundant.

However, whether or not these human relations issues are relevant, the central aims of reengineering are to improve processes by reducing 'costs', particularly transaction costs, of the process and the organisation achieving its goals. (see also Malone, 1987a; 1987b) on transactions costs and 'coordination structures'). We have seen process perspectives advocate the distilling of routineness or incremental standardisation and indeed within BPR there is often talk of the hidden rules or assumptions in the running of the organisation (policies, procedures, standards, responsibilities, authorisation mechanisms, delegation patterns, etc.) that may be revealed and made explicit in the analysis of process. For example, we have seen how with IBM's redesign story each loan application was no longer treated as unique and many were made routine. Factoring out prototypical processes allows the treatment of regular instances to be automatised or standardised. Costs are thought to be lowered by, for example, using expertise only when cases require it. (IT can play a further role by providing access to sources of expertise when and if it is needed).

Attempts to scale up the formally based process improvement techniques from projects and offices to organisations, and attempts to understand what metrics for 'good' business processes might be, are an on going concern of BPR as it seeks to base its evaluations, diagnostics and prescriptions on more than the informal observations of consultants. BPR also seeks to deploy some of the more formal representations utilised in Office Automation and Software Development Process Modelling.

Modelling notations

Earlier modelling techniques and notations associated with Process Modelling have also been applied to BPR. They are not reviewed again here but we have seen that they mainly focus on primitives such as roles, activities and interactions. While many of these methods recognise a need for multiple perspectives and try to involve a family of notations (including techniques such as entity-relationship representations, dependency diagrams, data and object flows, amongst others), in general, data objects are secondary subjects of the processes. For example, some in BPR see data entities as often being identifiable by examining the processes first, and it is taken that:

> "Much data that is seen around organisations either records the state of a process (because people have poor memories) or is just a way of implementing interactions.... an invoice is really only a physical means of making an interaction happen.... (Ould, 1992)

One of the major process modelling notations that has been adopted is Role-Activity Diagrams (RADs) in which roles exist independently of actors (people), and physical environmental constraints such as location; and are the primary structuring concept. It is considered an implementation detail as to whether activities are performed by people or machines. RADs are role-centric and begin by depicting the process from the view of an individual.

RAD is a diagrammatic notation used early on in the modelling phase and as with many methodologies the modelling tends to start with informal discussions, moving to natural language and diagrams and along paths of re-representation towards greater and greater formality. Users of RADs (e.g. Praxis Plc) follow this form of modelling with a formal phase that uses a textual language to describe the organisation's processes in a form that has well defined semantics and that allows consistency checking and animation (Ould, 1992). Similarly, the process modellers of ICL capture business objectives in English text. Analysis and design is then done Yourdon style, using Software through Pictures (StP), and using tool support StP structures are input via a bridging tool to a PML assistant (a graphical tool used to produce code in Process Management Language — PML). ICL have developed PROCESSWISE, a workbench of tools for "capturing, analysing and redesigning business processes" together with an integrator for "the generation of the necessary process support systems" and consultancy support services (IOPener, 5, 1992).

In this way, bridging the gap between process modelling and process support (White, 1992) is approached in different ways within the business process modelling literature (as it is in the general software engineering field). Some see direct routes in which the language of the process model is translated, perhaps automatically, into computer systems to support the modelled process. Others see the model as perhaps one of many ways of informing the design and designer about the context the system is being developed for and the requirements of it. This latter view of process models as 'briefs' seeks to influence design 'via the designer's head'.

More recently, process modelling has also been seen as a way of management thinking that is continuous rather than one-off. Some authors are developing links

between process modelling and Soft Systems Methodology (e.g. Kawalek, 1993). Consequently process change is seen as a major issue on the research agenda of process modelling. This includes the use of process models in the planned, deliberate and explicit 'migration' (rather than 'obliteration') of processes. It is suggested that

> "Process management is the way to approach the transformation of traditionally bureaucratic organisations into market-oriented companies" (Schael and Zeller, 1993)

Addressing process change has also included a recognition that innovation and flexibility can involve the deviation from formalised procedures. Both of these aspects of change have been considered in the CSCW literature and this is discussed below. Furthermore they both have implications for how process support technology is to be viewed — both as a means to rapidly reconfigure process support infrastructure — but also as potentially overconstraining of the necessary degrees of freedom for getting the work done.

Summary

Process modelling is a technique that is deployed for various purposes — an aid to office automation, to improving and supporting the software development process, a precursor to BPR and in the development of various IT systems (particularly workflow systems which we shall consider below).

The model itself is thought to be useful in understanding cooperative practices independent of whether it acts as a basis for requirements for computer systems or is somehow compiled into a process support system. It is one basis for analysing and recording/remembering in a re-usable way the process, the organisation, the way work is done, the division of labour, etc.. In this way organisational designs (or 'prototypes') are seen as being represented in an abstract explicit process modelling language as well as their 'implemented' form. They may be accumulated, explored in a 'what if' manner and perhaps form a basis for understanding the nature of organisational design itself.

The models may also be used in day to day activities to, for example, track progress through the organisational 'container', or to generate one type of awareness about organisational context in the form of processes happening, happened, or expected.

Furthermore, the exercise of modelling is itself often valued. It is interesting to note why this is thought to be so. The justifications and judgements about modelling experiences suggest that it is believed to promote a uniform common understanding, provide clarification and basis for reflection, and is thought to force the removal of ambiguities. This suggests that such understanding and consensus is not immediately encountered. Certainly the BPR approach places heavy demands on our abilities to consistently identify and isolate (amongst other things) distinct goals, tasks, tasks performed by technology, processes, customers and performers and turns in their cyclic relationship, costs, internal and external and intra- and inter-organisational boundaries, (stable) interfaces, points when goals can be said to have been achieved, atomic activities and their beginnings and endings. The

heavy demands in identifying the above may be contrasted with those, such as Shapiro (1993), who suggest that:

> "Work is not encountered as a set of discrete tasks involving chunks of data and distinct procedures, but rather as a free flowing gestalt contexture, with activities overlapping, and moving through foreground and background in response to (and constituting) the situation as it develops"

It may also be argued that organisational uncertainty over goals or customer vagueness about their requirements might not just simply require further effort to elicit this information. Rather, these problems may, in some cases, suggest that these units of analysis may not exist 'out there', 'ready made' and fully formed to be 'identified' by analysis (see also Bowers and Pycock 1993). The process of constructing a process model of an organisation, a product in itself, might in some ways be better seen as the debated and occasioned creation of one 'picture' of the organisation. On some occasions the exercise of process modelling may be seen as valuable to participants because it produces, rather than identifies, organisational goals, customers etc. and that it recruits members of the organisation to this common view of what the organisation is and does. Drawing boundaries, cutting categories, making explicit responsibilities, pointing to the 'ends' and purposes of an organisation and demarking the internal from the external are activities — political activities.

In the following section we shall consider approaches to modelling user's tasks adopted within Human-Computer Interaction (HCI) but it may be noted that the process modelling approach shares many similar orientations and has encountered similar problems with the identification of the granularity of atomic actions, user goals, etc. While process modelling differs from task modelling in that it is less concerned with single individuals, in the sense that process modelling assigns great importance to dependencies and interactions between people, nevertheless, it does employ a plan representation abstracted from the situated nature of work to describe the coordinated work of individual agents. In contrast, some writers have suggested that we should not engage in:

> "discovering the linkages which connect individual work together, but (is) rather what permits that 'individuation' in the first place" (Shapiro, 1993)

There may be further commonalities between task and process approaches. For example, as attempts have been made to use task models to inform the optimum design of the dialogue for user interfaces with minimum numbers of interaction steps, access to the functionality of the application provided at relevant times, coordination of display with the order of the task sequence etc., then so process models have aspired to inform the optimum organisation of interactions and the timely provision of information and support tools. Process models have also been used to evaluate existing working arrangements for complexity (see De Michelis and Grasso, 1993), redundancy, completeness, etc. in ways possibly analogous with the evaluative use of task models in considering the ease of use, ease of learning, cognitive complexity, etc. of interfaces. Furthermore, formal description techniques have been proposed to represent both user's tasks and process models

in order to be able to apply diagnostic reasoning techniques and as a bridging representation for describing (even programming) both models and support tools (see below and Johnson *et al,* 1993). Finally both task models and process models tend to make sequence/order/timing primary and data objects secondary, and consequently they both share the problem of relating such languages to object-oriented techniques where the primacy is reversed. (In task models, for example, the sequencing information that is to be related to the interface dialogue design is explicitly modelled, while in object-orientation it might be said to be distributed throughout the objects and to emerge from their relations).

Perhaps we can consider the relation between process models and dialogue models a little further. The BPR literature certainly suggests that business process modelling is most applicable in contexts where processes and cooperation amongst processes are particularly highlighted, and it is suggested that this is often the case with large bureaucratic organisations. It also suggests that the modern age values speed, service, etc. Consequently the concern of process models with the sequencing of interactions might be taken as a concern for organisational 'dialogue design'. Perhaps in this way BPR sees its usefulness as arising from the revealing of the hidden, implicit, emergent and distributed dialogue model embedded in the departmentally oriented organisation structure. When, in the case of IBM, BPR advocates "following one deal right through its course" it may be seeking to replace a concern for centralisation, specialisation, control of objects and tools, etc. with 'process awareness'. The notion of 'maturity' we saw suggested in the software development process modelling approach (Kellner, 1989), might be reflected in BPR as an organisation's self-awareness and explicit understanding or reasoning about its own internal dialogue.

While we have seen Business Process modellers agree that the organisation should organise itself around processes we have seen how in Software Development Process Modelling there can be differences in what 'around' means. An enactment view of business processes would almost view the recoordination of working practices as an unproblematic, controlled, even automatic and often top-down 'compiling' of a new design. For others the model is a simplification and resource that guides a continuously changing activity and whose embodiment in support tools should reflect this and the running repairs necessary to make use of it.

Such debates about the exceptional, contingent, temporarily ordered, situated and discretionary nature of much work and the role of mechanisms of interaction in coordinating activities (see Strand 3 Deliverable) are some areas that CSCW would see as central. Some process modellers suggest that the

> "loose association with CSCW will be important in redressing the authoritarianism implicit in process modelling" (Hall, 1992).

In some cases process modelling is also authoritarian with respect to its view of what organisations might be. On occasions it proposes that the errors of scientific rationalisation are errors of a focus that has outlasted the circumstances that had created the need for it. The way to achieve these "quantum leaps in performance" (Hammer, 1990) that it seeks is not thought to be to adopt an alternative view of the

very nature of organisations, but to reorientate the viewpoint towards inter-departmental processes. Often the process modelling still aspires to an explicit rational optimisation and reengineering of organisational life, retaining the primitives of tasks, roles, goals, etc. associated with this perspective. Distinctly alternative ontologies of organisation (see Strand 1 Deliverable) have viewed organisations as, for example, cultures or political systems. If process modelling sees its itself as *the* way to model an organisation rather than one of a multiplicity of ways, then this can tend to mean that all that is visible and supported in the organisation are the customer-performer relationships and that all activity must be rationally goal oriented. Interactions tend to be measured according to transaction costs rather than say social solidarity, bonding, etc.. Communication can become a flow of tokens. That alternative viewpoints might be useful, can perhaps be observed in, for example, the issue of organisational boundaries in that it is interesting to note that, while the process modelling perspective respects no inter-organisational boundaries, in practice 'other factors' seem to maintain them. Similarly with redesign its ramifications may be unrepresentable in the primitives of the process modelling view of organisations. Other perspectives might perhaps be usefully employed to avoid silencing these legitimate aspects of organisation.

As with all the above comments the process modelling community is not of one mind. Many authors pragmatically seek some involvement of other views of organisations (e.g.. Kawalek, 1993). Many authors (see for example, De Michelis and Grasso, 1993) also do not wish to exclude consideration of the advantages that departmental structures might have, for example, the sharing of common objects, critical masses of resources, informal communication and the availability of skills.

Finally the rhetoric of BPR is often grounded in a particular conception of the past, its legacy, and effects upon the present. We have heard how present organisational arrangements have their roots and rationales in the industrial revolution, that a working order can outlast and endure the occasions and circumstances that created a need for it, that we have let the inertia of convention sustain geriatric procedures and how we have neglected to review their relevance to the new competitive era. The 'fresh view' of BPR requires a stale past. However, while not excluding the possibilities for real improvements and changes aided by process modelling, it should be questioned whether the present is merely a fossilised sedimentation with forgotten rationales, caused by the past and resistant to change. Such a view hinders concern for the active, innovative, constructive and creative reproduction of order in the here and now. Attention to such possibilities may guide the sensitive design of future support and cooperation arrangements.

On occasions, care also needs to be taken to avoid over hasty dismissal of existing working practices by taking a more 'direct' cow path (as we saw that Ellis (1983) noted). Research in the CSCW field has shown a sensitivity to the possibilities that, amongst other things: redundancy of representation might provide fault tolerance (Hutchins, 1990); that layers of evolved practice can be a resource for understandability and the production of novel responses to unforeseen coordination problems (Hughes and King, 1992); that all forms of communication

are not necessarily direct and explicit (Heath and Luff, 1992); that repetitive actions can maintain 'knowability' and 'checkability' of systems (Shapiro, 1993); and that identifying what is and is not 'routine' can be the major task of work. Furthermore, some research in CSCW (see for example Luff *et al,* 1992; Hughes and King, 1992) has suggested that mediums of presentation can affect traditions of response and afford subtle differences in organising activity. Consequently, process modelling has to take care in abstracting from physical constraints and seeing as irrelevant whether, for example, invoices are in paper or electronic mediums.

Coordination Technology

As a final note on the perspective of cooperative processes and procedures we wish to briefly examine how this view has been developed as coordination, workflow or process support technology. Greater detail about such systems is contained in Deliverable 3.

Computers and their physical connections are seen as ubiquitous in the work place yet it is often only discrete tasks, such as word processing or spreadsheet calculations, that are supported in largely unconnected ways by pockets of IT. Process Modelling suggests that IT should reflect the context of the larger work processes rather than simply support individual steps, but in doing so it often calls for more than just open computing and data exchange. It calls for the ubiquitous computer to take a role in making the work flow around an organisation. It also suggests that the customer-performer relationship is often a primarily communicative one.

A Speech Act-based approach has been developed that focuses on the use of speech acts to represent action within organisations, the basic idea being that organisational action such as management and communication entails creating and taking care of commitments (Flores and Ludlow, 1980) or transactions and contracts (Lehtinen and Lyytinen, 1986; Auramäki *et al,* 1988). Speech act theory (Searle, 1969; Searle, 1979; Searle and Vanderveken, 1985) is applied to modelling this action of management of commitments.

It has aroused considerable interest in the IS community recently (e.g. Flores and Ludlow, 1980; Gooldkuhl and Lyytinen, 1982; Gooldkuhl and Lyytinen, 1984; Lehtinen and Lyytinen, 1986; Winograd and Flores, 1986; Auramäki *et al,* 1988; Flores *et al,* 1988; Dewitz and Lee, 1989; Dietz, 1991; Dobson *et al,* 1991, Donaldson, 1991; Kensing and Winograd, 1991; Dietz, 1992a; Dietz, 1992b). Since the theory of speech acts requires lengthy explication, this synopsis introduces only the main ideas and concepts. (In addition to the above original texts on speech act theory, we refer the reader to Lehtinen and Lyytinen, 1986; Winograd and Flores, 1986; Auramäki *et al,* 1988 and Dietz, 1991 for summaries).

According to speech act theory, speech acts are basic units of communication, taking place in a certain context of the speaker, hearer, time, place. The core aspect of a speech act is its illocutionary act, which has a propositional content and an illocutionary force. The propositional content refers to propositions expressed in an illocutionary act. The illocutionary force is the principal component of a speech act,

including among other things the illocutionary point. Searle suggests five categories of illocutionary points: *assertives* telling how the world is (e.g., to state or to predict), *commissives* committing the speaker to doing something (e.g. to promise or to agree), *directives* trying to get the hearer to do things (e.g., to order or to request), *declaratives* changing the world by saying so, and *expressives* expressing the speaker' feelings and attitudes. In the context of IS development, assertives and especially directives and commissives have been of most interest among these five categories. While the direction of fit in the case of assertives is from word-to-world aimed at getting the word to match the independently existing state of affairs of the world, the direction in the case of directives and commissives is world-to-word, i.e. the world is altered to match the word. In the former case adjectives such as "true" and "false" characterise how well the matching has taken place while in the latter case adjectives such as "fulfilled" and "unfulfilled" can be used to characterise how well the matching has taken place.

Speech acts form larger wholes, conversations or discourses (Auramäki *et al,* 1988) or networks of recurrent conversations (Winograd and Flores, 1986). They imply a specific set of alternatives which can be succeeded in a speech act pattern. For example, in the case of a question the alternatives are to answer, to request for clarification, to refuse, or to make a counter question.

Winograd and Flores summarise the application of this view to organisations by means of three assertions (Winograd and Flores, 1986):

1. Organisations exist as networks of directives and commissives. Directives include orders, requests, consultations, and offers; commissives include promises, acceptances, and rejections.
2. Breakdowns will inevitably occur, and organisations need to be prepared. In coping with breakdowns, further networks of directives and commissives are generated.
3. People in an organisation, including but not limited to managers, issue utterances, by speaking or writing. To develop the conversations required in the organisational network they participate in the creation and maintenance of a process of communication. At the core of this process is the performance of linguistic acts that bring forth different kinds of commitments.

Winograd and Flores also apply these ideas to the work of managers, interpreting managers as people who request and initiate actions that affect the work of others (Winograd and Flores, 1986) suggesting that instead of 'decision-making' we should talk about 'achieving resolutions'. They call the process of going from a situation of irresolution to resolution 'deliberation', which is a kind of conversation (in which one or many actors may participate).

The Speech Act approach has made considerable input to the representation of organisations and action within CSCW systems (though see Suchman 1993 for a recent criticism). However, rather than existing as a particular method, this form of analysis has been instantiated almost directly within systems. The most notable of

these systems include the workflow systems from Action Technology and the CHAOS system (see Deliverable 3). These systems are indicative of the representation of organisational work used within this class of systems.

The Action Technology system (the Action Workflow Management System (Medina-Mora *et al,* 1992)) uses the speech-act approach to communication as the basis for a model of organisational work as it attempts to support the workflow or coordination structure rather than traditional tasks. Computer supported coordination and workflow management systems aspire to be proactive in storing and forwarding forms and messages, making elementary decisions, bringing items to the attention of the appropriate people in a timely and reliable fashion, in accordance with pre-defined procedures, and pushing work around — something that is at present done largely by the actions of people.

With this in mind the Action Technology system attempts to assist in the tracking and monitoring of business processes, work done, sequence of workflows, and to automate repetitive and cumbersome tasks using a notion of requests and commitments expressed in a workflow loop (see figure 18).

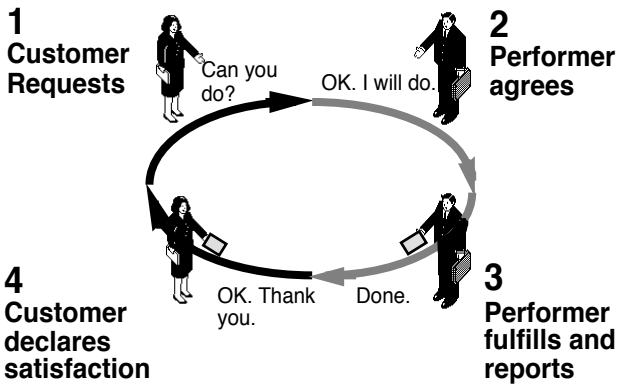

Figure 18 — The workflow loop

The workflow composed of people, procedures, information, tasks and management is defined by the language acts through which people coordinate, not by the tasks that individuals must deal with. Each workflow loop concerns a single task completed by a performer to satisfy a customer's request. Tasks can evolve in parallel or sequentially, as independent loops evolve simultaneously, while subordinate workflow loops must wait for the completion of subsidiary workflows. Each step can trigger the application of rules when one phase needs an input from another workflow. Time is expressed through deadlines, follow-ups and time for completion.

In accordance with the requirements for computer support suggested by a process model based analysis of work, the Action Workflow Management System (AWMS) aims to improve task performance by identifying, observing and anticipating potential breakdowns. Moreover, to keep processes moving along, AWMS manages reminders, alerts and follow-ups. The system allows the

definition of rules for routing documents, spreadsheets, forms, software programs, etc. among tasks. One goal of the AWMS is to provide workflow capabilities to new and existing computer applications.

In many ways these coordination technology systems embody much of the results from the process modelling research described above. Consequently there are those that have suggested that systems that coordinate users with each other and with data resources can be used to 'ensure that office procedures are correctly carried out'. While others, for example Finkelstein (1992) (in the context of the automation of the software development process), have expressed concern that a view of enforcing mandatory actions and of a computer driven orchestration of people as 'process inhabitants', can create a tendency towards developing "megalomaniac" computer systems.

Incorporating Users and Tasks

In addition to a consideration of the organisational setting and the nature of the cooperative work taking place, CSCW systems need to take account of the relationship between users and the supporting system. In particular, understanding the needs of users in terms of their everyday work becomes a central part of developing a comprehensive set of requirements.

Traditionally, the relationship between users and computer systems has been a central feature of HCI.

HCI as a research area involves more than just task-oriented approaches (for other aspects see as examples work on design rationale (MacLean *et al*, 1989), also work on user modelling (e.g.. Kelly and Colgan, 1992) and cognitive modelling, among many others). However, task based methods have explicitly addressed requirements capture. The development of these methods can be best described by a brief review of some major task modelling techniques.

Review of Some Task Models

In reviewing task models many HCI writers distinguish between those approaches concerned with the evaluation of existing interface designs and those concerned with the generation of design ideas or requirements. Related to this distinction is a view that users bring with them to the use of an interface: goals, task knowledge and skills. Furthermore, it is acknowledged that performing tasks with new tools can involve changes in what has to be done and what has to be known in order to complete them. In this way the performance of tasks is changed and previous knowledge of how to perform a task may be positively or negatively transferred to interacting with the new system. A central relationship in task based approaches to HCI is this mapping between users' tasks and task knowledge and systems functions and system-specific operations. Additionally, authors such as Carroll (1990) have considered how new technology may introduce both additional functions and suggest new requirements in an evolutionary task-artefact cycle.

There are, of course, many alternative ways of distinguishing task models from each other and a wide diversity of purposes for the models are available. Some are intended as design tools, some as research tools to understand HCI concepts, others to contribute to models of human information processing. Models may also be rated in terms of their formality, with some being based on grammars and formal description techniques, while others depend on looser structural forms. Task approaches have additionally been considered according to the aspects of psychological literature they draw upon and to what degree they have attempted to validate their representations in theories of cognition. For the purposes of this section the models are reviewed briefly to produce an understanding of how such models have developed in ways that may contribute to requirements capture.

Hierarchical Task Analysis

Hierarchical Task Analysis (HTA) (Carey *et al,* 1989) is well over twenty years old. It has classically been applied in other domains such as the development of programmed instruction, particularly for industrial process tasks, as well as in HCI. It uses two main interrelated forms of task representation, namely, simple hierarchical diagrams and flexible descriptions of plans. The diagrams represent overall task decomposition into component operations. The plans are presented in tabular form and contain information on sequencing and selecting subordinate operations within a task as, for example, prose or flow charts. The tables may also contain a wide variety of other information, such as explanatory notes, notes on the grain of analysis, ideas about interface requirements, task outputs and inputs, and so on. They describe the range of possible operations to generate transformations of state in the problem space and plans to combine them.

The approach claims some psychological validity by drawing on theories of learning, ‘chunking’ in skilled behaviour, aggregation of units of procedural knowledge and the hierarchical nature of skill and goals. Nevertheless it remains debatable as to whether or not the hierarchy assumptions underlying HTA are simply a convenient organisation artificially imposed upon streams of behaviour by the analyst.

It is interesting to note that, in a sense, models such as HTA structure the data collection and documentation of the analysis. The representation helps ensure a completeness in the analysis by giving an overall picture of task activities and suggesting a top-down approach of description. The hierarchical diagrams are also intended to be easy to understand by task incumbents during modelling or confirmation phases. They are said to aid communication between designers and to assist in systematising decision making about areas of design such as task allocation (discussed below). However, HTA does have problems with formally representing objects within the task domain (Carey *et al,* 1989)

Command Language Grammar

Command Language Grammar (CLG) (Moran, 1981) has been influential in HCI because it orientates to a methodology for the design of user interfaces. It proposes a top-down decompositional method with the intention of maintaining consistency

between levels in the evolution from high-level abstractions to low-level details. CLG provides a separation of levels of description of the interface. The model includes a conceptual component, separated from the particular form of the command language, and physical interaction details. The levels of description in the model are, firstly, a 'conceptual' component with task and semantic descriptions, secondly, a 'communication' component with syntactic and interaction levels of description, and, finally, the 'physical' component.

The physical component describes the spatial and device details, though this aspect of the model is perhaps less developed than the others. The communication component has a syntactic aspect describing the structure of the command language, its primitives and command contexts. The interaction aspects of this level include details of input devices, primitive system and user inputs and actions and some dialogue control. However, it is the conceptual component of CLG that is of most concern to us. This aspect of the model uses a hierarchy of goal oriented tasks; sub-tasks as mechanisms for achieving goals as a 'task level'; together with a 'semantic level' describing conceptual entities or data objects, permissible operations or actions, and methods or sequences of operations.

CLG suggests the evolution of design through these levels by advocating the description of each level in as much detail as possible without assuming the properties of the next level down. While the model helps structure these levels it does not necessarily provide for the mapping between them. The 'task level' analysis focuses only on those tasks carried out with the system (what users can do rather than what they might want to do) and describes them in terms of the computer system rather than in terms of the user and cognition.

Task Action Language

In some ways the Task Action Language (TAL) approach of Reisner (1981) can be located around the 'interaction' level of CLG. It uses a grammar linked to an action-language to provide early evaluation of an interface specification. Assessments of simplicity and consistency, linked to the rules of the grammar, are used to predict user behaviour such as errors and learning difficulties. The BNF grammar focuses on the lengths of terminal strings, numbers of rules, etc. as a way, it is claimed, of understanding what users have to learn and remember in operating an interface. Interfaces can be compared (for number of action steps, consistency of steps for related sub-tasks, etc.) if they are described in the same common grammar. As a specification based approach to the predictive modelling of user behaviour it requires that an interface design specification must exist before the TAL description can be constructed. The model is formal, and employs metrics of 'good' design (i.e. producing minimum numbers of TAL rules, lengths of terminal strings etc.). Some of the general approaches adopted by TAL were developed further in Task Action Grammars (Payne and Green, 1986).

GOMS

GOMS (Card, Moran and Newell, 1983) consists of Goals (that provide a symbolic structure defining states to be achieved), Operators (elementary

perceptual, motor or cognitive acts that change states and whose sequence defines behaviour), Methods (learned and memorised procedures for achieving goals), and Selection Rules (for choosing between competing methods of achieving goals). GOMS also proposes a 'Model Human Processor' (MPH) as a psychologically informed theory of human information processing with three interacting subsystems (perceptual, motor and cognitive). Together these two aspects of the approach — GOMS and MPH — are used to make predictions about time taken in operating an interface. While the approach has been criticised for it limitations to the analysis of skill performance using an additive model of serial behaviours, for our purposes its main limitation is that it again attempts to provide input to interface evaluation rather than to the initial generation of requirements.

Cognitive Complexity Theory

Cognitive Complexity Theory (CCT) (Kieras and Polson, 1985) attempts to predict the complexity of learning using an interactive computer system from the user's perspective. It suggests a comparison and mapping between the knowledge required to use a system and existing knowledge which the user possesses, thereby predicting what learning is necessary in order to use the system. Based on production rules and generalised transition networks, it constitutes a task based approach to user interface design evaluation but is not a design generation tool.

Task Knowledge Structures

The Task Knowledge Structures (TKS) approach to task analysis (Johnson, 1992; Johnson and Johnson, 1991a&b) makes quite strong claims to psychological validity based on the representation of the conceptual knowledge about tasks, stored in users' memories and recruited or required by a user in performing the task. TKS theory suggests a number of types of knowledge associated with tasks, including: a Goal Structure, which aims to model the goals and sub-goal relations; a Procedure Set, which models the behavioural mechanisms through which goals are realised, together with control structures for the actions; and finally, a Taxonomic Structure, that aims to model knowledge about objects and actions, their properties, categories and composition.

TKS offers itself as a cognitive model of users' knowledge about their tasks while not going into detail about how that knowledge is processed. There is a task analysis method associated with TKS called Knowledge Analysis of Tasks (KAT) (Johnson, 1990). This method suggests a number of analysis techniques (such as card sorting) and phases, such as 'generification' for reconciling different viewpoints and identifying common task elements.

TKS is advocated as a way to model the knowledge a user brings to their task and is thought to offer the potential to feed into design idea generation and system requirements. TKS is also said to be usable to represent the knowledge required by the user when performing a task in conjunction with a new interface design. This 'resultant task model' (Wilson *et al,* 1992) is said to offer the possibility of evaluating the interface design with respect to the extant task knowledge brought to the task by the user. The TKS approach suggests that it should be possible to

compare the extant and resultant task models for matches and mismatches and to interpret these according to theories about the positive and negative transfer of knowledge in task performance.

The TKS model also contains notions of the frequency and centrality of objects and actions, and ideas of superordinate and subordinate categories of objects and actions, however these ideas are at present perhaps less developed in terms of their contributions to requirements generation (though see Johnson and Nicolsi, 1990).

From Task Analysis to Requirements

From the brief review of major task modelling approaches given above it can be seen that a large number of the approaches have been oriented towards interface evaluations. However, recently effort has been put into developing the role of task analysis as input into requirements generation. Task analysis has been said to contribute to requirements and designs at different levels and phases in some of the following ways (Carey *et al*, 1989):

Scoping

At an early stage of design task analysis is seen as a way of understanding the basics of the users' current tasks and perhaps potential future ones, hence providing an overall scoping of the task and system functionality.

Task allocation

Task analysis is seen as a basis and structure for understanding what functions have to be allocated to humans, machines and to joint performance, and hence feeding into functionality requirements for the system. Task allocation is suggested to be informed by general notions about human and machine abilities.

Conceptual design

Task descriptions are thought to provide information that can be related to, or mapped onto, the design of the interface at successive levels of refinement (as CLG suggests). As this aspect of the contribution of task analysis to requirements is of concern to us, the resources for design provided by a task model are described in greater detail below.

Evaluation

A completed interface design or design specification may be a basis for evaluation from the perspective of users' tasks. Some task analysis methods focus on evaluating an interface for consistency and usability by producing a task model for its use and evaluating it 'within itself' with respect to metrics such as the number of rules and rule exceptions necessary to know in order to operate the system. Other models consider the compatibility of an interface design and the extant user's task knowledge and goals.

Task analysis can also be a means of ensuring that a representative range of tasks are evaluated in, for example, experimental evaluations.

Documentation, Training, Support in Use

A task model is also thought to be of use in designing tutorial and documentation materials, in understanding some of the reasons and intentions behind design features during the maintenance and modifications of systems in use. Finally the completion of a task analysis is said to be useful for traceability in 'reverse engineering' exercises.

Resources from Task Analysis for Design

Considering in further detail some of the above, task models can be seen as a structured collection of information about the user's work. These models may provide resources for the design of the user interface and the design of the interactions between human and computer. As well as providing some input to understanding functionality requirements, recent research has considered the role of task models in designing conceptual elements of the interface (Johnson, 1992).

Input/output

Understanding the distribution of tasks between users and computers and the communication requirements between them is further developed by identifying some of the requirements for user input to the computer, and computer output to the user. This analysis is also thought to be able to provide some further detail as to whether the input/output is numerical, locational, alphabetical etc. Perhaps even further detail can be added about, for example, whether some choices from a set of options are mutually exclusive or not. These levels of description offer possibilities of relating details of task models to high-level abstract descriptions of standard interface interaction objects such as radio buttons, tickboxes, menus, sliders etc. (the possibilities for such mapping is considered later). Further information, such as the size of the set of options or the frequency of performing a particular interface action, can provide further resources for the design of displays and interfaces.

Such direct mapping between task actions and objects to interface interaction objects tend to relate to standard libraries of interaction objects rather than domain specific (perhaps directly manipulable and interactive) interface objects.

Task objects

In addition to relations between actions and standard interface interaction objects, task models may also be sources of information about domain/task specific objects and their behaviours. Information in task models about task objects (their class structure, attributes etc.) might be valuable in designing the conceptual structure of the system and its interface. Providing interface objects and representations of objects in the task model is seen as allowing the 'transfer' of knowledge and aiding transparency of computer operation.

Task Actions

Similarly, domain/task specific actions may be described in a task model and available to inform elements of design such as the naming of options and the ordering and structuring of menus.

Dialogue Design

Finally, much recent work in relating information from task models to interface designs has focused on the dialogue design of interfaces and the sequencing information within task models. Tasks can be performed, among other ways, in parallel, sequentially, in set combinations of valid orders, etc., and this sequencing information is commonly explicitly represented in task models. Interface designs have a dialogue design in that they can allow access to objects, functionality, 'screens', etc. in particular orders. Consequently the available routes through interface states can be partially related to the paths of task performance modelled in the task model. Many of these approaches have sought to match the sequence of the task performance modelled in the task model (user's view) with the dialogue design in the interface (designer's view) to optimise the number of iteration steps, relevant provision of information, and access to functionality.

As a discipline drawing upon Cognitive Science some criticisms of HCI have drawn upon criticisms of Cognitive Science itself such as Coulter (1979, 1983, 1989) and Suchman (1987). Many of these criticisms are developed within CSCW and its contributing disciplines such as Ethnomethology. In some ways some of these arguments against task modelling appear to hold more for the approach in the abstract than for its application in task based design where design in practice is a pragmatic activity using whatever resources it can and in often irreverent ways. However, what has perhaps been of most concern to those advocating task oriented approaches to design and requirements has been the relative lack of uptake and impact upon the requirements process. As Shapiro (1993) has noted, the newly developing field of CSCW currently enjoys a certain 'suspension of disbelief' but a time will come when a "more critical eye is turned on us". Consequently there may be valuable lessons in the experiences of HCI and task analysis in attempting to inform requirements and computer design.

Integrating Human Factors and Development Methods

Task based approaches within HCI have been strongly contrasted with rapid prototyping development techniques. Advocates of rapid prototyping have seen task based methods as too time consuming, too abstract for users to understand, only loosely mapped to actual designing, and as generating replications of existing practices rather than innovations. On the other side task analysis advocates have portrayed rapid prototyping as trial and error in which an optimum design is thought to be converged upon by iterative evaluation and modification. They bemoan the lack of representation 'beyond the current implementation' reducing the possibilities, they say, for developing cumulative design theory. They see rapid prototyping tools as encouraging 'hacking' not understanding.

Task modellers have employed various strategies to defend their approach. They have argued that some consideration of users and tasks is inevitable and that they offer a more explicit structured method for undertaking such consideration. They have pointed to high costs and risks 'downstream' from incorrect user requirements. They have developed tools to support the process of task analysis,

and the creation and management of task models. Graphical versions of representations have been developed in an attempt to improve comprehensibility. Some, for example Hill *et al,* (1993), have also begun to recognise the traditionally individualistic focus of task analysis and have attempted to extend the task perspective by considering multi-user, multitask and distributed cognition (Hutchins, 1990).

Another central development in the defence of task based approaches has been attempts to integrate task analysis methods with established software design methodologies such as structured analysis. While this integration with established methods is partly a political attempt to promote a place for task methods and to obtain resources for this activity, nevertheless it has involved serious consideration of the relationship between representations used in task analysis and those used elsewhere in the design process. It has also involved attempts to understand more fully how and where the results of various forms of task modelling and human factors analyses can feed into structured design (Lim and Long, 1992). In a related manner those such as Johnson *et al* (1993) have attempted to relate the modelled information in task models with abstract descriptions of user interfaces, and to embody these mappings in tools for user interface design. In one sense this approach uses a task level description technique as a form of programming language for prototyping interfaces.

Some, for example Lim and Long (1992), have considered how the gap between task oriented approaches and rapid prototyping methods might be bridged. They have suggested that a task analysis may act as a means of producing a first 'best' prototype that can be iteratively developed. They have attempted to address what is perceived as the 'bolting on' of human factors too late into design where it can consequently contribute too little. A further implication of their approach is that prototyping, in support of analysis, can occur at the level of task models as well as concrete interfaces.

The remainder of this section considers two separate endeavours, each of which has attempted to involve some representation of human activity and task within structured development methods. These methods represent an initial attempt to provide some bridge between the analysis of tasks and the structured development of software systems. Many of the objectives of representing, recording and reflecting the needs and desires of users within the systems development process are shared by this strand of the COMIC project. However, the starting point for all of these methods is predominantly based on a cognitive understanding of the nature of human tasks rather than a social consideration of the context of work.

User Skills and Task Match

USTM (Macaulay *et al,* 1990; 1993) is aimed at the earliest stages of requirements analysis for generic systems. The motivation for this particular focus is the authors' recognition that the vast proportion of failures of information systems can be traced back to errors and omissions during the requirements phase of the system development process. This is coupled with and compounded by the fact that the

earlier a problem is recognised and dealt with, the cheaper it is to rectify. Macaulay *et al* take a user-centred and socio-technical perspective on system design in order to address these issues by concentrating on the users, the tasks they perform, and the other people and objects that they interact with during their work.

USTM consists of three stages, each of which is characterised by a two to three day workshop, with associated information gathering and consolidation activities taking place before and afterwards. The stages are as follows:

1. Describing a Product Opportunity (DPO)
2. Identifying a High Value Solution (HVS)
3. Delivering the Business Solution (DBS)

Each workshop is attended by a number of stakeholders in the system under investigation, and a facilitator. The stakeholders will include: system developers, technical authors, marketeers, user representatives, a project manager, and user managers. Obtaining the right set of stakeholders for the team is important to the success of the method. The membership of the team should provide knowledge of the type of users and workplaces that the system is to be introduced into, of similar systems developed previously and so on, such that relevant viewpoints on both user-centred and technical issues are represented. The stakeholders must also be of sufficient status such that any decisions made by the team are more likely to be acted upon, rather than requiring the approval of someone who was not present, and therefore not necessarily sharing the same sense of ownership of the proposal or solution. The facilitator is both a group coordinator in the usual sense of the word (see Viller, 1993), but is also an expert on the method. This individual is therefore central to the use and success of the method.

The starting point for the first stage, and for the process as a whole, is a market segmentation statement. This provides information about the marketplace that the proposed system or product is intended to be launched into. It will be a result of market research, and may include information about competing products, typical users, and degrees of credibility attached or required for the various items. During DPO, the requirements team

> "...explore and describe the critical characteristics and potential for change of the users and their environment in terms of work-groups, objects, tasks and interactions." (Macaulay *et al*, 1990)

This process is intended in particular to enable the team to assess the feasibility of the proposed system or product, and the output from DPO is called a *Human Factors Description*, which records the validity and credibility of the information gathered. By 1990, DPO had been used on approximately 30 occasions as input to the feasibility review for ICL products (Macaulay *et al*, 1990), and the method is currently used within ICL under the name 'Marketing to Design' (MTD).

In HVS, the team concentrate on product requirements that they see to be of high value to them as individual stakeholders. The concept of the user role and product role pair is introduced at this stage, during which the team is moved from a process of description towards one of analysis. The end of HVS results in a set of key user roles, along with usability goals and related metrics. The various stakeholders are

expected to verify this information outside the workshop, through consultation with human factors specialists, end-users, and prototyping where appropriate.

Finally, in DBS, the partially complete set of product requirements are returned to in the light of the verification process following HVS, and the team

> "...focuses on the requirements for supporting the product in the user's environment." (Macaulay *et al*, 1990)

Here, the team is involved in a decision making process, and at the end of the stage they will have produced a specification of the non-functional requirements for the system. Once again, the information must be subsequently verified with end-users etc..

The developers of USTM see the method as being complementary to other, more formal, methods that in many cases assume that an initial specification exists that requires analysis. In such cases, USTM can be used as the first part of the requirements 'capture' process which actually delivers this initial specification. The strength of USTM lies in the building of a team that includes stakeholders, and therefore expertise, from various viewpoints onto the system or product under development, with the aim of developing a shared understanding of the system and the problems being addressed. For example, including technical authors and end-user trainers at the earliest stages of product development alleviates to a great degree the potential problem of users receiving multiple and differing views of the system that they are to use: from the training they receive, from the user manuals, and from the interface to the system itself (Macaulay, 1993).

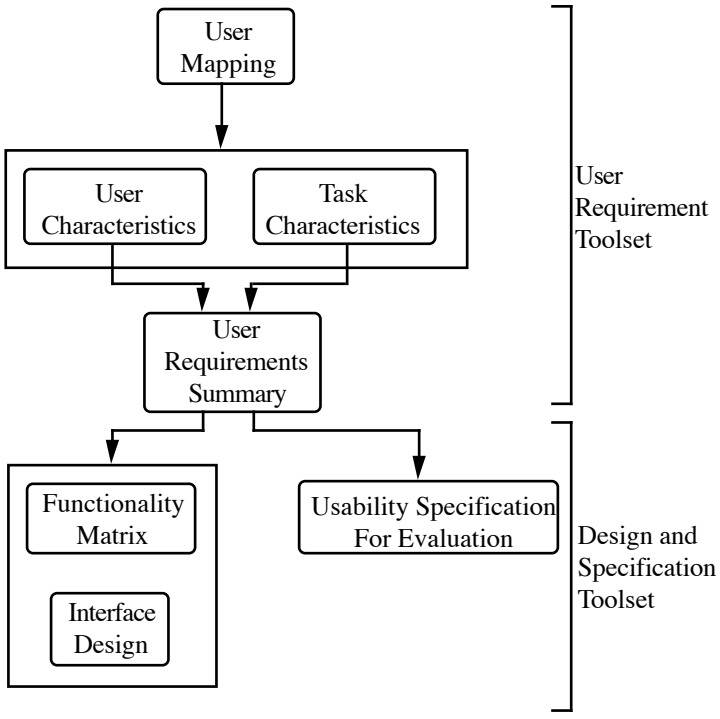

Figure 19 — Tools in the HUFIT toolset.

HUFIT

HUFIT (HUman Factors in Information Technology) (Taylor, 1990; Catterall, 1990) was an ESPRIT I project aimed at addressing the problem of providing IT products which cater for the specific needs and characteristics of end-users. The project resulted in the production of the HUFIT toolset which aims to provide a user-centred method of system design. The toolset is split into two parts: the user requirements toolset, and the design and specification toolset. They each in turn consist of a number of tools, which cover the early stages of product conception through to specification and design. The tools and their relationships to each other are depicted in figure 19.

HUFIT also has a workshop format in which various system stakeholders participate. The tools are paper based, and are intended to be easy enough to use such that developers with little or no human factors expertise can use them effectively. A further training device is used with the method, called Quick Ergonomic Design (QED), which provides an introduction to human factors issues and to the rest of the HUFIT toolset.

Concluding remarks

Similar remarks can be made for HUFIT as for USTM in terms of the emphasis on team work and achieving a shared understanding during face-to-face sessions. Both methods share a user-centred perspective, and address the earliest stages in system development; an area they believe to be neglected by other existing methods. Contact with end-users varies from one-to-one contact in interviews or prototyping sessions, through to the representative views as to what a typical user might be, as provided by marketeers.

Development Oriented Techniques

In this section, we use a traditional requirements example, that of a simple automated teller machine (ATM), to discuss each of the approaches. The ATM contains an embedded software system to drive the machine hardware and to communicate with the bank's external customer database. The system accepts customer requests and produces cash, account information, provides for limited message passing and funds transfer. The ATM is also required to make provisions for the major classes of customers, the home customer and the foreign customer. Apart from providing services to its users the ATM is also required to update the customer account database each time there is cash withdrawal or funds transfer.

It is interesting to observe the suitability of this example within the literature given its neutrality in considering the interactive setting within which the ATM is placed.

Functional Approaches

This section considers a number of methods for the analysis of requirements which exploit a functional approach to the representation of requirements. These methods

are closely associated with structured analysis, typically adopting a data flow perspective, and have been applied within that particular approach to systems understanding and development. The intent of many of these approaches is to construct models of the system in order to drive the development process. Data-flow approaches take as their primary viewpoint the way in which the system transforms inputs into outputs. Many functional and process-oriented approaches are typified by the data-flow model.

The data flow approach is based on the notion that systems can be modelled as a visualisation of the data interaction that the system, or some lower part of it, has with other activities, whether internal or external to the system. Figure 20 shows the notation for a data flow diagram (DFD). A DFD is composed of data on the move, shown as a named arrow; transformations of data into other data, shown as named bubbles; sources and destinations of data, shown as rectangles called terminators; and data in static storage (i.e., data stores), shown as two parallel lines. There is little uniformity in industry concerning DFD notation. The notation shown in figure 20 was advanced by DeMarco (1979). Gane and Sarson (1979) use rounded rectangles for bubbles, shadowed rectangles for sources and destinations, and squared off Cs for data stores. Orr (1981) uses rectangles for bubbles, ellipses for sources and destinations, and ellipses for data stores.

The data-flow approach is typified by the Structured Analysis method. The structured analysis method has undergone several refinements since its introduction, but its underlying principles have remained the same. Two major strategies dominate. The ‘old’ method popularised by DeMarco and Gane and the ‘modern’ approach of McMenamin and Palmer (1984) and Yourdon (1989). The next section reviews the DeMarco and Yourdon approaches, and three others

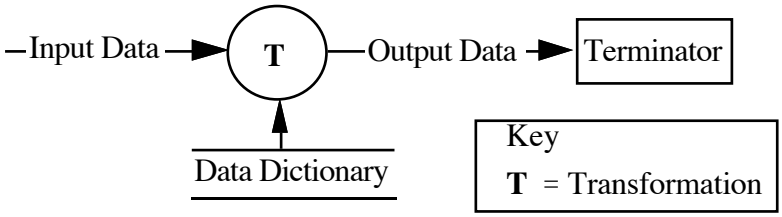

Figure 20 — The data-flow notation

DeMarco

The DeMarco approach advocates a top-down approach in which the analyst maps the current physical system onto the current logical data flow model. The logical data model is used to derive the proposed logical model by adding new logical requirements and modifications. The proposed system forms the basis for implementing the new physical system.

On explosion, DeMarco strongly recommends that no DFD should have more than seven processes on it, and that complex processes should be exploded until

each primitive process on the lowest-level DFD can be documented by a page or less of *Structured English,* or other form of logic.

The DeMarco approach can be summarised in four steps:

1. Analysis of current physical system
2. Derivation of logical model
3. Derivation of proposed logical model
4. Implementation of new physical system.

The objective of the first step is to establish the details of the current physical system, then to abstract from those details the logical data flow model. The derivation of the proposed logical model involves modifying the logical model to incorporate logical requirements. The implementation of the proposed model can be considered as arriving at a new physical system design. Figure 21 shows the top-level logical DFD for the ATM.

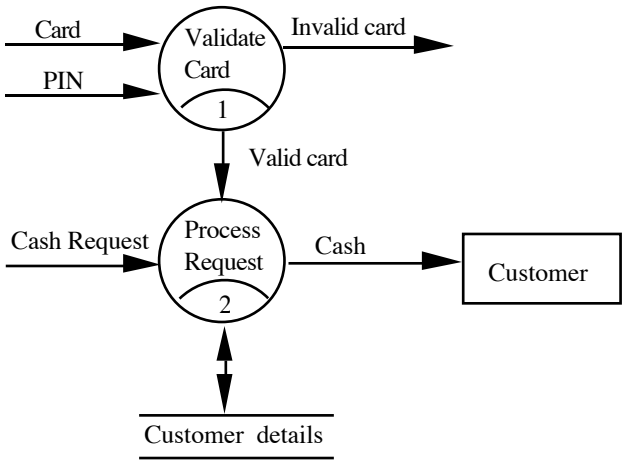

Figure 21 — The DeMarco top level logical DFD for the ATM

Gane and Sarson

Gane and Sarson (1979) proposed a different graphical notation for the data-flow diagrams and a slightly different approach to structured analysis from DeMarco. The Gane & Sarson notation replaced the circle representing process with a round edged rectangle, and the two parallel lines representing data store with an open ended rectangle. The notation also replaced the plain square representing the entity with a solid square.

The Gane & Sarson approach places more emphasis on the identification of data components of the system than does the DeMarco approach. They suggest augmenting the data flow notation with data access diagrams to describe the structure and contents of data stores in the system. Each data store is viewed as collecting several entity types, each of which is related in some way to one or more of the other entities in the data store. The data access diagram is a graphical representation of the different entities in a data store and their relationships.

The Gane & Sarson approach also takes a different approach to deriving the proposed logical model. They take the position that a decision has to be made on each project whether it is faster to get to the proposed logical model from the current logical system, or whether in the light of the user requirements it is better to model the proposed system from the beginning. The steps for the Gane & Sarson approach may summarised thus:

1. Determine the shortest way of getting to proposed logical model
   Way A. Derive from current logical model
   Way B. Model proposed system from start
2. Proposed logical model.

Figure 22 shows the equivalent Gane & Sarson DFD for the ATM.

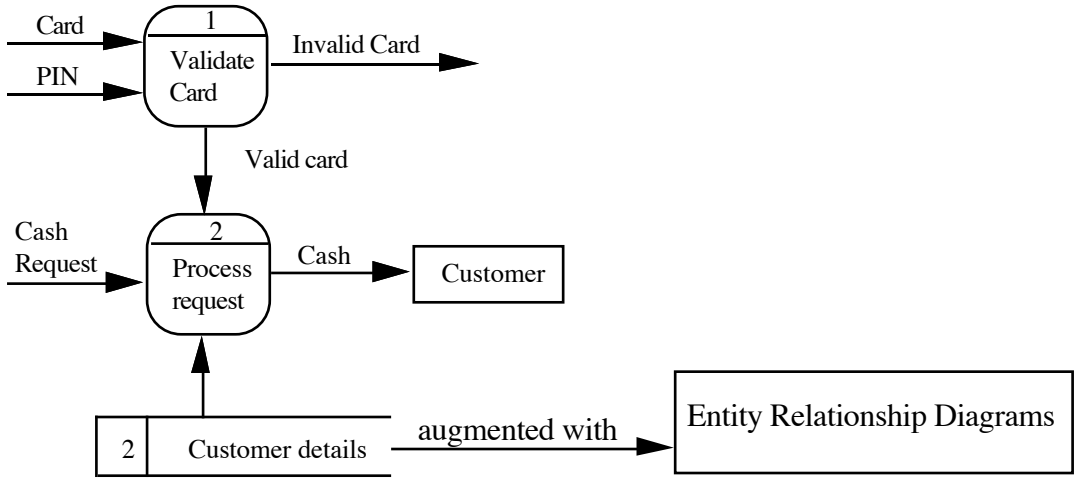

Figure 22 — The Gane & Sarson DFD for the ATM.

SADT

The Structured Analysis and Design Technique (SADT) was developed in the early 1970s by Ross at Soft-Tech (Ross 1977) and is based on a data-flow model that views the system as a set of interacting activities. A rectangular box representing the system's most abstract activity together with a set of inputs and outputs constitutes the starting point for functional decomposition. Consistency checking involves matching the set of inputs and outputs in each successive functional decomposition against the higher function.

The SADT notation is based on natural language augmented with a non-ambiguous set of graphical notations (figure 23). The notation consists of a rectangle representing some system activity and a set of four arrows, each with a different semantic. The arrow coming in at the left of the rectangle represents data input, the arrow leaving at the right represents data output and the arrow coming in at the top represents control information or data to facilitate the process. SADT integrates system analysis and design and incorporates this in its notation, the

arrow coming in at the bottom of the box represents the mechanism or algorithm by which the process is to take place.

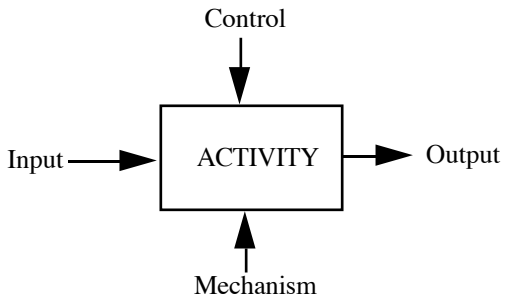

Figure 23 — SADT Notation

Several extensions have recently been made to SADT to facilitate interfacing with other methodologies. Perhaps the most notable of these is with a program design language (PDL) to facilitate the simulation model. The ATM example for this method is given below in figure 24.

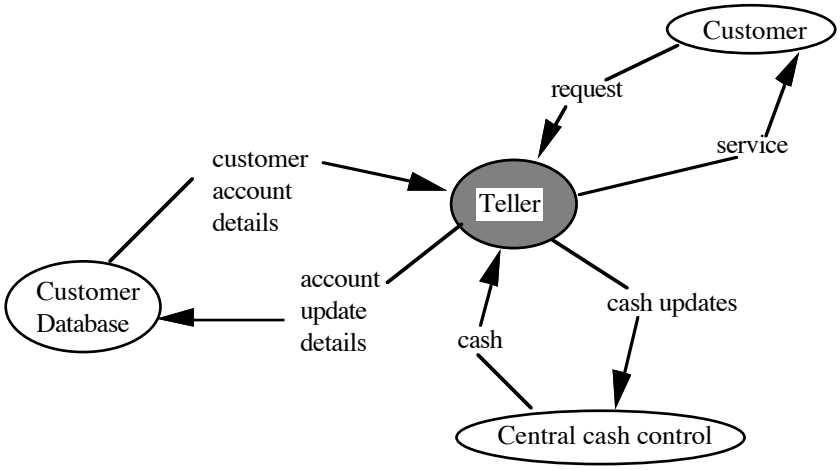

Figure 24 — User-level data flow diagram

Modern Structured Analysis

The “modern approach” first appeared in the literature as the work of McMenamin and Palmer (1984). MacMenamin expanded on DeMarco’s work and made considerable progress in elucidating the difference between users’ real needs (called ‘essential requirements’) and those requirements that represent the external behaviour of one particular incarnation satisfying those needs. Structured analysis has further been modernised by Yourdon (1989) to integrate the several ideas advanced in earlier approaches and to incorporate CASE tools.

Up and until the mid-1980s structured analysis had focused on information systems applications and did not provide an adequate notation to address the control and behavioural aspects of real-time system engineering problems. This deficiency

became painfully obvious when attempts were made to apply the method to control-oriented applications. Ward and Mellor (1985), and later Hatley and Pirbhai (1987) introduced real-time extensions into structured analysis. These extensions resulted in a more robust analysis method that could be applied effectively to engineering problems. Attempts to develop one consistent notation have been suggested by Bruyn *et al* (1988).

Real-Time extensions to structured analysis

Ward and Mellor (1985) extended the basic structured notation to incorporate control flows and control processes that could be defined as state machines. The Ward control notation combines control and data associated with each activity on the same data flow diagram. The control is differentiated from the data flow by a dotted arrow. The Ward notation also allows for control transformations, event flows, and buffers. The notation differentiates between discrete and continuous data, and three types of event flows: signal, activation, and deactivation events. The Ward notation supports information modelling and provides for entity behaviour to be modelled by using state transition diagrams.

In 1987, Hatley and Pirbhai published a slightly different set of real-time extensions to the basic data flow notation. The constructs are much the same as those suggested by Ward and Mellor but there are major differences in the tools suggested by Hatley and Pirbhai. They propose separate control flow diagrams (CFDs) and data flow diagrams (DFDs), and separate control and data flow specifications. CFDs and DFDs are addressed at the same level of abstraction, hence a CFD is usually associated with a corresponding DFD. CFDs are modelled using Mealy or Moore state machines.

Yourdon

The structured analysis notation was further modernised by Yourdon in his book *Modern Structured Analysis* (1989). Yourdon's modern structured analysis is a compilation of the experience collected during two decades of structured analysis in practice. Yourdon defines system analysis to include the development of two models: the environmental model and the behavioural model, which together are called the *essential model*. All of the models are composed of data flow diagrams and process specifications, which should be thoroughly analysed for consistency.

The environment model comprises three parts: a rationale for the system being modelled, the context diagram which shows the objects that interface and interact with the system externally, and a list of all events that precipitate a system response. The behaviour model comprises five parts: a complete set of levelled DFDs incorporating the Ward & Mellor real-time notational extensions, entity-relationship model, state-transition model for all control processes, data dictionary, and process specifications.

The Yourdon approach proposes minimising or even skipping altogether the model of the current physical system and is an interesting departure from the DeMarco and Gane & Sarson approaches. Yourdon gives his reasons as political: stating that experience has shown that time and effort expended in deriving the

model can frustrate the user to the point of cancelling the project. However, he acknowledges that for some systems the analyst must build a model of the users current system, and cites the example of when the analyst needs to model the current physical system in order to discover what the essential processes really are.

Other Structured Analysis Approaches

Other structured analysis approaches include Structured Requirements Definition (SRD) (Orr, 1981) and the Structured Systems Analysis and Design Methodology (SSADM) (Skidmore *et al,* 1992).

Ken Orr, the author of SRD, calls the methodology "defining the application context". The method comprises three main steps. The first of these is to define user-level entity diagrams. A user level entity diagram is a record of the inputs and outputs for each individual within the organisation under analysis. The next step is concerned with defining a combined user-level entity diagram. The objective of this step is to bring out the inconsistencies that may result from individual perspectives and to resolve them. The final step defines the application-level entity diagram; this is achieved by drawing a dotted line around that part of the diagram that corresponds to the "organisation" or "system" under analysis.

The Structured Systems Analysis and Design Methodology (SSADM) was originally developed for UK Government development projects by Learmonth and Burchett Management Systems and the Government's computing advisory body the CCTA. SSADM is a data flow method that covers only analysis and design; the early system requirements phase falls out of its scope. The SSADM methodology can be summarised thus:

1. Analysis of current system
2. Specification of requirements
3. User options
4. Data and process design
5. Physical design

Many variations of functional methods to support structured analysis have evolved over the years starting with the DeMarco approach in the early 1970s to the 'modern' approach proposed by Yourdon. The early versions focused on information systems and did not provide adequate notation to address control and behavioural aspects of real-time systems. This deficiency was redressed by the real-time notational extensions to the basic data flow notation, proposed by initially by Ward and Mellor, and later by Hatley and Pirbhai. However, several problems still persist in approaches based on the data flow model:

- One of the major criticisms levelled at the data flow approach is its strong functional emphasis and therefore weak change-resilience; changes in data tend to ripple through the entire structure.
- Because the primary structuring vehicle in structured analysis is a DFD (which lacks the means to focus on the persistent aspect of data) its emphasis on data structure has remained weak. Although later versions of structured

analysis have been augmented with an information model, the emphasis on transforming inputs to outputs has diminished its impact.

- Data flow approaches lack detail when modelling parts of the systems that primarily update or retrieve data.
- The data flow model has to undergo significant reorganisation in order to map onto design due to the shift in the underlying representation. Data flows are a network of bubbles and data stores; design-oriented structure charts are a hierarchical representation of modules.

The modelling technique based on Data Flow means that the focus is much more on how things are done rather than why, that is, work and data flow (Bengtson *et al,* 1985). Decision processes, knowledge, semantics of the processes and flows are out of scope. It is implicitly defined that the semantics can be expressed via the data and work flow.

The methodology presupposes that:

1) all data flow is known in advance and can be described exhaustively and unambiguously.
2) the relevant concepts of the field of work are stable and can be described exhaustively and unambiguously.
3) the field of work can be described exhaustively and unambiguously as a two dimensional system of data flows (Floyd, 1984).

The methodology focuses only on the aspects of the field of work which can be modelled in a procedural manner in order to implement it in a computer. As Yourdon (1989) himself phrases it:

> "the systems analysts has to provide sufficiently detailed information for the systems designer to concoct a technologically superior design"

It is a very simplified conceptual world that is modelled. The methodology cannot express e.g. abstractions or specialisation's in the conceptual models (Floyd, 1984). There are no possibilities for separating procedures (expressing the contextual requirement to the process) from routines (expressing existing practical experience concerning effectiveness and efficiency) or algorithms (expressing the "correct" logical method). Further, there are no techniques for modelling means/ends relations, causal relations, control structures etc. These are only implicitly represented in the models as 'hidden' aspects of the data flow.

In Bengtson *et al* (1985) an actual case of the use of Structured Analysis in a bank, is analysed and concludes:[34]

> "SA and SS didn't support understanding work situation with much variation from case to case.
>
> [...] DeMarco's methodology is not useful for describing the richness in the non-trivial parts of office work.
>
> [...] Things to remember is specified in minispecs. Things to look up is modelled as a file. This is a forced selection between extremes.

---

34 The translation is by Kjeld Schmidt and Peter Carstesen. See COMIC-RISØ-2-1

> [...] It is difficult to separate data flow and control flow.
>
> [...] There were problems in modelling how the work was organised. This technique was impossible to use because the work contained several integrated work processes performed by the same actor.
>
> [...] The mechanisms for decomposition is very strong but all in all the concepts in the methodology very poorly support analysis and description of the non-trivial parts of office work. The tools will probably be very effective for describing purely procedural decision situations, whereas context dependent interpretations and decisions cannot be modelled. Neither the decision process itself nor the exchange of information among the actors could be modelled."

The methodology cannot be used to model what experts are doing because it places the human experts outside the model where they are defined as sinks and sources (Floyd, 1984).

The methodology presupposes that the tasks in the field of work are partitioned into elementary sub-tasks which can be distributed to a number of specialists in order to solve them via the heavy exchange of structured data.

The methodology is only useful for modelling in transaction-oriented procedural systems containing no interpretations, best judgements etc. from the actors. The cooperative aspects of the work arrangements are not reflected in any other sense than the actual data flow among the actors. The semantics of the data flow, the semantics of the mechanisms of interaction etc., are not reflected at all.

Control Based Approaches

Control requirements define how a system controls its environment and how the environment controls the system. Control influences occur not only between the environment and the system but also between the elements of the environment themselves. Control relationships between the proposed system and its environment are largely due to the need to conform to, enforce, or assist control relationships between elements of the environment. In essence, control can be viewed as a distributed layered process through levels of the environment culminating in the system as the service provider at the lowest level.

In control-oriented methods the primary focus is the definition and description of how the environment is going to control the intended system or how the intended system is going to control its environment. They emphasise synchronisation, deadlock, exclusion, concurrency, process activation and deactivation. These methods are in applications that tend to have strict timing requirements. In the next section we will review two of them.

Software Requirements Engineering Methodology — SREM

SREM was developed in 1977 by TRW, Inc., under contract to the US Air Force Command, Control, and Intelligence (C^2I) systems. SREM was specifically designed to describe and analyse real-time, stimulus-response requirements of single-computer, missile systems (Alford, 1977, 1985). SREM is a formal integrated approach to requirements engineering activities. It begins when the

system analysis has identified system functions and functional interfaces between the subsystems and system operating rules, and the top-level system requirements have been allocated to the computing resource.

SREM includes a machine-processable Requirements Statement Language (RSL) and a Requirements Engineering Validation System (REVS) to automatically process the requirements statements and to perform a series of analysis and simulations on its centralised database. REVS constitutes the automated-tolls part of SREM. It uses a relational database (called Abstract System Semantic Model (ASSM)) to capture requirements processes from RSL.

The SREM Requirements Statement Language (RSL) expresses software requirements in terms of processing paths — that is, the sequences of data processing required to operate on an input stimulus to the software, and produce an output response. The first step in developing an SREM specification is to write it as a description of processing paths. Thus, if some data is input to a processing path, and a response is required at some other point, this is written in the specification as a fundamental component. To produce several thousand paths for a complex system, a method was developed to integrate simple paths into requirements networks called R-Nets. Thus, all paths initiated by the same input interface are integrated into a common network. Figure 25 uses the example of the *cash withdrawal* service of the automated teller machine to illustrates a network with data coming across the interface (shown as a hexagon). All data coming across the interface have common processing, as indicated by the first box. The OR (+) node specifies a conditional processing, of one of a choice of branches. The AND (&) node is next encountered to indicate a 'dont care' parallelism. That is, either branch may be processed first after the AND node. The decision is left as an option to the designer. Validation points (dark circles) are used to specify performance requirements in an explicitly testable fashion and are added to the paths close to the interfaces.

The advantages of SREM include:

- Verification of data-flow consistency,
- Specification of performance in testable terms (i.e., accuracies of calculation and response-time limits of paths of processing); and
- Traceability of system-level requirements to processing requirements.

The limitations of SREM include:

- SREM forces the requirements engineer to express requirements at a level of detail that conventional development techniques do not usually reach until test planning identifies the stimulus-response threads to be tested. SREM requires more effort during the requirements phase.
- SREM does not directly specify a distributed design because its model assumes that a message is completely processed before the next message is input.

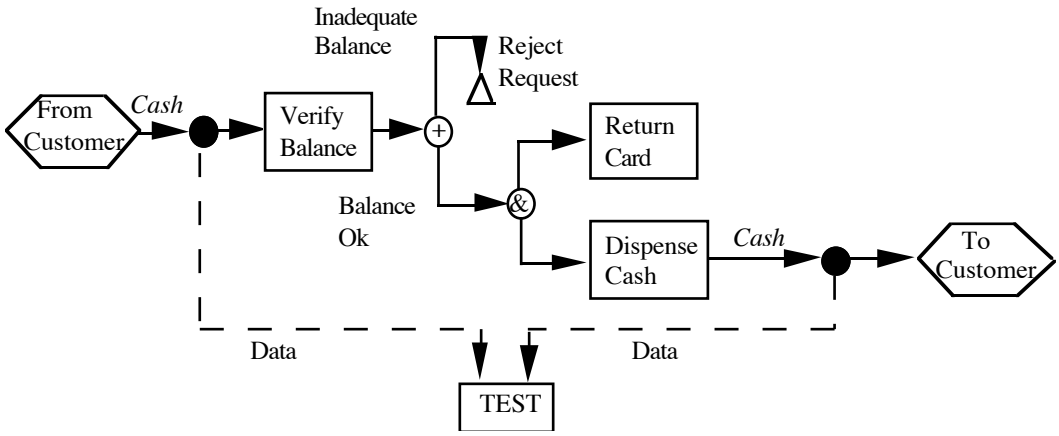

Figure 25 — An Example of path analysis structure for an ATM service.

StateCharts

Extensions to finite state machines (FSM) called statecharts were proposed by Harel (1988a&b) to model complex real-time system behaviour unambiguously. The extensions provide a notation and a set of conventions that facilitate the hierarchical decomposition of FSMs and a mechanism for communication between concurrent FSMs. The first extension to FSMs suggested by Harel allows a transition that is a function not only of an external stimulus but also of the truth of a particular condition. the next simple extension is the *superstate*. The superstate can be used to aggregate sets of states with common transitions.

Harel also introduces the concept of the default entry state. This is the subordinate state of a superstate into which the FSM enters if no other subordinate state is specified as the next state. The default state is specified with a small arrow coming into the state.

Figure 26 illustrates the conventional finite state model of the cash withdrawal service of the ATM, modelled as simplified ATM states. The states comprise idle, validating, verifying and dispensing states. The machine makes a transition to the validating state when a cash-card is inserted. In the validating state the machine validates the card and checks the customer PIN; an invalid card or PIN transitions the machine to idle state. A *cash* request event causes the machine to change from the validating to the verifying state. In the verifying state the customer balance is checked against the cash requested. An inadequate balance invokes a machine transitions into the idle state while an adequate balance results in cash being dispensed. The transition events are represented in italics and system states as round cornered rectangles; actions are preceded by a forward slash (/)

Figure 27 shows the equivalent Harel finite state model of the same service. The system is reduced to two states; the *idle* and *active*. Active state is an aggregate (superstate) of three subordinate states; validating, verifying and dispensing. All the subordinate states transition to idle; however the transition from idle to active state is to validating, indicated by the default entry state arrow on top of validating.

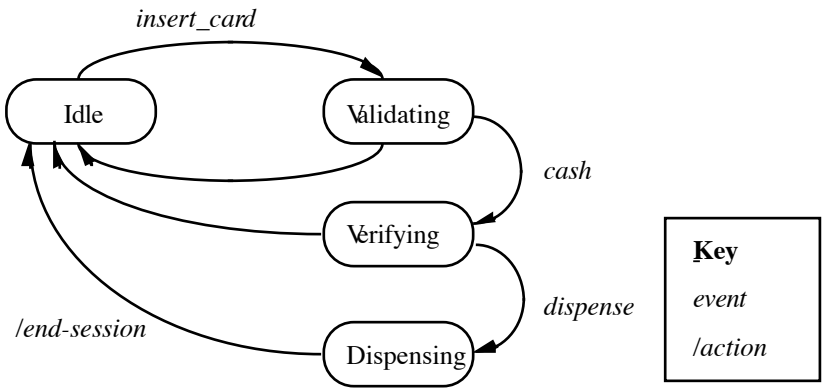

Figure 26 — Conventional finite state model of the cash withdrawal service

Harel's statecharts make its easy to model complex real-time system behaviour unambiguously. Ward (Ward and Mellor, 1986), Hatley (Hatley and Pirbhai 1987) and Yourdon (1989) had all proposed modelling real-time systems as hierarchies of data flow diagrams in which the behaviour of selected processes is defined as a finite machine. However their levelling conventions relied on hierarchical relationships inherent in DFDs; there was no concept in these extensions to structured analysis that captured hierarchies of finite state machines.

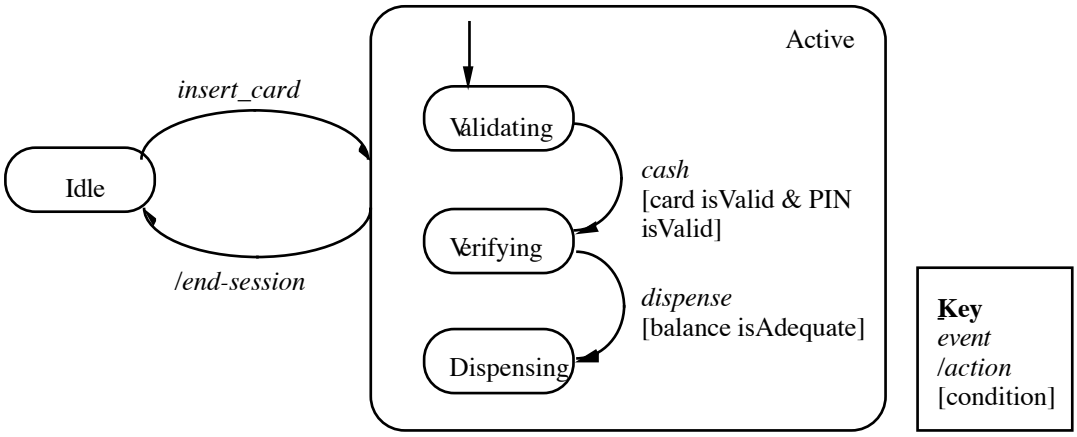

Figure 27 — Finite state model of the cash withdrawal service using Harel notational extensions

Statecharts provide a way of detecting static, structural behavioural and protocol errors. It also provides an explicit structural view of the system independent of its behavioural views. Statecharts, however, suffers a similar problem to other formal approaches in that it is difficult to understand for the non-computer scientist.

PAISLey

The Process-oriented Applicative and Interpretable Specification Language (PAISLey) was developed by Pamela Zave at the University of Maryland and later at AT&T Bell Laboratories (Zave, 1982; Zave and Shell 1986). PAISLey is a language for specifying embedded systems using an operational approach. This

means that the resulting specification is executable, the resulting behaviour is assumed to mimic the behaviour of the intended system.

PAISLey uses a simple but rigorous language adopted from the disciplines of asynchronous processes and functional programming. When using PAISLey, the requirements writer decomposes both the system under specification and its environment into sets of asynchronous interacting processes. Once this is accomplished, each process must be defined. This is done by defining the range of possible states in which the process may enter (i.e. state space) and by declaring and defining sets of functions that define how the processes change state (successor functions) and how the processes interact (exchange functions). Processes that are part of the environment are not treated differently to those that are part of the system.

PAISLey has been criticised for producing design rather than requirements specifications. The reasons given for this is that the resulting specification is virtually impossible for the typical customer or user to understand, and that writing in PAISLey requires a lot of rigorous detailed ideas.

Data and Object Approaches

Notions of control, process and functional transformation have dominated the requirements definition methodologies available until recently with the emergence of object oriented systems. The information model popularised by Chen (1976) is the closest precursor to object-oriented analysis and design. The model is based on refining hierarchies of sub-types and associated objects; and although this approach closely mirrors the object-oriented approach it is lacking in data encapsulation. The first published material on object-oriented analysis was by Shlaer and Mellor (1988). Since then several other requirements definition methods based on object orientation have been published the most notable of these are by Berard (1993), Bailin (1989), Colbert (1989), Coad and Yourdon (1989), Wirfs-Brock *et al* (1990), Rumbaugh *et al* (1991) and Martin and Odell (1992). A review of each of these approaches is provided in the next section. We review the method's view of the object-oriented concepts such as class, operation, state and inheritance. We also discuss the notations used and the development context within which the method can be placed.

Shlaer and Mellor

The Shlaer and Mellor (1988) method is intended to cover the requirements analysis and design phases of the software development process. The method begins by identifying the objects in the problem domain; objects are defined as tangible things, roles, incidents interactions and specifications. General guidelines are provided for identifying objects in the problem space. The next step associates the objects with their attributes and finally maps out the different relationships between the entities. A state transition model is used to model the object states and to verify the interactions between the state models and the object communications model.

Operations are derived from events drawn from state process tables. Two simple rules are used to define operations:

- If the event generated by the event generator does not cause a new instance of the object to be created, define the corresponding event taker as an instance operation. Name it TakeEvent<event label>.
- If the event generated by the event generator causes a new instance of the object to be created, define the corresponding event taker as a class operation and name it Take and Create <event label>.

The method partitions the problem domain based on the concept that objects in an information model fall into clusters: objects that are interconnected with one another by many relationships. This is as opposed to the fact that relatively few relationships connect objects in different clusters. Partitions are referred to in the method as subsystems.

The method uses both graphical and structured natural language notation to define requirements. Inheritance diagrams are used to represent specialisation and object communication diagrams to represent the communication between objects. Module interfaces are described using what are known as class diagrams. Concepts not supported by the notation include; meta-class, aggregation, concurrency and time. Reuse is mentioned but only in the context of reusing processes.

In summary, the Shlaer and Mellor approach is more representative of information modelling than object-orientation. It fails to account for several fundamental object-oriented concepts. The notation provides no way of expressing such concepts as messages, methods and procedure encapsulation. The method is not automated and relies on paper diagramming and unique naming conventions to describe the resulting system. The approach nevertheless provides the specifier with a starting point.

Coad and Yourdon

Coad and Yourdon (1990) define an object as an abstraction of something in the problem domain, reflecting the capabilities of a system to keep information about it, interact with it or both. They further define this to mean an encapsulation of an object's attributes and its exclusive services. They define a class as a description of one or more objects with a uniform set of attributes and services, including a description of how to create new instances of the class. An operation is defined to be a service: a specific behaviour that an object is responsible for exhibiting.

Coad and Yourdon's object-oriented analysis (OOA) is a five step method corresponding to identifying objects, subjects, attributes and services. Coad and Yourdon adopt a layered notation where the first layer is the subject layer which is used to guide the model reader, the second layer is the attributes layer and the third layer is the services layer. They provide a guideline similar to that provided by Shlaer and Mellor for identifying objects. Objects are roles, tangible 'things', structures etc. As the objects in the problem domain are identified, several structures may appear and the model may become complicated and cluttered. OOA identifies two types of structures: classification and assembly. To simplify the

understanding of the objects underlying these structures, they are viewed more abstractly as subjects. The next step involves defining the attributes associated with the objects. The last step in OOA is specifying the services performed by each object. According to Coad and Yourdon there are three types of service: occur, calculate and monitor. Occurrence services are related to creation, deletion and modification of instances of objects. Calculate services are used when an object requires the calculated result from another object. Each 'calculate' service is explicitly placed in the model. Monitor services are needed if an object is to continually monitor a process for a condition or event.

Coad and Yourdon's notation uses *whole-part* to represent aggregation and *gen-part* to represent specialisation. Communication between objects is modelled using message diagrams and object state is modelled using state diagrams. The notation does not provide a way of representing time.

Their approach is one of the few object-oriented approaches that provides a unique notation. Coad and Yourdon provide a set of guidelines on object identification and a way of structuring the complexities that arise from object identification. However, control-flow is not modelled in their methodology beyond being seen as messages. Issues of scheduling are not included in the notation.

Colbert

Colbert's (1989) approach distinguishes between active and passive objects. An active object "displays independent motive power" and a passive object acts only under motivation of an active object. Examples of active objects includes entities such as sensors and human beings. The concept of a meta-class is not supported in the method. The external behaviour of an object is seen as relating the operations performed on it to the operations it performs on other objects. Its internal behaviour relates the operations performed on it to the operations it performs on its component objects.

The Colbert approach recommends four steps:

1. Object-interaction specification
2. Object-class specification
3. Behaviour specification
4. Attribute specification

The Colbert notation includes object interaction diagrams for modelling communication between objects, object hierarchy diagrams for representing aggregation, and object-class diagrams for specialisations. Module interface descriptions are represented using an ADA like structure and object state changes using state transition diagrams. Colbert does not provide for timing in his notation.

Martin and Odell

Martin and Odell (1992) define an object as anything to which a concept or object type applies. A class is defined as an implementation of a concept or object type; the concept of a meta class is not incorporated in the method. An operation is defined

as a process that can be requested as a unit. The authors also add that operations may or may not change the state of objects. Functions are operations that do not change the object state. However, in an event schema, the operations that result in events do change states. Each state changing operation is called a transaction. A transaction is a process or a series of processes acting as a unit to change the state of an object. Instances of operations are called invoked operations.

The method can be summarised thus:

1. Define analysis goals
2. Clarify event type
3. Generalise event type
4. Define operation conditions
5. Identify operation causes
6. Refine cycle results

The Martin and Odell notation supports concepts of object aggregation and specialisation through the use of a 'composed-of' relation and object generalisation diagrams. Communication between objects is modelled using object flow diagrams and object state changes using state change diagrams. The method however provides no obvious partitioning mechanism.

Berard

The Berard (1993) approach can be summarised in 8 steps:

1. Identify the sources of requirements information
2. Characterise the sources.
3. Identify candidate objects
4. Build object-oriented models of both the problem and potential solution, as necessary.
5. Re-localise the information around the appropriate candidate objects.
6. Select, create, and verify candidate objects.
7. Assign candidate objects to appropriate section of the object-oriented requirements specification.
8. Develop and refine precise and concise system description.

Berard defines objects as the real and conceptual things we find in the world around us. Hardware, software and concepts such as velocity are all valid objects. Berard defines a class as an abstraction of objects; the class may itself be parameterised (i.e., it represents a family of very closely related classes). The method advocates that relevant objects should be identified through a process of domain analysis, which seeks to identify reusable items localised around objects and kits. A kit is defined as a collection of objects all of which support a single large, coherent, object-oriented concept. Kits are seen as "granular", so that while all the components of a kit are logically related, there are very few physical connections that bind them together.

Berard describes an operation as an action which is suffered by, or required of, an object and explains that an operation may be a selector, constructor or iterator. Operations are contained in an object's interface. The concept of a metaclass is also incorporated in the method and is viewed as a class whose instances are themselves classes.

The Berard approach is supported by several notations. Object-message diagrams are used to model communication between objects, and semantic network diagrams are used to model aggregation and specialisation. Module interfaces are described using program unit graphs. Dynamic concepts such as timing and object state changes are represented using timing and state transition diagrams respectively. Berard also uses petri-nets to model the interaction of multiple objects in various states, interacting over time.

Wirfs-Brock

The Wirfs-Brock (Wirfs-Brock *et al*, 1990) approach perceives objects as cooperating and collaborating agents that have responsibilities. Objects can be *clients* or *servers* and their interactions are prescribed by *collaborations*. A class is defined as a group (set) of objects that share the same behaviour, there is no explicit mention of meta-class in the method. The approach introduces the concept of a *subsystem* as a partitioning mechanism. A subsystem is a set of classes (and possibly other subsystems) collaborating to fulfil a set of responsibilities. The Wirfs-Brock approach can be summarised thus:

1. Study the textual requirements.
2. Extract noun phrases and build list.
3 Identify candidate classes from noun phrases.
4. Identify candidates for abstract superclasses.
5. Annotate each class with its purpose.
6. Identify classes with their responsibilities using (5) and the actions associated with class from (1).
7. Find collaborations by examining the responsibilities associated with classes.
8. Discard classes that do not collaborate with other classes.
9. Construct class hierarchies.
10. Construct contracts defined by each class.
11. Draw a complete collaborations graph of your system.
12. Identify possible subsystems within the system and simplify collaborations.
13. Construct protocols for each class.
14. Write a design specification.

The Wirfs-Brock entity relationship notation includes, in addition to *part-of* relationships, an *is-analogous-to* relationship to model relationships of classes that share the same responsibilities. It also introduces new concepts such as class card

to represent aggregation. Communication between objects is modelled as collaborations. Object specialisations are represented by hierarchies. The approach provides no support for modelling changes in object state and object interactions over time.

The Wirfs-Brock approach encourages anthropomorphic thinking. Objects are perceived as cooperating and collaborating agents; they have responsibilities; they become clients and servers; their interaction is prescribed by contracts. The description of the dynamics of objects in the Wirfs-Brock approach is not typical of other object-oriented approaches. There are no state transition or data flow diagrams. The behaviour of objects is instead formulated in terms of contracts, responsibilities and messages.

Rumbaugh

Rumbaugh (Rumbaugh *et al,* 1991) defines an object as a concept, abstraction or thing with crisp boundaries and meaning for the problem at hand. Classes are viewed as a group of objects with similar properties (attributes), common behaviour (typified by operations), common relationships to other objects and common semantics. A meta-class is a class describing other classes and an operation is seen as a function or transformation that may be applied to or by objects in a class.

Rumbaugh defines a subsystem as a major component of the system organised around some coherent theme, and uses this to divide the system into partitions and layers. A partition is a subsystem that provides a particular kind of service. A partition may itself be constructed from lower level subsystems. A layer is a subsystem that provides multiple services, all of which are at the same level of abstraction, built on subsystems at a lower level of abstraction.

The object-oriented modelling approach proposed by Rumbaugh covers not only the requirements definition phase of software development but also provides a transition to design. Rumbaugh has added functional modelling to object modelling and dynamic modelling to capture the semantics of operations. Operations are further augmented with data-flow diagrams.

The Rumbaugh approach uses an extended state transition notation to model object behaviour. The state model used is based on an extension of the work of Harel (1988) on statecharts. The state notation distinguishes between activities and actions, and allows for abstraction. The activity within a state forms the basis for further decomposition.

The Rumbaugh method can be summarised thus:

1. Analysis
   - Initial problem description
   - Build an object model
2. Develop dynamic model
3. Construct functional model
4. Verify, iterate and refine the three models

Rumbaugh's object-oriented model incorporates functional and data flow models to augment the operational aspect of the object model. Object behaviour is modelled using an extended state transition notation that supports the nesting of state transition diagrams. Their approach supports two types concurrency inside an object. In the first case an object is seen as an aggregate of component objects where parallelism is induced by the (relative) autonomy of the components. In the second case, an object harbors multiple (semi) independent state transition diagrams.

Other Object-Oriented Methods

Other object oriented analysis methods have been advanced by Wilkinson (1990), Jacobson (1987), and Bailin (1989). Wilkinson uses an approach that borrows ideas from Shlaer and from Coad and Yourdon. He uses this approach to specify a commodity paging system. The Wilkinson approach begins by defining the system context, this sets out the various interfaces of the system. The interface is then described in terms of the data and control information that affect it. Wilkinson has incorporated scheduling in his interface description. The next step involves modelling the relationships between the entities that interact with the system. A special *schedule* entity is incorporated in the entity relationship diagram. Wilkinson uses a notation similar to Coad and Yourdon's to capture the organisational structure of the entities. A data-flow diagram is used to show the processing and control-flow within the system. Wilkinson models the different states of the system using a state machine.

Wilkinson does not have a unique notation as such and borrows from other approaches. The Wilkinson approach is the only one to provide for the modelling of control and scheduling issues.

The Jacobson (1987) method is provided with very limited documentation. The method was developed by Objective Systems as part of a large project for the Swedish telephone company, Ericsson.

Bailin (1989) proposes an object-oriented requirements specification approach that models the problem domain as a hierarchy of entity data-flow diagrams (EDFDs) and a set of entity-relationship diagrams. Bailin's EDFDs are used just like conventional DFDs except that the nodes fall into two different categories: entities and functions. Every function must occur in the context of an entity, i.e. it must be performed by or act on an entity. Bailin also distinguishes between passive and active entities. Active entities are seen as processes and passive entities as data flows. In EDFDs active entities are represented as diagram nodes while passive entities appear as data flows or data-stores.

Bailin's approach is a seven step method outlined below:

1. Identify key problem domain entities
2. Distinguish between active and passive entities
3. Establish data flow between active entities
4. Decompose entities (or functions) into sub-entities and/or functions.

5. Check for new entities
6. Group functions under new entities
7. Assign new entities to appropriate domains.

Bailin has used several ideas from structured analysis to develop his method. The result has been an object-oriented approach that attempts to integrate the behavioural modelling strengths of a functional approach with the benefits of an object-oriented approach. His EDFDs are, for example, a modification of data flow diagrams. Bailin does not account for certain object-oriented concepts like inheritance and message passing. Timing, scheduling, and control issues concerned with sequencing are not tackled in the method.

Embley and Kurtz (1992) have proposed a model driven approach in which objects are organised around an object-relationship model (ORM). Objects may be physical or conceptual; an object is a single entity or notion that is unique. Objects that belong together for some logical reason constitute a class. Each object class has a name that is generic and denotes any member of the object class. The Embley and Kurtz approach defines *actions* rather than operations. Actions are performed by objects and may cause events that create or destroy objects and relationships, and may send and receive messages. The class of 'high-level object' groups together objects, classes, relationship sets constraints, and notes into a single object class, thus providing a mechanism for organising the problem into manageable chunks.

The Embley-Kurtz method can be summarised thus:

1. Construct object relationship models
2. Construct object behaviour models
3. Construct object interaction models
4. Integrate models.

Most of the object-oriented methods discussed differ with respect to concepts, notation, integration and level of tool-support. Berard, Colbert and Wirfs-Brock seem to be the most "object-oriented", with their strict emphasis on objects, and the downplaying of associations and relationships. Coad and Yourdon, with their accommodation of data (attributes and instances variables), may be viewed as the next most "object-oriented". Rumbaugh, Embley and Kurtz, and Shlaer and Mellor, who place strong emphasis on associations and relationships, come next. Martin and Odell's approach is the least object-oriented, seemingly presenting slight extensions to the information model with a very heavy behavioural orientation.

Although the methods differ in relation to notations, the notations share a common weakness in informality. The semantics of the core notations advocated by the methods lack precision. As a result, users face a limitation on the rigor that they can achieve with a method.

The process of defining requirements differs dramatically from method to method. Rumbaugh, Shlaer and Mellor, Wirfs-Brock, and Berard each have relatively well elaborated processes. Colbert, and Embley and Kurtz are less defined, but still relatively clear. Martin and Odell present little in terms of process

definition. There is little to indicate how and where each model is to be developed during the process.

In terms of tool-support for the methods, Coad and Yourdon, Rumbaugh, and Shlaer and Mellor are each well supported with a significant number of tools. Embley and Kurtz, and Martin and Odell do not have any commercially available computer-based tools supporting their methods.

Only Berard attempts to support traditional software engineering principles of testability, traceability, reusability and conceptual integrity. These principles are only lightly addressed, if at all, by the other methods reviewed.

Knowledge Based Approaches

Knowledge-based schemes are increasingly being applied to requirements analysis (Balzer *et al,* 1978; Rich *et al*, 1987; Reubenstein and Waters, 1991; Davis 1990; Greenspan 1984). Several schemes for developing knowledge-based tools to support requirements engineering have been proposed.

Mathews and Ryan (1989) have proposed a scheme whereby domain knowledge is maintained in terms of high level abstract descriptions that are subsequently refined to generate requirements suited to the specific context of the user. Reubenstein has also suggested the use of specialised knowledge for translating informal requirements into formal specifications. Czuchry and Harris (1988) with KBRA, and Green *et al* (1986) with KBSA, have proposed a scheme for maintaining implementation knowledge for translating requirements into executable implementations. The aim of this approach is to provide the analyst with a quick way of simulating and animating requirements.

The use of domain and implementation knowledge for critiquing requirements and design from different perspectives has been proposed by Fickas (1987), and Fickas & Nagarajan (1988). Domain knowledge has also found use in the analysis of requirements for completeness and consistency (Fickas *et al,* 1991; Reubenstein and Waters, 1991; Czuchry and Harris, 1988).

Schemes have also been proposed whereby knowledge-based tools carry knowledge of specific methodologies and guide the requirements analysis process (Burlon *et al*, 1989; Ryan, 1989; Finkelstein and Fuks, 1989; Finkelstein *et al,* 1990). It has also been demonstrated that the knowledge of the requirements analysis tasks can be incorporated in tools to help analysts manage the low-level activities (Kaiser *et al,* 1988).

The section briefly introduces and discusses a sample of knowledge based approaches and their contributions to the requirements engineering process.

RML

Borgida *et al*, (1985) argue that the requirements for a software product often describe no more than its *functional specification*. However, to understand or use such a specification requires a great deal more information than is usually given in the specification alone. They cite the case of the system developers who are not

usually expert in the application domain. Borgida *et al* propose a language called RML to capture the attributes of, and interrelationships between, objects in the real world, and advocate that this serves as the software specification.

RML is based on an object-oriented framework. In this framework, constructing a requirements model comprises identifying the appropriate classes and describing them in terms of applicable properties and constraints. Each object specified by RML stands for some entity, or activity, or, more generally some concept in the world being modelled. For example in the ATM system there are objects corresponding to customers, account, balance, cash card, and so on. Objects are related to each other by property for example, customer *hasAccount* nn, nn *hasBalance* bb. The RML below shows the partial example of the ATM object *Customer*.

```
Customer
  parts
    name: PERSON_NAMES
    address:ADDRESSES
    hasPIN:PINs
    hasCard:CASH_CARDS
    hasAccount:ACCOUNT_NOs.
```

RML also provides for activities and assertions to be modelled as objects, for example the activity Withdraw_Cash can be modelled thus:

```
Withdraw_Cash
  participants
    home Customer: CUSTOMER
    customerAccount:ACCOUNT_NOs
    customer_card:CASH_CARD
    custome_PIN:PIN

  parts
    cash:GET_INFO
    update:UPDATE_ACCOUNT

  precondition
    canDispense ?:ADEQUATE_BALANCE
```

RML has incorporated the notion of time into the object-oriented framework. Time is viewed as an infinite and dense sequence of time points. Assertions are true at a given time point, and the value of an object's properties as well as its membership in a class is always evaluated with respect to a specified time.

Requirements Apprentice

Rich *et al* (1987) argue that requirements acquisition is one of the most important and least well supported parts of the software development process. They point out that most requirements analysis tools focus on what is usually the requirements validation phase, and do not address the key issue of how informal requirements

evolve into formal specifications (i.e. appropriate inputs for the tools). They propose a Requirements Apprentice (RA) as a way of bridging the gap between informal requirements and formal specification. The Apprentice is intended to assist an analyst in the creation and modification of software requirements. The Apprentice gathers information and knowledge about the particular domain of requirement using various knowledge acquisition techniques including dependency-directed reasoning, hybrid knowledge and the reuse of common forms *(cliches)*. Figure 28 shows how the RA interacts with other people involved in the software development process.

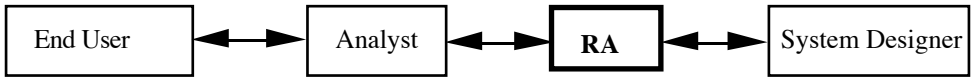

Figure 28 — Interacting with the RA

The development of the Requirements Apprentice is centred around two fundamental issues in knowledge acquisition. The first is informality and the process by which informal descriptions evolve into formal specifications. The second is the role of, and specific content, of prior knowledge of the common structures (cliches) of the domain. Rich *et al.* elaborate on the list of general features proposed by Balzer *et al.* (1978) as characterising informality and separating natural language from essential informality issues. The RA also augments this informality with the notion of cliches as a way of representing, organising and applying domain knowledge.

The output from the RA comes in two forms. First there are direct statements to the analyst, for example, describing a contradiction that has been discovered. The other output comprises textual displays generated from parts of the RA's internal representation of the evolving requirements.

Other Knowledge-Base Approaches

Feather (1993) argues that the ability to rapidly reconnoiter requirements, that is, construct, contrast and complete a system's requirements, would be highly beneficial. He suggests a simple network representation of the problem domain, that can be used to support the rapid reconnoitering of a system's requirements.

Feather suggests representing the domain requirements as a collection of entities and relationships. Entities comprise goals which represent the objectives of the overall domain, and policies which represent a means of achieving the goals. Relationships comprise four types:

- Supports — a policy may support a goal
- Impedes — a policy may impede a goal
- Augments — a policy may augment another goal
- subGoalOf — a policy may be a subgoal of another policy

Feather uses this model to represent domain requirements. This basic model is augmented with descriptions that support the representation of systems that are

instances of that domain. Additional entities include *instance* which defines a system to be viewed as an instance of the domain. Relationships include PolicyInstance and ImpossiblePolicyInstance which represent a policy of the system and a policy deemed impossible for this particular system respectively.

A simple instantiation of a resource-management domain is used (the wilderness permit system used by the U.S. Forest Service to control access to wilderness areas) to illustrate his approach. Feather demonstrates with this simple example how a variety of 'requirements freedoms', that is, inconsistency, incompleteness alternatives and comparisons are supported by his approach.

Feather has demonstrated how a very simple model of requirements for a domain can be built as an entity-relationship network and how this permits reconnoitering by issuing simple queries against stored information. However, Feather's model by his own admission (Feather, 1993) does not address certain pertinent issues in requirements engineering. It lacks any representation of many of the requirements and policies of resource management (i.e. it provides no way of representing 'confirmation' of reservations). It also lacks a connection to a quantitative or behavioural model. Finally the relationship between the domain model and instance level is rather simplistic.

The software development process is more unstructured and problem solving oriented than most engineers would like to believe. It involves the participation of many experts, in various aspects of the software development and application areas. In addition, each participant may have responsibilities and concerns which may change and shift as the software develops and evolves. Current CASE tools have tended to focus on streamlining the software engineering process through methodologies and representations that provide support for clerical, communication and co-ordination activities. The support of these tools by knowledge-based schemes seeks to enhance their potential by making them "knowledgeable".

It is still too early to judge the impact that knowledge-based schemes are going to have on requirements engineering and the software development process as a whole. However, current research has shown four key areas in which requirements engineering may derive benefit from knowledge-based schemes. The first of these areas is the analyst's intelligent assistant. Rich *et al* (1987) have demonstrated with the Requirements Apprentice (RA) how a variety of techniques including dependency-directed reasoning, hybrid knowledge representation and the reuse of common forms can be effective in formulating requirements. Another key area in which knowledge-based schemes may be useful in future is in translating informal requirements into formal specifications (Balzer *et al*, 1978; Reubenstein and Waters, 1991). Automated program synthesis has also been suggested as a possible area that may benefit from knowledge-based schemes. The initial synthesis of the design may be performed with a canned architecture used for all applications (Davis, 1982), with the requirements-implied architecture for operationally specified requirements (Zave, 1982), or with a knowledge base containing expert knowledge of optimal designs (Barstow 1985).

Formal Techniques

Requirements specification techniques can be categorised on a "formality" spectrum. Structured analysis with its use of a combination of text, diagrams, tables and simple notation used to describe and model software requirements falls at the informal end of the spectrum (Pressman 1992). At the formal end of the spectrum, formal syntax and semantics are used to specify system function and behaviour.

The objective of formal methods is to enable a software engineer to specify, develop, and verify a computer-based system by applying a rigorous mathematical notation. Formal methods focus on what is usually the validation phase of requirements. The main objective of this phase is to increase confidence that a given requirement actually corresponds to the user's desires.

A formal specification language is usually composed of three primary components:

- S*yntax* that defines the specific notation with which the specification is represented. The syntactic domain of a formal language specification is often based on a syntax that is derived from standard set theory notation and predicate calculus.
- *Semantics* that help to define a "universe of objects" that will be used to describe the system. The semantic domain of a specification indicates how the language represents system requirements.
- A set of *relations* that defines the rules that indicate which objects properly satisfy the specification.

Formal methods for requirements specification are moving slowly away from being a purely academic research interest to becoming important for industrial software development. This is particularly true in the case of systems where safety or security is critical (Spivey, 1990; Sommerville, 1992). However, the general acceptance of formal methods amongst software developers is still some way off (Greenspan, 1991; Davis, 1990). This can be attributed to several factors, including: the difficulty by system procurer (or users), and indeed some software developers in understanding the notations; the difficulty in formalising certain types of requirements, for example, the specification user interfaces; software management is conservative and unwilling to adopt new techniques whose payoff is not obvious.

The number of formal specification languages in use today can be broadly divided into two categories:

- Those that use model-based notations such as Z (Spivey 1990) and the Vienna Development Method (VDM) (Bjorner and Jones, 1978; Jones, 1990).
- Those based on process algebras such as Communicating Sequential Processes: CSP (Hoare 1985; Moore 1990), CCS (Milner 1989), ACP (Bergstra and Klopp 1984) and LOTOS (Bjorner and Jones 1978).

The use of process algebras are largely restricted to specialised application domains with model-based specifications being more generally applied.

The following sections briefly review the Z and VDM formal specification methods, and provide a discussion on the future of formal specification methods in software engineering.

Z

The mathematical notation used in Z is based on conventional set theory and predicate calculus with some minor extensions, which provide the expressive power to allow the specifier to deal with many structures which are often encountered during the development process.

A specification in Z is presented as a collection of schemas where a schema introduces some specification entities and sets out relationships between them. A Z schema comprises three main parts: the schema name; the schema declarations, which set out the names and types of the entities introduced in the schema; and the schema predicates, which set out the relationships between the entities in the signature by defining a predicate over the signature entities. A schema's general form is given below in figure 29:

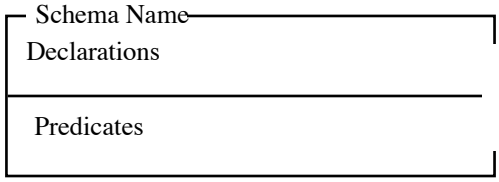

Figure 29 — General form of a Z Schema

where the variable declarations are of the form:

```
identifier: type
```

and the predicates give the properties of, and relationships between, the variables.

A schema may be used to describe either a state or an operation. To describe a state the declared variables form the components of the state and the predicates give the invariant properties of the state. For an operation, the declarations consist of the initial state components, the final state components, and the inputs and outputs of the operation. As a convention the final state component names are dashed versions of the initial state component names. For an operation, the predicates part describes the relation between the inputs, outputs, and initial and final states.

In summary, Z is a rigorous system specification technique. It uses set theory and logic to abstract the essential features of a design. The subsequent analysis and development can be carried out on well founded mathematical principles. Unlike many formal methods Z pays particular attention to presentational aspects. It's semi graphical notation and modular descriptions present specification in a more readable manner than most formal methods, and tends not to alienate first time users so much.

Some of the problems encountered with Z include:

- The notation is not suited to modelling control oriented problems.
- No account of time is made in the notation, making it difficult to model time intensive applications.
- There is no notation for modelling concurrency.
- Z is not readily integrated with other design techniques.

Vienna Development Method (VDM)

The Vienna Development Method (VDM), like Z, is a model based formal specification method that uses the set theory. VDM was originally developed by Jones and Bjorner (Bjorner and Jones 1978) as a mechanism for accurately defining the structure and semantics of programming languages. The main steps in a VDM development can be summarised thus:

1. List the state variables of the system.
2. Define a state invariant. This invariant must be verified at every operation of the VDM state description.
3. List the operations.

The specification of operations, done in terms of pre- and postconditions, establishes a link between the initial and final configurations of the state variable. A VDM development is made up several *state descriptions* at successive levels of abstraction, and *implementation steps* which link the state descriptions. A state description comprises:

- State variables
- Operations on the variables, specified in terms of pre-conditions and post-conditions
- An invariant on the state variables, which must be verified before and after the execution of any operation.

The implementation of an abstract state description S_i by means of a more concrete one S_{i+1}describes:

- Either data reification, i.e. a refinement in the data structures of state variables
- Or an operation decomposition, that is how the operations of S_{i+1} implement the ones of S_i.

At each stage of the VDM development, *proof obligations* have to be verified to ensure that the operations of a given state description are compatible with the state invariant and are implementable. Other proof obligations validate the implementation of one state description by means of a more concrete one. VDM provides rules to systematically derive these proof obligations.

VDM is a formal development method based on the set theoric approach. It has been used extensively to define the formal semantics of programming languages into machine language. It has also been used to define a relational data base system (Neuhold and Olnhoff 1980). Although the developers of VDM claim wide

applicability, VDM has not been successfully demonstrated in applications outside a quite narrow range.

Other Formal Methods

Process algebras are generally recognised as convenient tools for formally describing the features of concurrent systems at different levels of abstraction. These features include synchronisation, deadlock, exclusion, concurrency and process activation. CSP (Hoare, 1985) and LOTOS (Bolognesi and Brinksma, 1987) are two process algebras that are being used to enable concurrent systems to be formally specified and analysed. In CSP a process is described in terms of external *events*, that is, the communication it has with other processes. The history of a process is represented by a *trace* which is a finite sequence of events. A system represented in CSP can be analysed to determine its behaviour.

The specification language LOTOS has been developed within the International Organisation for Standardisation (ISO) and has been recently standardised as a formalism for specifying concurrent systems specially suited to run the Open System Interconnection (OSI) computer network architecture. The main goal of LOTOS is to support the production of OSI standard specifications which need to be unambiguous, precise and complete.

Summary

There is little doubt that formal specification methods such as Z and VDM offer some advantages over other less formal methods. Because they are based on mathematical formalisms, it is possible to automatically verify the correctness, completeness and consistency checking of the specification. In addition, formal specification removes ambiguity and encourage greater rigor in the early stages of software engineering.

However, formal methods suffer from several problems. They focus primarily on function and data. Timing, control and behavioural aspects of a problem are more difficult to represent. In addition, some requirements (e.g. human factors issues) are highly subjective and can only be determined through complex empirical evaluations, possibly through prototyping.

Formal methods focus on what is usually the validation phase of requirements. The main objective of this phase is to increase confidence that a given requirement actually corresponds to the users' desires. This does not, however, address the key question of how the requirement is constructed in the first place.

Another reason for the low spread of formal methods is the lack of adequate tool support (Davis, 1990; Sommerville, 1992). Some tools have been developed to automate certain elements of the specification and verification process for Z, VDM, CSP and Larch. Finally, the use of formal methods in software specification also represents a significant "cultural shock" to many software engineers due to the difficulty in learning them.

Formal methods will take some time to find general acceptance in the software engineering community. Greenspan (1991) concludes that many of the obstacles to

success in requirements engineering are not amenable to formal methods. Even if some aspects of the problems could be formalised, it may not be worth doing, because the amount of knowledge needed may be too great. However, research on formal methods for specifying and reasoning about systems and their requirements have had, and will continue to have, great value in increasing our understanding of system development.

Viewpoint Approaches

Several requirements definition languages based on models that adopt a multiple viewpoint approach to requirements capture and analysis have been advanced as powerful alternatives to those based on global reasoning. Structured Analysis and Design Technique (SADT) (Ross, 1977), Structured Requirements Definition (SRD) (Orr, 1981), and Controlled Requirements Expression (CORE) (Mullery, 1979) all support, implicitly or otherwise, some variant of the viewpoint approach. Finkelstein (Finkelstein and Fuks, 1989; Finkelstein *et al*, 1990, 1992) has advanced a software development approach based on viewpoints as a framework for distributed software development. Leite (1988, 1989) has proposed an early validation scheme based on viewpoints, as a means for identifying and classifying problems related to correctness and completeness in the process of requirements elicitation. Van Lamsweerde, Fickas and Dardenne (Fickas *et al,* 1991) see a goal directed approach as providing the 'roots' at which conflicts should be resolved and multiple viewpoints reconciled. Dubois, Hagelstein, and Rifuat (Dubois *et al*, 1988) have developed an approach to requirements that focuses on the larger system formed by the computer and its environment rather than on the required system software.

The next section discusses five methods that adopt a form of viewpoint oriented strategy to requirements analysis and definition. The example of the automated teller machine (ATM) used in earlier discussions is returned to in order to demonstrate each approach.

Structured Requirements Definition (SRD)

The SRD methodology is based on what its author Ken Orr calls 'defining the application context'. The four step process is given below:

1. Define a user-level data flow diagram by interviewing each individual in the organisation to be analysed, currently performing some useful task. Record the inputs/outputs as a data flow model.
2. Combine all like user-level data flow diagrams to create one integrated data flow diagram. Conflicting data flows can be resolved at this stage.
3. Define the application level data-flow diagram by drawing a dotted line around that part of the combined user data flow diagram that corresponds to the organisation being analysed.
4. Define the application-level functions by showing the order that comprise each function being performed by the organisation for its environment.

The SRD perspectives or viewpoints are individuals in the organisation being specified that are doing or performing manually some aspect of what will be automated. Unfortunately, in the case of the ATM, the only individual that can be interviewed is the teller as it is the teller's task to be automated, (Figure 30). In performing her/his task the teller is supported by several other systems and individuals, who while their information may add to the overall understanding of the system, cannot be interviewed as they constitute viewpoints external to the system being automated. Step 2 of SRD is therefore not applicable in the case of the ATM. If we apply step three of SRD, and collapse the diagram around that part of the organisation that needs to be automated, we return to the teller, (Figure 31). SRD therefore does not work well if the system being specified does not involve the automation of several manual tasks.

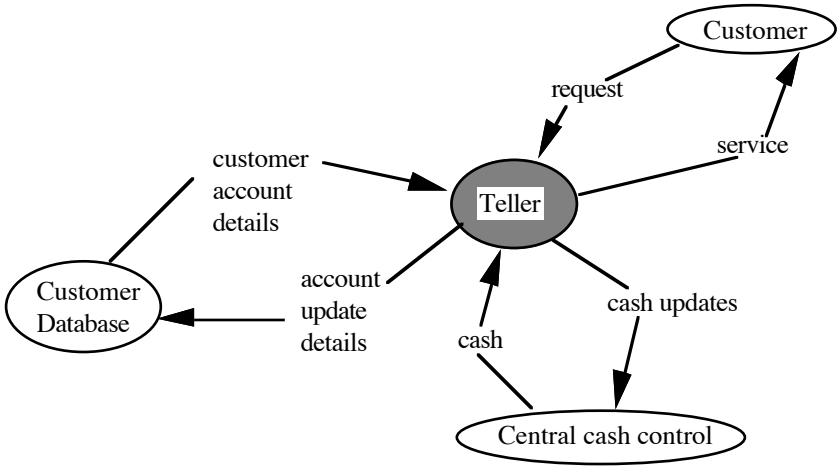

Figure 30 — User-level data flow diagram

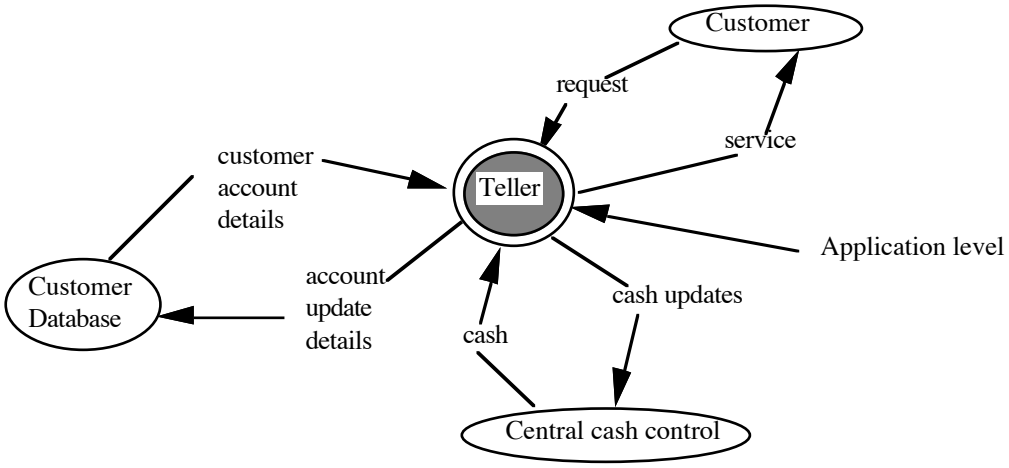

Figure 31 — Application level data flow diagram

SADT

Structured Analysis and Design Technique (SADT) has been discussed in an earlier section. The method is based on a data flow model that views the system as a set of interacting activities. A rectangular box representing the system's most abstract activity together with a set of data input, data output and control arrows forms the starting point for functional decomposition (Figure 23).

'Viewpoints' as described in SADT are an intuitive extension of its underlying modelling technique rather than seen as data sources and sinks. Figure 32 shows the most abstract level of the ATM, with two possible 'viewpoints'; the 'customer viewpoint' and the 'customer account database viewpoint'. The 'customer viewpoint' represents the source and destination of data associated with the bank customer, this includes customer requests, validation data (PIN) and Card. The 'customer account database viewpoint' represents the source and destination of information associated with the customer account database. The control information associated with these processes is shown coming in at the top of the box, it includes information related to card validation and service availability. The control information sets out the conditions under which the inputs coming in from the environment, are transformed into the outputs shown on the right. Because the control information affects the resultant data output, it can be associated with the viewpoints which act as the data sinks. The card validation process and service availability can both be associated with the 'customer viewpoint' as they affect the transformation of requests into services. SADT does not provide a framework for this type of broad association of control information with viewpoints.

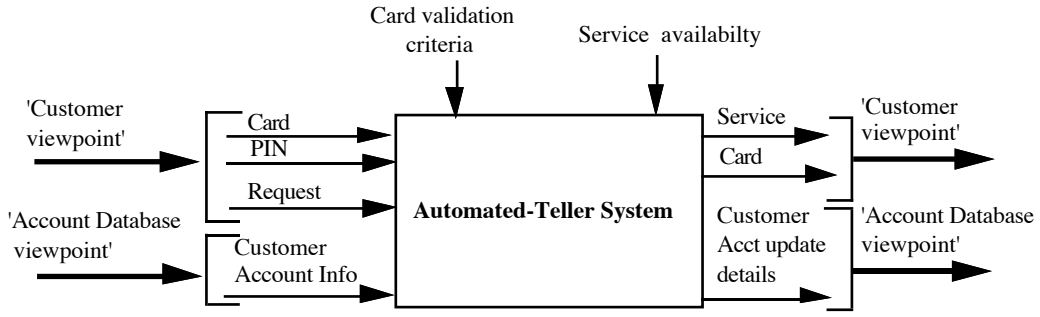

Figure 32 — SADT viewpoints on the ATM (first level)

CORE

Controlled REquirements Expression (CORE) is a functional requirements definition method based on a viewpoint approach. The method, developed for British Aerospace in the late 1970s by System Designers (Mullery, 1979), has been used to specify large aerospace systems by British Aerospace. Some notable systems include the Experimental Aircraft Programme (EAP) in the mid 1980s, in which CORE was used for system and software definition and, more recently, the

European Fighter Aircraft in which CORE has been chosen as the standard requirements analysis method.

CORE is one of the few requirements definition methodologies that explicitly adopts a viewpoint approach and attempts to attach some definition to it. CORE defines its viewpoints at two levels. The first level consists of all entities that interact or affect the system in some way. At this level CORE provides guidelines for distinguishing between functional and non-functional viewpoints. The second level is concerned with distinguishing between defining and bounding viewpoints. Bounding viewpoints are entities that interact indirectly with the system while defining viewpoints are sub-processes of the system, viewed in a top-down manner.

The CORE methodology comprises the following steps:

1. Viewpoint identification
2. Viewpoint structuring
3. Tabular collection
4. Data structuring
5. Single Viewpoint modelling
6. Combined Viewpoint modelling
7. Constraints Analysis

The first step in CORE involves identifying possible viewpoints. From this list functional and non-functional viewpoints are identified. There are no hard and fast rules for identifying relevant viewpoints, rather CORE suggests a session of 'brainstorming' composed of the users, buyers and specifiers of the system. After a set of possible viewpoints have been identified, they are distilled into a set of functional and non-functional viewpoints. The final step in viewpoint identification involves dividing functional viewpoints into a set of bounding and defining viewpoints, Figure 33 shows the bounding and defining viewpoints of the ATM. Each bubble represents the most abstract form of the viewpoint.

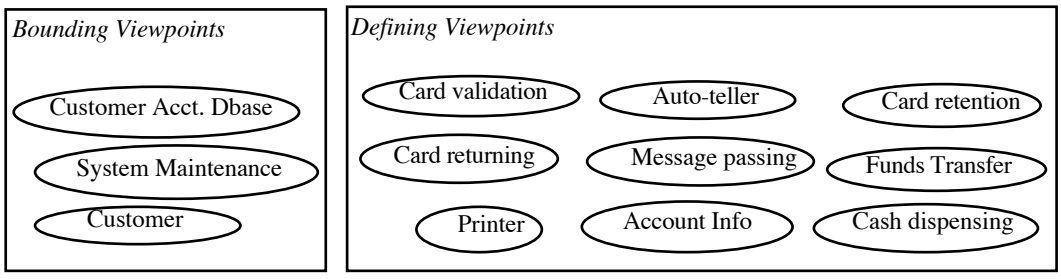

Figure 33 — Bounding and Defining Viewpoints

Viewpoint structuring provides a framework for requirements capture and analysis. It involves iteratively decomposing the 'target system' into a hierarchy of functional sub-systems, essentially performing a top-down decomposition. Structurally bounding viewpoints are placed at the same level as the target system.

Each functional subsystem constitutes a viewpoint. Figure 34 shows part of the viewpoint structure of the ATM.

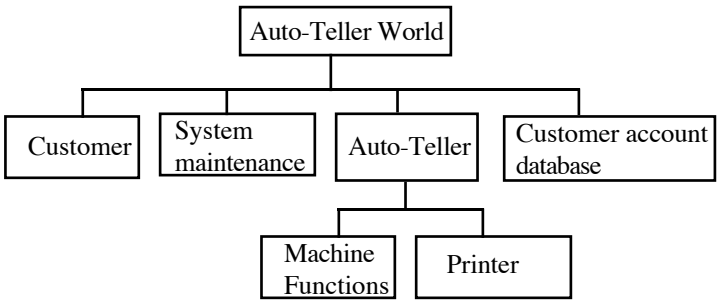

Figure34 — Viewpoint structuring for ATM

Step 3 in CORE is tabular collection. Tabular collection is a mechanism for gathering information about a viewpoint. Each viewpoint is considered in turn with respect to: the action it performs, the data used for these actions, the output data derived, the sources of the data, and the destination of the data. Tabular collections serve the purpose of exposing omissions and conflicts in the information flow across viewpoints. Figure 35 shows part of the tabular collection for the ATM.

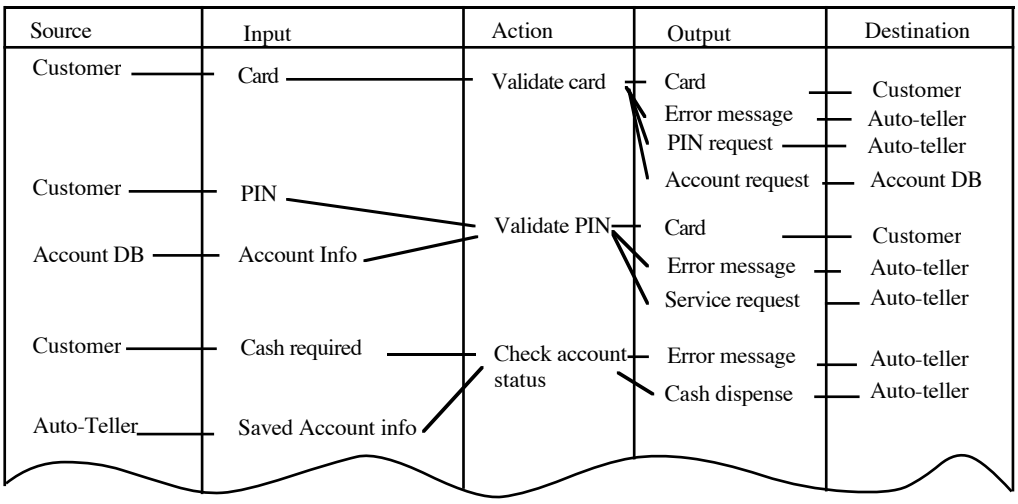

Figure 35 — Part of tabular collection for the ATM

Viewpoint Oriented Software Engineering (VOSE)

In Viewpoints Oriented Software Engineering VOSE, Finkelstein *et al*, (1990, 1992) propose the use of viewpoints as a way of managing the software development process. They argue that developing large software systems involves the participation of many experts, in various aspects of the software development and the application area. In addition, each participant may have responsibilities and concerns which may change and shift as the software develops and evolves. Finkelstein *et al* contend that the key to managing these activities and the associated forms of knowledge is to structure and contain them so as to provide a partitioned

distributable structure for the software development process, and a partitioned distributable structure for the software specification.

VOSE uses viewpoints to capture the role and responsibilities of a participant at a particular stage of the software development process. Viewpoints are identified by the role of the participant, the domain relevant to its interest and its knowledge about the domain. This knowledge is encapsulated in a viewpoint and represented using a single appropriate representative scheme (*style*). A VOSE viewpoint is defined as "a loosely coupled, locally managed object which encapsulates partial knowledge about the application domain, specified in a particular suitable formal representation, and partial knowledge of the process of software development". A viewpoint is seen as a combination of a style or representative scheme in which the viewpoint expresses what it sees, the domain it sees, a specification, a work plan that defines the conditions under which the specification can be changed, and a work record. VOSE viewpoints can be organised in configurations, which are collections of related viewpoints. A template is defined as a viewpoint which has only the style and work plan.

A configuration for a hypothetical problem domain may consist of templates with different styles 'viewing' the same partition of the problem domain, or templates with the same style 'viewing' different partitions of the problem domain. In the second case it is a simple matter to ensure that there is consistency in information flow between the different partitions, however, the same cannot be said of the first case. The difficulty arises in trying to map representative schemes whose modelling approaches conflict. A template based on an object-oriented model may, for example, require major recasting before it can map onto one based on a data-flow model. One therefore needs to be selective in picking representative schemes.

Figure 36 shows two domains in the ATM world, the customer domain and the ATM domain, and the activities associated with them. The main activity associated with the customer domain is service requests, and the with ATM domain, the provision of services. Figure 36 shows how VOSE models the service request of the customer domain and the cash withdrawal activity of the ATM domain. A configuration comprising three templates is used to model both domains. The template modelling the customer domain is based on the state machine model, two templates based on the state machine and data flow models are used to model the cash withdrawal activity of the ATM domain. Each domain defines the boundaries of the knowledge encapsulated by the viewpoint. As far as the state model of the ATM world is concerned, the customer and ATM domains see only those states relevant to themselves — the internal workings of one domain is hidden from the other. The data flow model of the ATM domain is only concerned with functions relevant to that domain.

Conflicts between the similar templates of the customer and ATM domains are resolved by checking for the loss of continuity between their respective models. The conflicts between the different templates of the two domains are resolved by identifying the correspondence between the models.

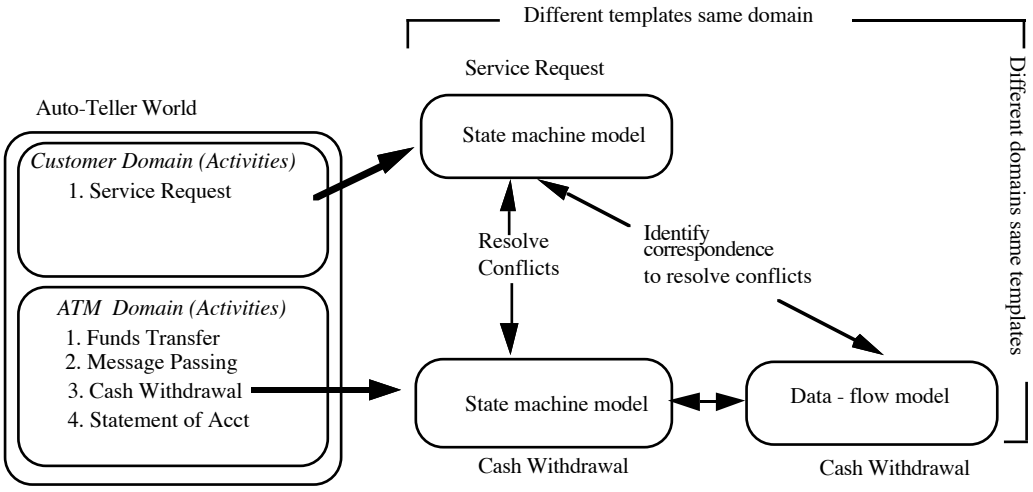

Figure 36 — Customer and ATM domains in the Auto-Teller World

Viewpoint Resolution Approach

In "Viewpoint Analysis: A Case Study", Leite (1989) advances a viewpoints resolution approach as a means for very early validation in the process of requirements elicitation. The objective of the approach is to identify and classify problems related to completeness and correctness. Leite uses the principle that many sources of information provide a better understanding of a subject than one global source. He contends that by using this principle in the process of requirements elicitation chances of detecting correctness and completeness are 'enhanced'. Leite defines a *viewpoint* as a standing or mental position used by an individual when examining a universe of discourse (the overall context in which the software will be developed, including all the sources of information and the people involved). A *perspective* is defined as a set of facts observed and modelled according to a particular aspect of reality. Viewpoint analysis uses three modelling aspects: the data perspective, the actor perspective and the process perspective. A *view* is defined as an integration of these perspectives. View integration is achieved by a view-construction process. A *hierarchy* is defined as the is-a hierarchy of concepts of the universe of discourse and the parts-of hierarchy of concepts in the universe of discourse.

Leite uses a viewpoint language, VWPL, to represent the viewpoints. The method involves at least two analysts (viewpoints) using VWPL to express their perception of the universe of discourse. The analysts use the data, process and actor perspectives, and the hierarchies is-a and parts-of to improve their own view. Each analyst then solves the internal conflicts and integrates the final perception into a view. This view is expressed in the process perspective together with the hierarchies. After that, both viewpoints are compared and analysed. In order to construct a view an analyst describes the problem using the three perspectives and two hierarchies. The perspectives and hierarchies are analysed and a 'list of discrepancies' and 'types of discrepancies' is produced. The next step is the

integration of the perspectives into a view. After two views are available it is possible to compare different viewpoints for correctness and completeness.

Viewpoints Oriented Requirements Definition

Viewpoint Oriented Requirements Definition method (VORD) (Kotonya and Sommerville, 1992), is a service-oriented viewpoint-based approach to requirements definition, intended to support an object-oriented development process. VORD defines a viewpoint thus:

- An *external* entity that interacts with the system being analysed *and*
- An entity that can exist *without* the presence of the system.

A viewpoint can interact with the system in three distinct ways:

- It may receive one or more services from the system.
- It may provide control information to the system.
- It may provide data to the system.

Viewpoints are not just end-users, but may include other systems and sub-systems that interact with the intended system. Thus, for example, an external database accessed by the system and an operator controlling the system are both valid viewpoints.

The service oriented view adopted by VORD provides a framework for addressing functional and non-functional requirements. Services are synonymous with functional requirements and have an intrinsic end-user association. Associating services with end-users (viewpoints) provides a powerful and flexible framework for addressing non-functional requirements from the user perspective. The framework provides for viewpoints to privately impose constraints on the services they receive and for different viewpoints to place different constraints on comparable services. Conflicting constraints are easily accommodated by the framework.

VORD adopts a two layered approach to analysis. The first layer, the viewpoint layer, is concerned with identifying user classes; capturing and structuring their requirements. The second layer, the system layer, reconciles and interprets user requirements into a cohesive object-oriented framework. This layer is essentially concerned with identifying key system entities responsible for the provision of services to the viewpoint layer, and reconciling them with viewpoint layer information.

The system layer is intended to support the development of an object-oriented design from a set of viewpoint-oriented requirements, and the system forms the transition from requirements to design. We will not be concerned with the details of the system layer as its not relevant to this discussion.

Viewpoint Structuring and Documentation

A VORD viewpoint is a hierarchy with differing levels of abstraction or specialisation. The top level contains its most abstract description and the lowest its most specialised. Each specialised level inherits the services, attributes, events and non-functional requirements of the more abstract viewpoints. The viewpoints,

nonetheless, may impose their own constraints on inherited services and may also have their own particular service requirements. A specialisation automatically inherits the services, events and attributes of its parent class, but has a provision for disallowing inheritance and describing its own special requirements. Figure 37 shows the ATM viewpoints.

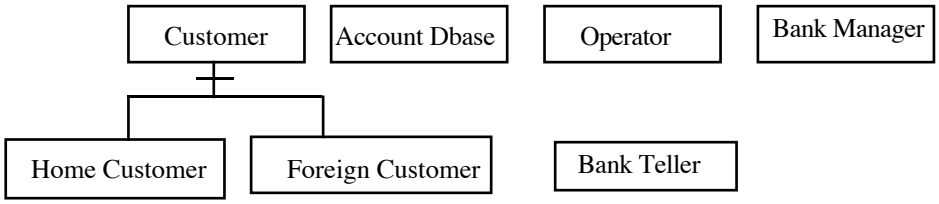

Figure 37 — ATM viewpoints

A viewpoint can interact with the system in three distinct ways; requiring service, providing data, or providing control information. VORD uses standard templates to describe viewpoint layer information. Each viewpoint is uniquely described by a standard template comprising:

Viewpoint reference:	Viewpoint name or identity.
Viewpoint attributes:	Components of the viewpoint whose values are relevant in the context of the intended system.
Viewpoint events:	Describes the sequence of events together with exceptions associated with the exchange of information between a viewpoint and the system. Events are collected into event scenarios. Each event scenario describes a type of viewpoint-system interaction.
Viewpoint services:	Synonymous with the functional requirements of the intended system.
Viewpoint instances:	Viewpoint specialisations.

A service template comprises six parts:

Service reference:	Service name or identity.
Service specification:	Service specification in appropriate formal and informal notations.
Service user:	Refers to the viewpoint receiving the service.
Service rationale:	Describes the reason for receiving viewpoint to require the service.
Service providers:	System-level entities responsible for providing service.

Figure 38 illustrates the viewpoint and service templates associated with the ATM *Customer* viewpoint.

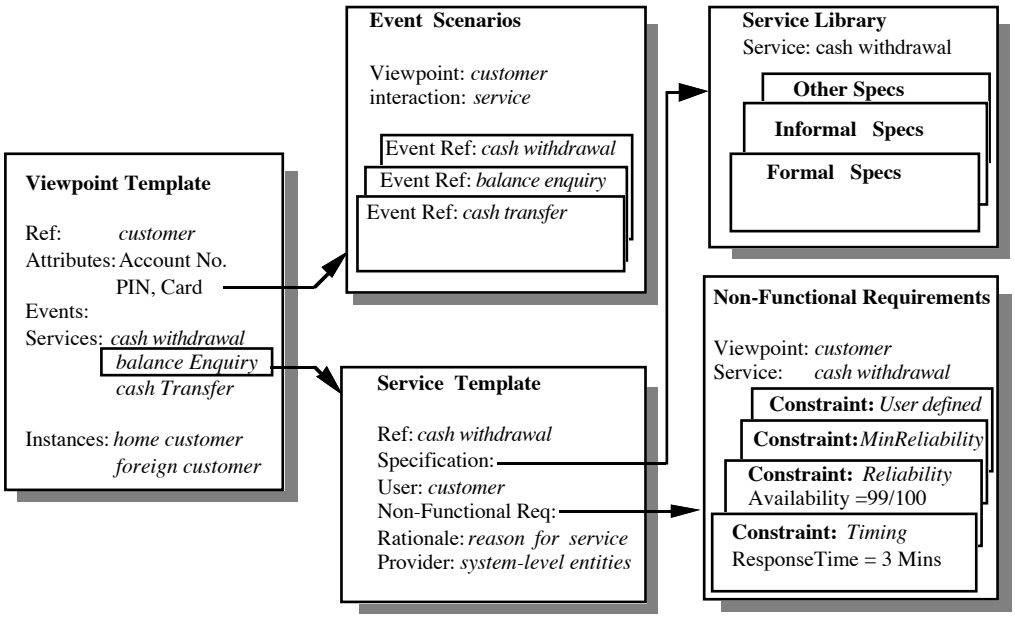

Figure 38 — Viewpoint and service templates associated the ATM *Customer*

Other Viewpoint Approaches

Other approaches include those proposed by van Lamsweerde, Fickas and Dardenne (Fickas *et al,* 1991) and Dubois (Dubois *et al,* 1991). The former propose a goal directed approach for model acquisition. They describe a conceptual meta-model in terms of which requirements models are acquired. The requirements model acquired is made of instances of the meta-concepts. Goals are seen as determining the respective roles of agents in the system and as providing a basis for defining which agents should best perform which actions (according to their responsibilities, ability, reliability, motivation). Goals are linked to agents in two different ways; an agent may be responsible for a system goal if it can guarantee, maybe jointly with others, that the goal will be achieved in the system. An agent may wish a private goal if it wants it to be met. The assignment of responsibility of system goals to agents is guided by the goals wished by each agent. Because system goals can be the responsibility of several agents it is possible that agents' private goals will be in conflict with system goals. The authors see goals as providing the "roots" at which conflicts should be resolved and multiple viewpoints should be reconciled.

Dubois *et al* have developed an approach to requirements based on an expressive formal language called ERAE. The authors contend that better requirements are obtained not by focusing on the needed system software but rather on the larger system formed by the computer and its environment: essentially the different perspectives of the system.

The shortcomings of the viewpoint approach adopted by both SRD and SADT have already been highlighted in discussions of the approaches. In summary both

SRD and SADT have an intuitive rather than a defined notion of a viewpoint, making it the secondary rather than primary technique used in the methodology. In both cases viewpoint analysis is not extended beyond data sourcing and sinking. Because SRD viewpoints are internal perspectives tied to people performing manual tasks in an organisation the approach fails to work in cases where the analysis involves only the automation of tasks associated with a single person.

Perhaps the major weakness of CORE is its poorly defined notion of a viewpoint. Because the CORE viewpoint can be any entity that interacts with the proposed system it is difficult say what is and what is not a valid viewpoint. This problem is further complicated by having what CORE calls defining and bounding viewpoints. Bounding viewpoints are supposed to be 'external' entities that exchange information with the system, while defining viewpoints are parts or sub-processes of the 'target' system. The bounding viewpoints are represented by entities while defining viewpoints are typified by functional processes.

CORE gives some guidelines in identifying 'reasonable' viewpoints but acknowledges that two analysts specifying the same problem are likely end up with completely different viewpoints. The drawback of CORE's weakly typed viewpoints is made manifest by the difficulty in automating the method beyond providing for basic diagramming and editing.

The major analysis in CORE is concentrated on what are really internal viewpoints — defining viewpoints. Bounding viewpoints which represent the system's interaction environment are not analysed beyond being seen as data sources and sinks. Because of the structure of the underlying CORE viewpoint model, it is difficult to model processes using different representations.

VOSE is both a requirements approach and an approach to facilitate distributed software development. VOSE aims to create a partitioned distributable framework for the software development process, that combines appropriate development methods, specification techniques and languages. Finkelstein sees the software development process as an activity involving many participants with varied expertise in aspects of the software development process and aspects of the application area. He proposes the use of viewpoints as the structuring vehicle to constrain the roles and responsibilities of the participants to aspects of the application domain relevant to their particular interests. A viewpoint is seen as encapsulating that aspect of the application domain relevant to the particular role, and utilising a single appropriate scheme to represent that knowledge. VOSE provides a framework for multiple representation of the application domain and uses this as a basis for resolving conflicting information in the development process. However, VOSE fails to provide a firm framework for resolving the conflicts between representations that have little correspondence. VOSE provides no obvious way of integrating functional and non-functional requirements and capturing the different classes of interacting entities.

Leite associates viewpoints with the roles of the people analysing the problem domain. He associates each viewpoint with a set of perspectives as a basis of uncovering discrepancies between the perspectives. Perspectives associated with

each viewpoint are integrated into a view, which is subsequently compared with other viewpoints for omissions and conflicts. Leite's viewpoint approach is concerned with very early validation in the requirements elicitation process. His viewpoints are actors in the universe of discourse or people with specialist knowledge about the system to be analysed. VWPL is not a requirements language, its main objective is to register early results of the fact elicitation process in the requirements exercise.

VORD is a service oriented viewpoint approach to requirements that adopts a two layered approach to analysis. The first layer, the viewpoint layer, is concerned with identifying user classes, and capturing and structuring their requirements. The second layer, the system layer, reconciles and interprets user requirements into a cohesive object-oriented framework. The system layer is seen as a transition to object-oriented design.

VORD defines a viewpoint as an external entity that interacts with the proposed system, but that can also exist without the presence of the system. Viewpoints in VORD are not just end-users and may include other systems and sub-systems in the environment of the proposed system. The authors argue that the service oriented view adopted by VORD provides a framework for integrating functional and non-functional requirements, integrating formal and informal specifications, and forming a basis for service reuse.

A limitation of VORD may be its weakness in addressing problems associated with systems that do not fit neatly into the service-oriented systems (SOS) paradigm. Service oriented systems can be viewed as service providing enterprises. They employ systems composed of people, computer hardware and software, and other mechanisms to perform service actions in the customer environment (Greenspan and Feblowitz, 1993).

Other than VORD, none of the viewpoint approaches so far mentioned provides an effective framework for integrating functional and non-functional requirements. In SRD and SADT they do not form part of the viewpoint analysis. CORE and VOSE analyse non-functional requirements as separate viewpoints — in the case of CORE after the functional model is completed.

As opposed to SRD and SADT, all the other approaches mentioned have viewpoints that assume a role/responsibility focus. The CORE viewpoint is an entity in the system environment that 'does something'. CORE viewpoints are associated with the roles of entities in the 'real world' or functional in the system environment. A CORE viewpoint can be seen as an agent whose role is typified by its interaction with the 'target system' whether directly or indirectly. A CORE viewpoint is also the source of domain decomposition. VOSE viewpoints are characterised by the roles and responsibilities of the participants in the software development process. Because viewpoints encapsulate only that aspect of the domain relevant to the participant, they provide a mechanism for recognising and incorporating the distributed nature of the knowledge and participants in the development process. Leite viewpoints reflect the roles and responsibilities of participants in the requirements elicitation process. They can be seen as providing a

vehicle for resolving inconsistencies in the requirements elicitation exercise. In VORD the interaction between the system and the viewpoint defines the role of the viewpoint in the system environment. Thus, for example, a bank manager withdrawing cash from the ATM is viewed by the system as a customer. Services in VORD are associated with end-users who have the ability to privately impose constraints on received services. In the van Lamsweerde, Fickas, and Dardenne approach, agents can be seen in the context of viewpoints. They are responsible for guaranteeing that system goals are achieved which in turn determine their roles. Assignment of responsibilities is done according to a set of heuristics to prevent agents' private goals conflicting with its responsibilities. Another set of heuristics is used to assign actions to agents according to their abilities, motivation and reliability.

Methods, Requirements and CSCW

In this Part of the Deliverable we have reviewed a selection of the vast array of different methods and techniques to understand what needs to be built. The sheer number and variety of methods are in themselves a testament to the needs of engineers when building large highly complex systems. We have considered these techniques in terms of the degree to which they are oriented towards understanding users and supporting the problem of developing a software system.

A fundamental problem in assessing or evaluating methods is that each has evolved to tackle a particular set of problems and meet the needs of a particular concern. Thus sociotechnical systems development is interested in the overall effect of an IT system on the organisation within which it is placed. In contrast, object-oriented requirements techniques are interested in recording maintaining and keeping consistent the needs of a constructed piece of software. Clearly, a large CSCW system will exist within an organisation which it alters and changes. Simultaneously the same system will contain a complex selection of software. One could imagine in this situation the use of techniques derived from these different traditions being used in harness.

The complexity of organisations and CSCW systems strongly suggest that it is naïve to consider a single method which is capable of meeting all of the demands of system construction. Many of the concerns are distinct and disparate while others are closely related. For example, the concerns associated with using a database to support police work can be divorced from the detailed concerns of which particular flavour of a language is used to realise the system. However, if a choice results in a dramatic alteration in implementation cost which causes fewer terminals to be purchased this will effect the utility of the system in practice. It is not important that a single method records all of the associated requirements consistently or completely, but that the different perspectives on the system are open to review and the relationships between them open to examination. Recording these viewpoints and the relationships between them enables the consequences of their interactions to inform the systems development process.

However, such a placing together of the requirements which emerge is difficult to achieve. Many current methods adopt a particularly committed theoretical stance. The value of the method is bound up in the degree to which a user of the method accepts the basic theory which underpins it. This is particularly true in the case of methods which address the needs of organisations and this mirrors the findings of Strand 1 in considering the modelling of organisational context. Methods are massively variable and heterogeneous in nature. This heterogeneity is manifest in terms of:-

- The origins and presuppositions of the method
- The particular problems they address
- The extent to which they focus on detail
- The objective of the method
- The community the method is intended to serve

It is important that this heterogeneity of approach exists within CSCW systems development and that techniques are provided which allow it to be managed. The question of heterogeneity and the associated concerns it raised are dealt with in Part IV of this deliverable and motivate much of the next stages envisaged for the project.

Of particular emphasis to our work and one which we wish to now turn to is the extent to which social methods of investigation can support the development of requirements for CSCW systems. In this section we have considered a plethora of techniques and methods, many of which make claims to representing the social and organisational nature of work. However, many of these techniques embody a particular interpretation of what constitutes the social nature of work. Many of these concepts have themselves to been open to debate and argument. Again it is worth highlighting the various positions which have emerged in gaining insight to organisations reviewed in Deliverable 1.1.

We wish to focus in particular on a consideration of techniques which provide insight into the nature of work in practice rather than an abstracted and overly theoretical rendering of what constitutes an organisation. Thus the question becomes not what is work or an organisation but rather what constitutes work and organisation as it emerges from the concerns of users of a system. An important technique which provides this complementary insight to the strengths of those studied here is that of ethnography.

Many prominent studies of work in CSCW (Heath and Luff, 1992; Harper, 1992; Hughes *et al,* 1992) have made use of a qualitative method of data collection in social research known as ethnography. For sociologists, the tradition of ethnographic enquiry is well-established. Ethnography is an observational technique which uses a naturalistic perspective. That is, it seeks to understand settings and their activities as they naturally occur, rather than in artificial or experimental conditions, and usually involves quite lengthy periods of time at the study site.

However, where ethnographers have historically concerned themselves with issues which might be described as quintessentially 'sociological', more recently they have been called upon as a means of informing system design and change-management processes within organisations. Such studies have included investigations of domains such as Air Traffic Control; Police work; Software Engineers; Architects; the International Monetary Fund; Stock Exchange dealing rooms; and a London Underground control room, among others. The rationale for this is that such descriptions not only provide a base line, but also a detailed understanding of work practices as currently constituted in such a way as to provide resources for understanding change.

As a research method, ethnography attempts to understand work and organisations from the standpoint of the participants, explicating ways in which they perceive and manage their roles and responsibilities and interactions with their colleagues. Organisations and occupational conduct are considered as embedded within, and constituted by social interaction, and the objective of such enquiry is to gain systematic knowledge of this institutional life as it emerges within the accomplishment of the day-to-day tasks and responsibilities by the participants themselves. The organisational culture provides and exhibits routine practices, collective representations, definitions and understandings concerning the nature of work, the everyday ways in which participants understand and manage tasks in their working organisational life. Ethnography thus seeks to bring out how and in what ways organisational life is complex and variegated.

In the following Part of the Deliverable we consider the relationship between ethnographic analysis and systems development. The intent of this examination is to uncover the issues which emerge when these different traditions are combined. Many of these issues form the research agenda for developing techniques to support the construction of CSCW systems and outline future work for the forthcoming years.

Part III
A Social Point of View for Design

Part III: A Social Point of View for Design

The aim of this Part of the Deliverable is to provide some important background to the longer term objectives of Strand 2 as set out in the Technical Annexe. More specifically, we intend to develop the lineaments of a social point of view for design which can contribute to inter-disciplinary working in CSCW. This will extend some of the discussion of Part I, though in a more positive and substantive manner.

In Part I we discussed, *inter alia,* a number of issues arising from the association of CSCW design with social science, particularly with sociology, emphasising some of the less welcome effects of this. Here we want to begin an examination of a sociological approach for design which is built around the ethnographic study of work and its social organisation.

As part of this exposition we will make use of analyses of studies of the design process itself:

- the social organisation of design project work
- interactional processes in requirements capture

We use these to provide an example of the way in which a social point of view on the understanding of the organisation of work and activities can be exploited, and through this, also focus on a substantive area which is of prime importance in Strand 2, namely, collaborative support for system design itself.

The first study is part of an ongoing project into the design process and the material here is based on field work done by Val King, Lancaster University and Wes Sharrock, Manchester University at two software engineering sites in the UK. The second study is an illustration of how requirements can emerge during the use of a prototype system known as Designer's Note Pad, DNP. This latter study is included as an appendix.

We begin with a discussion of the rationale for ethnography in CSCW.

The Preference for Ethnography in CSCW Work

More by accident than by design, one of the major sociological contributions to CSCW is the ethnographic investigation of the social organisation of work domains. Though not necessarily originally inspired by CSCW concerns, such studies include photocopier use (Suchman, 1987), office work (Suchman and Wynn, 1984), the police use of IT (Ackroyd *et al*, 1992), air traffic controllers (Harper and Hughes, 1992), a London Underground control room (Heath and Luff, 1992) among others. What is distinctive about such studies is their use of ethnographic field work to study the 'real world' character of work activities. This has not only brought ethnography into prominence in CSCW, it has also raised important research questions about how and in what ways such studies can make a

contribution to CSCW requirements elicitation specifically, and system design more generally.

The interest in ethnography as a research strategy in CSCW system design is doubly motivated: first, by dissatisfactions with traditional methods of 'requirements capture' and, second, to develop much more appropriate methods of analysing work activities and their settings as a means of informing system design and making end-use a much more prominent feature in CSCW design. It also proposes a new kind of end-user who is a person within a 'real world' setting and not some abstract homunculi of cognitive theory.

In this Part we want to address some of the issues regarding the role of ethnography as an exploitation of a social point of view in system design by way of preparation for the future research in this Strand. In keeping with the pragmatic attitude we outlined in Part I, our aim is to situate ethnography within a thorough appraisal of the other methods of system design and development in order to see better what it can and cannot deliver.

The Rationale of Ethnography

The origins of ethnography lie mainly in anthropology though it became part of the sociological tradition through the research activities of members of the Sociology Department at the University of Chicago in the 1920s and 1930s. It became closely associated with Symbolic Interactionism and, latterly, with Ethnomethodology.[35]

In our earlier discussion of some of the consequences of the incorporation of sociology into system design, we had occasion to note that much of the debate is highly misleading in the ways in which the divisions and debates among sociologists are presented. Hirschheim and Klein, for example, portray them as mainly theoretical and substantive; as, essentially, disagreements about the actual nature of social life as whether, to use one of their examples, it is essentially harmonious or conflictual. However, the division between sociologies is not necessarily exclusively, or even primarily, theoretical but is, in some cases, predominantly methodological. This is particularly the case in respect of the place of ethnography in social research.

One of the main tensions within the discipline of sociology is between those who regard it as overwhelmingly a theoretical venture and those who regard it as a more observational and investigative pursuit. Something of the flavour of what this tension involves can be gained from the two contrasting statements which, though by no means recent, do capture what are continuing issues effectively.

The prevailing view about the nature of the sociological enterprise was set out by Zetterberg (1965) who argued that empirical research was about testing theory. Much of sociological theory up to that time, and this is still much the case, had little

35 It is particularly closely associated with the former. Its association with Ethnomethodology, however, is more tenuous since this branch of sociology espouses no especial research method in the ways in which this is commonly understood in social research. For further discussion of ethnography see Hughes, King, Randall and Sharrock (1993).

or no systematic relationship to empirical research. There was no shortage of grand theoretical statements within the discipline, but little by way of research which was responsive to theory or which could test which of the many theories on offer was the more adequate. Zetterberg's argument, and it is a standard one, was that sociological theories, supposedly like their counterparts in the natural sciences, should be deductive in form so that low level hypotheses could be formulated for empirical testing. Methods of research and data collection were to be devoted to testing the predictions deductively derived from the theories.

Glaser and Strauss (1967), however, in a counter statement challenged this conception by arguing that Zetterberg's recommendations would simply perpetuate the problems it was intended to resolve, especially if the theory was created in an *a priori* fashion without an awareness of the nature of the phenomena it was supposed to cover. Glaser and Strauss' proposal for 'grounded theory', as they referred to it, was meant to promote the idea of developing sociological theory in conjunction with sociological research; that is, developing theory on the basis of data collected during the research and collecting data in such a way that theory could be so derived.

This debate became part of what is a long standing issue in sociology between 'quantitative' and 'qualitative' methods. The view of theory as deductive/predictive was derived from primarily philosophical preconceptions of the nature of scientific explanation which was, at least in the more advanced of the natural sciences and, therefore, paradigmatic of scientific explanation, quantitative in character.[36] However, the view of such as Glaser and Strauss is that quantitative research instruments in social research are often constructed with little or no knowledge of the phenomena they are intended to characterise and their use often represents, it can be argued, the arbitrary imposition of a 'grid' on phenomena which are not adequately understood in the first place. Such a process is not measurement or quantification as understood in natural science. Accordingly, at this stage of sociology's empirical understanding, the realistic objective must be an improved, a better informed, qualitative grasp of the phenomenon of study, such as can be gained only by relatively close and relatively protracted first hand observation of the day-to-day activities which are the focus of sociological theorising.

The above is essentially the nub of the argument within social research for ethnography. Glaser and Strauss's 'grounded theory' was not the first, or the only, argument against *a priori* theorising and against inappropriately — indeed, spuriously quantitative research, though it was a prominent and influential statement of such resistance. Similarly motivated approaches, such as symbolic interactionism and ethnomethodology, gave renewed impetus to the making of observational investigation of day-to-day conduct in various spheres of social life.

---

36 No less a figure than Kuhn (1970) has argued that this represents a serious misunderstanding about the development of even the most quantitative of the natural sciences. These did not begin with quantification. This only developed on the basis of a rich qualitative understanding of the phenomena being measured. The attempt to set and measure without a prior qualitative understanding was, he argued, bound to prove a failure, citing the contemporary social sciences as an example of the consequences of the premature efforts at quantification.

The practice of making prolonged study of day-to-day conduct within a social setting was, as indicated, the very stuff of social and cultural anthropology, and it is the anthropological term ‘ethnography’ which has been appropriated to describe all kinds of qualitative studies. Some (Button and King, 1992) have reservations about the utility of this term, particularly with respect to the way in which it makes it seem that the essential focus is upon data collection, rather than upon the analysis of such data. Nonetheless, within the world of system development the term ‘ethnography’ has come very much to mean first hand observational study of day to day activities, including the use of video and audio recordings.[37]

Ethnography and System Design

There are a number of developments within system design which have drawn attention to ethnography, some of which we have already discussed. Many of these have to do with the dissatisfactions expressed toward traditional requirements methods and, in particular, the ways in which many of these incorporate preconceptions about ‘users’ and their behaviour that are, to say the least, unsatisfactory. In addition, technological developments in respect of the computer meant that design could be more flexible and that the designer’s decisions could be driven less and less by what the technology would allow and more and more by what users wanted or needed. To adapt the words of DeGrace and Stahl (1990), systems could be developed which moved the computer from the centre of the system and replaced it by human beings.

Given that this new attention, especially within CSCW, required a thorough investigation of the socially organised activities of work and other settings in which systems would play their part, the turn toward ethnography as a means of furnishing such understanding is not surprising. Studies such as those mentioned earlier have played a role in sensitising designers to the complexities and the intricacies of the activities their systems are designed to service. Of course, they are not a unique source of such sensitivity for, as we have seen, there are many aspects of system design which have increasingly drawn designers’ attentions to the need to understand the organisation of the activities the system is to support. Suchman’s (1987) study of the actual use of a photocopier, for example, stimulated designers to reconsider the design of such machines as posing problems of communication with the user, or re-examining the ways in which the design was implemented in the machine and made available to users.

In addition, and perhaps of more importance, is the inherent connection between the ethnographic approach and the social point of view which differs from, and sometimes conflicts with, the ‘cognitive assumptions’ which have tended to

37 It needs to be pointed out that ethnography is not a single, unitary beast. There are, one might say, ethnographies and there are considerable debates within and between them over the nature of the enterprise, for example, over ethnography's alleged naturalistic orientation, the relation between theory and practice, and more. See Hammersley and Atkinson (1983).

dominate in the system design field.[38] Once again, Suchman's study was an important landmark in making designer's aware that there is an alternative to the 'cognitive model'; an alternative which emphasised the 'contingent' and 'emergent' character of courses of activity. It also emphasised the complexities of such courses of action and the structures which putatively planned them.

This emphasis on 'contingency' and 'emergence' carried implications, as we have also indicated earlier, for the determination of requirements. The idea of 'capture' suggests that these exist in a predetermined state, given in advance of an effort to identify them, whereas, and as we will illustrate later, requirements are very much negotiated and emergent features of the design process. Moreover, the inherent connection between the ethnographic approach and the social point of view means that the 'environment' of systems is, from the outset, considered in its socially organised terms. This further means that the examination of the role or the impact of the system is necessarily wider than the cognitive operations and immediate tasks of the individual user. In contrast, it requires that attention be paid to the collective and collaborative practices which sustain the organisation of work more generally, rather than just the specific conditions for the carrying out of some particular task.[39]

Although ethnography is used in various aspects of the COMIC project, and is the particular focus of much of the research of this Strand, it is not excepted from the warning against panaceas. Ethnography is not the Holy Grail. It is not *the* answer to the problems of requirements capture in software engineering (Sommerville *et al*, 1993). It does not even necessarily replace other methods of requirements specification and may be used on conjunction with them. Goguen's (1993) notion of 'zooming' and of using ethnography strategically is certainly relevant here. Ethnography is, after all, time consuming and, accordingly, an expensive procedure, and one which is in important respects an unfocused and open-ended one relative to most, if not all, methods of requirements elicitation. It is typically 'data driven' in that its objective is, characteristically, to observe what happens, *whatever that is*. The point of ethnography, as we have discussed, is to resist making *a priori* assumptions about how things in the setting must be related.

Another potential problem for ethnography in system design is, as Schmidt (1993) points out, its 'functionalist' inclinations which may be inimical to design thinking. The ethnographer's achievement is characteristically designed to be 'appreciative' of the lives it studies, to emphasise the skill and the rationality with which activities are conducted, and to exhibit the 'unperceived' functionality with which those activities are interrelated. Showing that a pattern of activities has the intricacy, balance and reciprocal interdependence that is found among the parts of a watch, to use this analogy, is to have accomplished an impressive ethnography. To do this, of course, in various ways militates against the idea of design intervention,

---

38 Although as we note earlier, the utility of such assumptions have been questioned.

39 Interestingly, there is a convergence between some approaches derived from cognitive science, such as 'disributed cognition' (Hutchins, 1990; Resnick *et al*, 1991) and what we have referred to as the social point of view.

an intervention which may disrupt by reorganising a well-established pattern of activities.

There are, of course, other questions which arise regarding the role of ethnography in CSCW, some of which will form the basis of further research in this Strand.

Incorporating Ethnographic Information in The Design Process

Among the important research questions about ethnography are those concerned with how ethnographically derived information is to be included in the design process, and at what point. As noted previously, ethnographic work has been important in affecting and enhancing designers' awareness of the character of the activities for which they are designing. However, the qualitative and relatively unfocused character of ethnographic materials makes their interjection into requirements specification problematic.

Another important point to make is that it is possibly misleading to think of ethnography in terms of 'data' or the 'materials' it generates. The 'materials' generated by ethnography consist mainly of field notes, audio and video recordings, together with other displays of the settings life.[40] These materials are used to provide vivid exhibitions of the activities which generated them. But data in this sense is not the critical thing which ethnographic work generates. It is the ethnographer's understanding of the setting from within which the exhibits are extracted which is crucial. It is therefore the case that the use of ethnography means a process of interaction between the ethnographer and the design team, one in which, in important respects, the latter debrief the former (see, for example, Bentley *et al,* 1992, Sommerville *et al,* 1992).

Ethnography in Design Collaboration

There are at least two models of the way in which the collaboration between system designers and social scientists may go.

- abolishing the distinction between designers and social scientists by requiring that each of them acquire a competence in the other's field:
- preserving the division of labour between the two 'sides' of the process by acknowledging their distinct and specialised competencies within a consultative context

The first of these seems both unnecessary and, above all, impractical. It undervalues the competencies of the respective parties as well as the prolonged training which is a precondition of such competencies. It also makes major presuppositions about the respective motivations of the 'two sides' to make the necessary effort to become multi-competent. System developers may have, of necessity, to acquire an interest in the domain for which they are designing, but this can be no more than functional for the purposes of system design. As Bansler and Bødker (1993) found in their study of the use of Yourdon Methods, the necessity

40 See COMIC-LANCS-2-4 (Hughes *et al*, 1993) for some discussion of these.

to take account of users is one which may be recognised as a necessity, but it is not necessarily one which is enthusiastically embraced. The designers thought of themselves primarily as system builders, and their concern was with the computational aspects of their task. Similarly, in the case described by Button and Sharrock (forthcoming), again concerned with the introduction of Yourdon methodology and CASE tools, the developers claimed that their failure to use the tools was not due to inadequacies of the tools themselves but, rather, to the fact that they were prevented from using them. Exploring their potential and acquiring mastery of them could not be combined with the actual doing of the project's work. The demand to get the job done and the project completed dominated that of mastering and implementing the tools. In other words, there is a strong specificity to professional interests and skills.

Reciprocally, of course, there appears to be no particular way in which the acquisition of an intimate acquaintance with the highly technical aspects of system design should be expected to inform the ethnographer's work or enhance its execution. This is not to argue against the need for some awareness of the system design process, its purposes and its problems, but this is far from requiring a shared competence between two 'sides' of the design team. Accordingly, for us the only realistic model to follow is the second, though this is not without its problems. The essential need is, of course, to ensure that intelligible, mutually relevant communication takes place between designers and ethnographers (Sommerville *et al*, 1992).

The use of ethnography is without neither difficulties nor drawbacks, but it does have various utilities, several of which are, in the short term, valuable correctives to the deficiencies of other methods, providing at least a contrasting and balancing emphasis to their 'one sided' inclinations. The appeal of 'process models', for example, is that their construction involves the kind of skills of which computer scientists, software engineers and system developers have a specific mastery, particularly those of modelling and programming. The construction of such models is an essentially logical exercise, and it is, therefore, almost inevitable that the conception of software process models will be of generic and abstract character. The problems which arise are, in many respects, the same as those which arise in respect of the Waterfall Model, itself a species of process model.

The Waterfall Model assuredly identified the principal logical elements in any development process. If one is building a system then, however in practice one goes about it, its development will involve, in some ways and in some combinations, designing, implementing, unit testing, system testing, and so forth. Not only are these logical components of design and development, but if one is considering modelling them abstractly, then they would appear to assume a natural logical order: one must design before one can implement, and one must have something implemented before one can test it. The Waterfall Model provides a logical and orderly presentation of the constituents of design and development work. The shape of 'process models' is, therefore, dictated more by the potentialities of software development techniques than by an understanding of the

'real world' practical requirements and 'domain' variations through which software is actually developed. Ethnography has its emphasis upon understanding 'just how' activities get done, and upon the local and specifically 'situated' character of activities, and of the practices for managing their co-ordination. It counterbalances the abstract and generic disposition of process modelling with an insistence upon its embedded and particularised nature, not least in respect to the requirements — in terms of skill, experience, domain expertise, for example — that are necessary to the interpretation and realisation of abstract, general procedures in particular cases. Ethnographic studies have provided one effective means of highlighting the considerable difference, and often discrepancy, between the logical and the practical order of an activity.

Representing the Character of the Phenomena

The features of ethnography which we have just mentioned, such as its protracted, unfocused and 'data driven' character, are at once limitations and strengths. At best, ethnography will have a limited use in the process of requirements elicitation and design. But, certainly at the present time, it can be of immense assistance in highlighting the problems and the limitations of those more focused, problem-driven methods and frameworks which are used in an attempt to understand the organisation of activities within relevant domains. Ethnographic descriptions are capable of displaying the extent and the nature of those discrepancies which obtain between those *a priori* grids which are imposed upon data and phenomena, and the realities which those grids are intended to characterise.

The problem with such grids is not that they are selective and simplifying but that their structure is typically determined not by the need for understanding the phenomena to which they can be applied but by the requirements of information display and processing. This problem is a familiar one in design, namely, letting things be driven by the technology even though it may be inadequate to the task. Thus, process modelling is often governed by considerations of what can be programmed and by the resources of formal modelling. Such procedures are, of course, well known to system developers, indeed, better known than are the phenomena that are being modelled.[41] It is in this way that such representational grids become impositions upon the phenomena that needs to be understood. It is not that these modes of representation are inherently ineffective, but because they are formulated in terms which define, in an *a priori* way, what it is to understand something, and a definition which is dissociated from any actual familiarity and detailed knowledge of what needs to be understood. This, of course, was the focus of Glaser and Strauss' complaint against much of theorising in sociology.

It is in the reversal of such approaches that ethnography offers the most utility for design. It seeks, first of all, a familiarity and detailed knowledge of the activities to be understood and then confronts the problem of how this understanding is to be represented. It may be that formal representations can be constructed adequate for

41 This is the burden of Bucciarelli's (1988) complaint about the engineering mind set as discussed in Part I.

this purpose, even where there are limitations imposed by the requirements of information management, display and processing. But at least there will be a greater awareness of the respects in which such formal representations fall short and how they may be compensated for.

The Social Organisation of Design

In this section we want to present an account of two studies of the design process, both of which take a social approach to the design process and used, within the broad remit in which it is understood in system design, ethnography. One is based on studies done of the industrial context of design where the software engineers were involved in relatively large projects; one a firm concerned with the design and development of photocopier equipment, the other with the production of software avionics. What we have done in the exposition of this material is focus upon two themes: the 'tasks and troubles of design projects' and 'projects and plans' within the overall thematic of Design as Workaday.

The second review illustrates how requirements elicitation can emerge through interaction between a developer and potential users working with an early prototype of the system. This is a much smaller system that those reviewed in the former studies.

The exposition is intended to illuminate both the ethnographic study of work activities and, by its attention to design as work, lay the foundations for the future work of this Strand.

We have earlier pointed out the plurality of approaches in sociology which could take the study of design in a number of directions. Some of these were canvassed in Hirschheim and Klein's schematic discussed earlier. Another is toward the study of designers-at-work in 'real worldly' conditions of industrial design and development. Such an approach provides some compensation for the 'individual' and 'cognitive' emphasis which has been a dominant influence of much thinking about design. Instead, it draws attention to the extent to which system design is a collective, collaborative enterprise and should provide an opportunity for understanding design methodologies in the context of the problems that designers confront in their day-to-day work.

System Design, Collaboration and Method

The design of systems is itself a prime example of computer supported cooperative work which is, these days and for the reasons we reviewed in Part I, increasingly supported by methods and tools, most of which have come in for no little criticism. Much of the burden of these criticisms is that designers and developers do not use the methods and tools in the ways prescribed or find them inadequate to the tasks they have to do within the 'real world' circumstances of design. However, despite the inadequacies of the methodology, despite the inadequacies of the tools, developers get the work done, and it becomes of some interest to see how this is achieved.

For example, Bansler and Bødker (1993) report in their study of the use of 'structured analysis' that,

> "...when we look at the way designers carry out their work, we observe four major deviations from the prescribed procedures: (1) the designers do not use data flow diagrams when communicating with users, (2) their diagrams do not comprise manual procedures, (3) the designers limit the number of models they construct to one (a model of the new system), and (4) they always supplement the model with other types of descriptions, diagrams or prototypes".

Designers and their organisations have "very pragmatic attitudes" toward the use of methods. The tools of structured analysis were used but were "combined freely with other tools" as they saw fit. What they did not do is comply with the "rules and procedures of Structured Analysis".

MacLaren *et al* (1993) have neatly formulated the orientation to the use of methodologies on the part of the experienced designer as one which is,

> "...highly pragmatic, using different methods as and when it suits them and their problems. They tend to use a given method only partially rather than 'by the book'."

They also emphasise that methodology is sometimes a substitute for experience but not an alternative to it. They identify, in effect, the phenomenon of the 'old hand' who is less inclined than the 'novice' to do things 'by the book'. 'Old hands' are, of course, wise, even cynical, in the ways of their working environments, and are often sceptical of the merits and necessities of adhering to prescribed ways, and more than capable of improvising on and around the official procedures. The authors further suggest that this is because,

> "...analysts tend to operate under a number of constraints such as time, human resources and money, they become good at trading-off and can be seen as 'pragmatists' and 'satisficers' rather than 'engineers' or 'rational optimisers'"

This last observation bears upon a major issue in the discussion of methodologies which concerns their core rationale namely, to provide a *rational* procedure to follow in order to achieve some predefined end. As we have repeatedly pointed out, one of the major justifications for design methodologies is to provide a systematic organisation to the manifold tasks and activities that system engineering involves. For Parnas and Clements (1986) achieving this is a chronic problem because though it is not possible to develop a fully 'rational' procedure for software design and development, it is necessary to 'fake it', that is, proceed *as if* such procedures can be formulated and followed. They argue,

> "...a perfectly rational person is one who always has a good reason for what he does. Each step taken can be shown to be the best way to get to a well-defined goal. Most of us like to think of ourselves as rationally professional. However, to many observers, the usual process of designing software appears quite irrational. Programmers start without a clear statement of desired behaviour and implementation constraints. They make a long sequence of design decisions with no clear statement of why they do the things the way they do. Their rationale is rarely explained. Many of us are not satisfied with such a design process. That is why there is research in software design, programming methods, structured programming, and related topics. Ideally, we would like to derive our program from a statement of requirements in the same sense that theorems are derived from axioms in a published proof. All of the methodologies

that can be considered 'top down' are the result of a desire to have a rational, systemic way of designing software".

Parnas and Clements while indicating their own preference for this ideal recognise that it is and must remain an idealisation. Software development cannot proceed as a 'logical derivation' from a statement of requirements because:

- the requirements cannot be formulated completely or unambiguously
- the requirements do not exhaust the knowledge of the domain that is necessary for successful system design; just what is relevant will only be learned as the design process progresses;
- the informational requirements for designing and building a system are too great to be practically manageable;
- people make errors and these cannot be eliminated;
- the design process is informed by designer's preconceptions, prejudices and enthusiasms;
- the 'economising' requirements on software development work pre-empts 'rational' procedures.

Despite these 'shortcomings' of design relative to the ideal of a 'rational' process Parnas and Clements maintain that the production of idealised 'rational' versions of the process are desirable because:

- designers need guidance
- even though it is not thoroughgoing, the rational procedure introduces systematicity into the order of work
- the rational procedure provides a standardised format which facilitates movement between projects
- progress according to a standard procedure facilitates appraisal of that progress.

Rationality and Method

The trouble with such a conceptualisation of the situation is with the conception of 'rationality' which it employs. The conception of a 'rational decision maker' in terms such as these is typically an idealised construction and as the history of models of 'rational decision making' (March, 1991) shows, there has been a steady retraction of the idealising assumptions built into such models.

The 'rational decision maker' is typically constructed through the incorporation of assumptions which simplify the environment, such that information has no costs, that the decision-maker has perfect information, etc., and which make obviously counter-factual assumptions about persons, events or processes. As Garfinkel (1967) draws attention to, such constructions are akin to being 'in a game' which involves, by definition, the suspension of the presuppositions and procedures of 'serious' life. Such constructions involve what are calculatedly counter-factual assumptions designed to provide the necessary simplification to make the problem of, for example in the classic instance, constructing a mathematical formulation of the game of poker a tractable one.

We have chosen the phrase 'calculatedly counterfactual' to indicate that the unrealistic, perhaps even false, character of the assumptions built into such models is something that their contrivers are well aware of and intend. In other words, the aim is to contrast the rational course of action specified in the model with the actual courses of action that transpire in the 'real world'. The construction of the model on the basis of such assumptions is justified on methodological grounds and the model itself treated as an analytical tool. This is certainly the sense in which Weber (1949), one of the founding figures of sociology, saw his 'ideal type' constructs; that is, as models designed to simplify complex empirical situations and, through this, highlight the divergence between the courses of action envisaged in the 'ideal' model and that occurring in empirically observable situations.

Roundabout as such a method of proceeding might seem, its use is intended to enable the structuring and focusing of, in Weber's case, historico-sociological inquiries using the divergence between the empirical case and the rational model as a resource for explanation.[42] The point was not to judge that the empirical case was irrational, for the model would be constructed in such a way as to say what, *other things being equal,* would be the most rational course of action. What, for example, other things being equal, would be the most rational way for the general to win this battle. If the study of the battle revealed that the general had not followed this course, then this would be an occasion to open up the *ceteris paribus* clause to determine which of the assumptions had not obtained. The general's course of action might then be shown to be entirely rational relative to the actual circumstances which obtained at the time.

Whether or not such a contrastive procedure for constructing concepts is either necessary or truly effective is not one which we can elaborate or evaluate here. We bring these issues to the fore to argue that a rational model almost certainly cannot be developed as a straightforward representation of what persons actually do. Indeed, it can be argued that it would be a highly irrational person who sought to implement the model of the rational actor in which, say, the actor did not have clearly defined and ranked preferences, in which information was less than perfectly available, without taking into account the cost of acquiring information, etc.. It would, in other words, be an irrational approach which treated design and development work as something which could be determined in advance down to its last detail, and which presumed that its course was exempt from contingency, confusion and error.

The implication is that Parnas and Clements are deploying an inappropriate notion of 'rationality' in their appraisals of the software development process. The practical utility of devices which are constructed on the basis of simplifying assumptions depends upon remembering that they will involve simplifications and their application to 'real world' situations will require adjustment for those simplifications. The use of a 'rational method', or any method for that matter,

42 Such models are, of course, typical of economics which exemplifies a highly abstract, deductive approach based on the development of models which are deliberately simplifying in order to make its problems tractable to logical and mathematical modelling. See Anderson *et al* (1989).

depends at the very least upon its user being alert to the extent to which the assumptions incorporated in the method depart from the realities of the situation to which it is applied.

The designer might as well be conceived as a strongly rational operator *in the circumstances in which he or she finds themselves,* capable of recognising the discrepancy between the 'ideal' character of many prescriptions and techniques and less than ideal circumstances under which they are applied. Aware, therefore, of the necessity to manage the application of those methods and techniques in the situation to hand. The skills upon which the working designer draws are not just those of technical design and development, but also those of 'negotiation' of relationships with others, manipulation of the organisation's ways, and so on: in effect "working the system" (Sharrock and Anderson, forthcoming).

Design in Organisations

The point about the inappropriate notion of 'rationality' is, in effect, a reminder that design and development takes place within a less than ideal world. This is, of course, a point we have been making, in various ways, throughout Part I of the Deliverable.

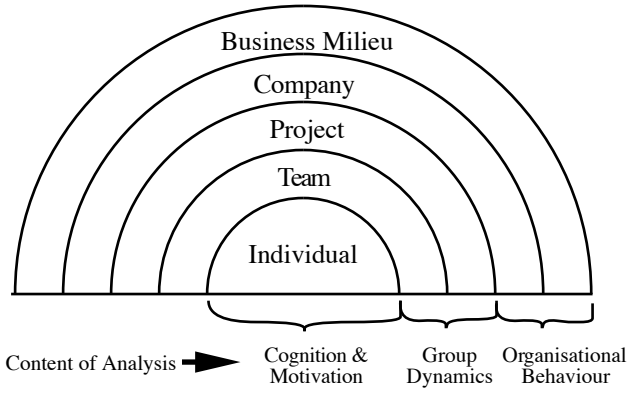

Figure 39 — Curtis *et al*'s Layered Behavioural Model of software development

It is also a theme taken up by Curtis *et al* (1988) in their proposal of a 'layered behavioural model' for design and development based upon a questionnaire study of the development process across a number of projects. The model focuses on the behaviour of those involved rather than looking at the "evolutionary behaviour of the artefact through its development stages". They propose 5 levels of analysis: individual, team, project, company and business milieu (figure 39). At the individual level software development is analysed as an intellectual task and subject to the effects of cognitive and motivational processes. At the team level these processes interact with the social. In larger projects, several teams have to integrate their activities and "*inter*team group dynamics are added on top of *intra*team group dynamics". Projects have to be aligned with the goals of the company and, as a consequence, are affected by corporate politics, cultures and procedures. Interaction of the company with other companies in the business milieu introduces

other external influences. It is these cumulative effects which are represented in the layered behavioural model and the size and the structure of the project determines how much influence each layer has on the development process.

It is in terms of this structure that they go on to analyse 3 themes that arise in system design, namely, the thin spread of domain knowledge, fluctuating and conflicting requirements and communication and co-ordination breakdowns.

Thin Domain Knowledge

Curtis *et al* acknowledge that knowledge of the domain for which a system is being designed is a requisite for effective design but that, typically, such information is spread thinly within design teams. At the *team* level there will be some designers, whom they term 'exceptional designers' and who are akin to 'project gurus', who have a clear and comprehensive vision of the project's purpose and the way in which its parts are to integrate. These designers, perhaps supported by a small coalition built around themselves, were capable of dominating decision making within at least the early phases of the design process. At the *project* level developers were found to be self-educating in the necessary domain knowledge, something which was not allowed for in the scheduling and costing of the project since it was presumed that the developers would be designing full-time. At the *company* level they found that software development expertise would migrate into management with the result that the value of technical knowledge would diminish as persons were removed from development work, and middle level management would become isolated from both the detailed project decision making and from that of corporate, higher level management. At the *business milieu* level, the operation of multi-company teams did not necessarily involve cooperative work. Clashes along lines of organisational separation took place along with efforts to shift the difficult problems elsewhere.

Fluctuations and Conflicts Among Requirements

Fluctuation and conflicts among requirements typically resulted from market factors such as differing needs among customers, the changing needs of a single customer, changes in underlying technologies or in competitor's products as well as from misunderstandings of the application domain. Problems could also emerge from internal company sources, such as marketing, product management, and so on. Further, the design team often negotiated to limit the requirements to those that could be implemented on schedule, and within budgetary and technical constraints. In fact, agreements were difficult to enforce across teams and programmers often added unrequired enhancements. The result was that the stability of requirements was illusory.

Communication and Co-ordination Problems

Extensive documentation did not reduce the need for communication. It was essential for clarifying issues, especially during the early phases of the project. Barriers to communication among project teams created a need for individuals to create informal communication networks which spanned team boundaries. Since no single group served as the sole source of requirements in either commercial or

governmental environments, organisational communications became crucial to managing projects.

The overall conclusion of the authors is that the development of large software systems at least is a learning, communication and negotiation process and that

> "...much early activity on a project involved learning about the application and its environment, as well as new hardware, new development tools and languages, and other emerging technologies. Software developers had to integrate knowledge from several domains before they could perform their jobs accurately. Further, as the project progressed they had to learn about design and implementation decisions being made on other parts of the system in order to ensure the integration of their components. Characteristically, customers also underwent a learning process as the project team explained the implications of their requirements. This learning process was a major source of requirements fluctuation..... There was a natural tension between getting requirements right and getting them stable". Curtis *et al* (1988).

Design as Workaday

Studies such as the above have the virtue of attempting to understand design as a 'real world' activity subject to all the constraints and contingencies that arise from this fact. However, they uneasily make distinctions between, for example, levels of analysis, which bear no sound warrant.

Naturally, they are also 'designer centric' in outlook and, as a result, there is the danger that they do not sufficiently stress the necessities of design and development within organisations. In this section we want to develop our own approach which can be characterised as a study of the design process as workaday

Obviously, the notion of designer, like that of design itself, is a complex one since the kinds of ventures in which they are involved and the kind of persons that designers are is extensive and variable. So much so that we must issue a permanent warning on the dangers of over-generalisation. As indicated earlier, much of what follows is based on 'designers' we have encountered in our studies.

In turning to a social point of view of design one of the problems is trying to get a view of design 'from the inside' as a socially organised activity and, in doing so, avoid many of the metaphysical issues which, as we have pointed out previously, infect the sociological study of design and technology as so much else in sociology.

One of the major sociological approaches used in understanding the social character of technology and design, is that of 'social constructionism' which, as we have indicated, often involves argument of a metaphysical kind focused primarily on epistemological issues. Its argument in this context is that design is not merely a technical matter from which social, political and cultural factors can be excluded. As many studies show, technological design is infused through and through with such factors.[43] An implication that is typically drawn from this is that 'system rationalists', as Kling (1980) calls them, that is, those who draw a firm line around technology to the exclusion of other considerations, are not only wrong but

43 See Part I for relevant literature.

seriously deluded for such a view denies the realities of design and the social circumstances of its production.

It is characteristic of 'social constructionist' views that all activities are *essentially* 'political' and that there is, in fact, no separation between 'technical' and 'social' issues'. To the extent to which this is ignored by software engineers, then not only is this dogmatism and arrogance, but it also contributes to the capacity of system design to control human lives rather than support them.

However, social constructionist views do not necessarily have to have an epistemological slant to them. Indeed, they were developed predominantly for descriptive sociological purposes. Berger and Luckman (1966), the original formulators of social constructionism, made a sharp distinction between sociological and epistemological problems and made the wry comment that the attempt to combine the two within their own venture would have involved attempting to push the bus in which they were riding.

The spirit in which we offer the following, therefore, is one of descriptive characterisation rather than attempting in any way to offer metaphysical or other judgements about the work of system design. We have already noted that there is considerable variability in outlook among engineers and designers and we do not doubt, therefore, that there are 'hard line' rationalists who do, indeed, hold their views in a metaphysical spirit in arrogant and dogmatic ways.[44] But these are not the working designers and developers that we have encountered. Although they may be characterised as 'system rationalist' in that they could fairly be said to believe that a line could be drawn between 'technical' and 'political' issues, and that the latter were 'irrational' and 'somebody else's business', such views could better be construed as recognitions of personal and professional limitations rather than as arrogant and dogmatic certainties about the general nature of technology and engineering.[45] The contrast between the 'technical' and the 'political' as implemented in the practices of engineers was one which had, prevailingly, to do with the 'things they liked to do', the 'things they were competent to do', the 'things they were entitled to do', and so on.

In treating actual, observable practice as the vehicle of a metaphysical world view can lead to the neglect of the extent to which practice has to do with persons making quite realistic judgements of their occupational competencies and the practical manageability of their situation. Anderson (unpub. manu.) has emphasised that 'the workaday' is very much the tone in which practical affairs of daily life, including those of work, are conducted.

In what follows we concentrate on two themes drawn from the study of large scale industrial system development: the tasks and troubles of design and projects and planning.

---

44 Making an elegant match for equally 'hard line' social constructionists!

45 It is important to note that the terms 'technical' and 'political' though they might often turn up as parties in a dualistic constrast do not invariably mark the same contrast.

Tasks and Troubles of Design Projects

One of the important reminders that emerges from the study of the workaday character of design is:

- It is very much a collective activity
- The considerations which affect the course and outcome of the design process are more numerous than those which are involved in the organisation of the operation itself. Many of these have to do with the relationship of the design work to the environment within which it is situated.[46]

As we have noted, the conditions of design decision making on projects are, very commonly, less than ideal. It is simply the case that relative to the ideals of design, procedures and techniques are subject to considerable constraints in terms of available arrangements and resources. The design tasks and problems of a project are not necessarily even the highest priorities among members of the design team itself. A given project is after all very often only a single event in longer term relationships, and those who are involved may well have to work together again, and the maintenance of good, or at least civil, relationships can have a higher priority than the immediate issues to hand. Although confrontations do occur this may often be only after attempts to avoid or contain them. There can be much give and take simply in order to avoid trouble.

The transfer of learning from one project to another is not given any very high priority. Each project is overwhelmingly preoccupied with its own problems and, in this respect, is in competition with other projects in the same organisation each seeking to maximise its resources and minimise its troubles.

Among the design teams we studied, the completion of projects did not result in extensive post-mortem lesson learning either. It was widely regarded as counterproductive to engage in the retrospective analysis of projects, not least because a good many of them were ‘failures’ or at least abandoned. Within design and development organisations it can be a matter of routine expectation (cf. Grudin, forthcoming) that many design projects will be aborted, and this is not a matter for either analysis or acrimony after they have been so. Though the notion of ‘learning from projects’ may be one which is ideally desirable, the conduct of ‘post-mortems’ on failed projects can be kept as minimal as possible, with staff being reassigned to other work as rapidly as possible and the project and its problem’s being ‘buried’ equally hastily. Designers did, of course, learn from experience, both over the course of a project and over the course of their careers, but it was a personalised cumulation of intimacy with domain phenomena, with characteristic and specific project risks and problems, and of ways of handling and resolving problems, rather than any strong collective cumulation of lessons from failures and difficulties.

---

46 It is worth noting that ‘the organisation’ is itself, in substantial part, the product of design and that the business of organisational design is not inherently more successful that is that of system design, a fact testified to by the steady succession of fashionable theories of organisation and management and the recurrent redesigning of organisations.

The business of project development is, however, very much a decentralised one. The emphasis is on getting on with the project's work and the requirements for knowledge and learning tied to the pragmatics of the project, since, of course, the capacity to study, collect information and provide training must compete with the other resources.

There is an 'in for a penny, in for a pound' feature to the resourcing of projects. There are tendencies to keep the budgets of projects as low as possible, leaving out of account items such as training and preparation costs, the support for distributed work, and restricting the prospect for iterations as much as possible. Relatively modest outlays which could facilitate the solution of problems will be avoided. Although this may well result in more substantial outlays subsequently, this is not, perhaps, quite so irrational as it looks. The budgeting of a project may be critical to getting it 'on the road' and keeping the estimates as low as possible may well facilitate this. Once the project is 'on the road' and its performance is at risk, then additional expenditure — having moved from being large relative to the economising motives of budgeting to being small relative to what has already been invested — is more readily justified.

The management of commitment to projects is also a problem for designers. The vigorous pursuit of a project's objectives and overcoming its difficulties, is something which can require members of the team to identify strongly with the project and its product. At the level of what Curtis *et al* (1988) term 'interteam dynamics', it is, of course, entirely possible that a given individual is a member of more than one team at a time, in which case teams are competitive for the individual's loyalty.

In one of the design teams studied, a system known as 'matrix management' was used, in which individuals are members of both functional departments and project teams and, therefore, members of potentially more than one team. The allocation of individuals to teams is, of course, made upon estimates of the needs of respective teams for that person's specialised skills. But such allocations can prove to be wrong, given the contingencies that always arise in the course of a project, both with respect to the workload required and to the scheduling of the work. Thus, it is common for an individual to be pressured to prioritise the demands of one project over another, and some project managers seek to build loyalty to them and their projects among team members.

The building of such team identification can, however, mean that the cancellation of a project is a major frustration for those involved, to the extent that some designers will be reluctant to join projects which have a high risk of cancellation. However, acceptance of risk of cancellation can be presented as a matter of professional maturity, of acceptance of the risk of cancellation as an inescapable fact of life in development work, and one which may originate from the exigencies of corporate fortunes as much as from deficiencies in the work of the project team or its product.

The other aspect of this is, however, that the designer's approach to their work may then be governed by their 'realistic' expectations about the likely completion of

the project. If there is an expectation that their current work will not particularly matter anyway, then team members may be reluctant to invest effort above and beyond the call of duty and unwilling to undertake work which has significance for developments 'downstream'. There were certainly strong features of this in the avionics team studied. Also, as Grudin (forthcoming) has argued with respect to the introduction of 'design capture' devices, the perceived utility of the data captured in them is for those who are further downstream rather than those who generate the data. For the latter, it may well be that they judge this a waste of time since they will never use it.

The setting up of projects is not itself something which is done with an eye to the provision of optimal conditions for design and development work. There is, for example, the concern of the organisation to get work. Finding itself without work is a serious situation for an organisation. In a competitive environment there may be a need to provide tenders which will be successful and to do so without necessary regard to the viability of the project with respect of cost, schedule or scale. For example, in one case a software house, seeking to gain repeat business from a previous contractor, had obtained a contract for further research to follow up two earlier projects. In order to encourage the client to invest further funds in a sequence of projects which had not, as yet, had any practical pay-off, the contract had to be financially and temporally stringent. In fact, the arrangement was such that the project team found at its initial meeting that the project was already *de facto* two months into a twelve month project.

Where the bringing in of work is the imperative, it is essential to ensure the contract is gained. But that the contract is a disadvantaging one, and that there may be serious problems in its fulfilment is recognised, and that while the concern is to secure the contract, these problems are not the dominant ones. These are problems which can be left for later and, most likely, for others. The problems of finding a way around the scheduling, financial and technical difficulties which arise from the contract are ones which may be passed onto to the design team, as ones to which they must seek to find a solution in the course of their work.

An illustration of this is found in Sharrock and Anderson's (forthcoming) description of how a project was given the task of reconciling a target 'unit manufacturing cost', one which was dictated by the price that could be realistically charged for the product in its intended market, and one which would be achievable only with a small production run. This problem was acknowledged from the start, and was a continual preoccupation for the project team who would continually look for ways of keeping costs down and, particularly, for achieving substantial reductions in average cost, though their economising efforts did nothing to make a significant dent in the discrepancy between the target cost and their projected unit manufacturing cost. Though the problem was an immensely serious one, and one to which there was no apparent solution, it was nonetheless not a problem that the team directly confronted. Rather, it was one for which there was a hope that some solution might turn up or, failing that, a dilemma in the project that their reviewers and managers might be compelled to confront and, somehow, resolve.

The determination to 'bring in' that project also involved bypassing many of the formal procedures for instituting a project. Many of these involved the explicit, documented assent of parties to the agreement, and bypassing them required informal agreement on the assumption that they would be given formal sanction retrospectively. Enough was done in terms of requirements to get the project on the road, it being assumed that detailed specifications could be accomplished by the time they were needed. However, the making of informal agreements was subject to the contingencies:

- important parties to agreement changed their position within the organisation;
- those who made the agreement had achieved only a vague understanding of what they were agreeing to, understandings which, when it came to specifics, could be variable;
- following through on what had been agreed would require more work than a party to it had bargained for.

Indeed, the detailing of the specifications on this project became fateful for solving the technical requirements of the project, but this could not be done at the point at which they were necessary. The developers simply had to take risks, making decisions about the specifications in order to fix their problems, hoping that their decisions would eventually be sanctioned.

Those involved in setting up and carrying out projects were, then, involved in taking risks. They were confronted with what they saw as the practical necessity to do so. The choice was between, for example, getting the project (and taking the risk of being unable to deliver on its terms) and not getting the project, where being without the work was the serious problem. Meeting the conditions of the contract could await the evil hour. Similarly, the design team had the choice between waiting for detailed specification information on a project, and thus delaying further work on a project where scheduling was, in any case, impossibly demanding, and making some unilateral decisions that would allow them to get on and provide a fighting chance of meeting their schedule.

Getting the immediate problem solved is the issue, and dealing with its consequences is something else, and, perhaps, somebody else's problem.

Curtis *et al* (1988), as we note above, discuss the way in which there were clashes within multi-company teams following attempts to shift different problems onto one another. This might imply that there was a political struggle of a nakedly self-interested sort amongst the elements drawn from the different companies. We do not know whether there was or not, but the willingness of persons to hand on serious problems to others in the cases we studied should not be understood in such terms. Though the project teams we studied were made up of heterogeneous assemblies of different skills, and of sub-groups of professional specialists, there was, within the teams, a general spirit of cooperation, of mutual appreciation and sympathy. Many of the individuals involved had worked for the company for many years, knew each other well and had friendly relationships. They were frequently willing to help each other out. However, they had, and recognised each other to

have, problems of their own. The organisation's objective, with respect to workload, was to ensure that its workers had their hands full: 'We expect you to arrive early and leave late' was a warning given to new appointees. There was a willingness (though not, of course, invariably) to see the other person's point of view and to sympathise with their difficulties. But this was not identical with a willingness to do anything to help with them, particularly if this meant shouldering problems which had not been allowed for and which were not unequivocally 'owned' by one group or another.

Thus, on a project involving cooperation with Japan, no one had made allowance for the fact that a large amount of technical data would need to be translated from the Japanese, and the Japanese branch of the company was entirely willing to collaborate with and make available the technical data, and help in interpreting it, but it did not see that it should take on the cost and trouble of translating it. Nothing had been agreed on this, and it was not, therefore, its problem. That one was actually or prospectively creating troubles for others was a feature of many decisions but, from the point of view of those taking them, this was unavoidable because they had enough troubles of their own.

Related to this feature is the distinction drawn between the 'technical' and 'other aspects of design, such as human factors. As previously mentioned, in their study of the users of structured analysis methods Bansler and Bødker (1993) note that the devices which were intended, as part of that methodology, particularly for communication with users were not very enthusiastically employed for that purpose by developers whose enthusiasm was invested in building computer systems. Our own observations agree with this. The distinction the designers' drew, for example, between 'technical' and other aspects of design was one which drew a line between those things in which they were interested, and which drew upon the skills that they liked to exercise, and those in which they would, at best, take a reluctant interest and in which they had no professional competence.

The distinction, furthermore, related, quite realistically, to their position within the hierarchical division of labour and authority within the organisation for which they worked, between those matters which were and those which were not 'any of their business'. Part of their work involved them in living with and attempting to clear up the mess that other people's decisions had made, decisions over which, from their point of view, might well be erroneous, gratuitous to and obstructive of the conduct of the affairs for which they were organisationally responsible. These were people who were typically interested in doing engineering tasks and although they might recognise the import of the social and political aspects of their work, they did so with little enthusiasm. 'Human factors' were matters which were routinely acknowledged, but regarded as a specialised expertise and, in one of the firms we studied, located in another department (which could exercise a veto over other decisions). Human factors was the business of the Human Factors Department. If human factors failed to send a representative to meetings, or dispatched a junior as their representative, then that was their problem.

This is not, we hasten to add, to suggest that engineers were mechanically dismissive of human factors considerations in design unless they were actively put by the relevant specialists, for they were aware of the gross ways in which 'human factors' could impinge upon their design and had to be allowed for. But attentiveness to the range of implications of human factors was, indeed, 'someone else's business'.[47]

Projects and Planning

An issue which has surfaced in recent discussions of formal methods in requirements and planning is a consequence of some problematic interpretations of Suchman's (1987) critique of the 'planning model' in cognitive science; a critique which is, perhaps, (mis)interpreted as saying that people do not follow plans which are, in reality, *post hoc* rationalisations of courses of action. The complexities of the argument between 'cognitive science' and 'situated action' are considerable and Suchman's argument is embedded in these. It is a critique which is focused upon the idea that there are *mental* plans which operate as determinants of subsequent sequences of action. The attack is on the notion that people have 'plans' in the sense that some cognitive scientists suppose they do and not upon the idea that people never follow plans in the sense in which we ordinarily observe them doing, for example, in the sense in which we observe American football teams calling out and following through on predetermined plans. Many of their plays are made according to plan and some of them unquestionably go according to this plan.

Suchman's own formulation of her argument sometimes provides hostages to fortune. The impression that she treats plans as *post hoc* reinterpretations is given by the citation of views attributed to G.H. Mead which distinguish between two kinds of action:

> "...an essentially situated and *ad hoc* improvisation — the part of us, so to speak, that actually acts. The other kind of activity is derived from the first, and includes our representations of action in the form of future plans and retrospective accounts. Plans and accounts are distinguished from action as such by the fact that, to represent our actions, we must in some way make an object of them. Consequently, our descriptions of our actions come always before or after the fact, in the form of imagined projections and recollected reconstructions" (Suchman, 1987)

Such a quotation is to be contrasted with her description of the way in which, for example, canoeists approach the taking of rapids, inspecting the course of the rapids and roughing out a plan for making their way through them.

> "In planning to run a series of rapids in a canoe one is very likely to sit for while above the falls and plan one's descent. The plan might go something like, "I'll get as far to the left as possible, try to make it between these two large rocks, then backferry hard to the right to make it around the next bunch. A great deal of deliberation, discussion, simulation, and reconstruction may go into such a plan. But, however detailed, the plan stops short of the actual business of getting your canoe through the falls. When it really comes down to the

47 See Sharrock and Anderson (forthcoming).

details of handling a canoe, you effectively abandon the plan and fall back on whatever embodied skills are available to you"

When taken in the conjunction with the rest of Suchman's argument, there is no reason why these remarks should lead to the interpretation that plans only or typically function as retrospective justifications. Nonetheless, they have given encouragement to responses such as that of Vera and Simon (1993)

'Lucy Suchman has focused on the issue of planning. Planning has traditionally played an important role in systems that interact with the environment. A large part of robotics research (at least into the '80's) involved improving robots' plans. Suchman takes the rather extreme position that plans play a role before and after action but not really during it. The action is carried out at its own independent level. Before actions, plans serve only an organisational or predictive function. Following action, plans serve as 'accounts of action taken' or 'reports of choices made'. There is no causal relation between the plans and actions performed by an intelligent system.'

However, Suchman (1987) also says that "plans are resources for situated action but do not in any strong sense determine its course" which means, as can be seen, the issue is not about whether people follow plans (or only appeal to them in *post hoc* rationalisation) but about what is involved in following out a plan; whether the relationship between plans and the actions which implement them is a simply a mechanical one. Plans may *sometimes* be used in retrospective justification of courses of action taken with reference to them, but there are also courses of action which follow plans. As Schmidt (1993) has written

"...for design purposes we need to investigate the relationship between plan and situated action in more detail and far more precisely. Plans do not determine the course of action in any absolute sense but different plans in different settings may determine the course of action differently. Conversely, contingencies may be more or less complex to deal with, more or less serious in terms of effect, scope etc., more or less frequent and so forth, and different contingencies may affect the outcome of action and the validity of plans differently. These are issues that can be determined empirically.'

The essence of Suchman's critique of the planning model is that it requires that activity be thoroughly determined in advanced and causally directed in its every detail by 'a plan'. Such a conception cannot be adequate to the phenomena it seeks to describe because it does not allow room for the fact that the plan is an abstract construction which will, at the very least, require articulation with, and application to, the specifics of circumstances in which it is to be followed. More will be involved in following the plan than is specified in it and, furthermore, the relationship between the plan and the action it purportedly directs cannot be causal. Such a critique may be effected by demonstrating that the requirements for carrying out a course of action in a situation exceeds what can be specified in a supposed 'plan' and this is what Suchman attempts.

However, the objections against the 'planning model' are not objections to the utility of plans in ordinary life, nor do they carry the implication that plans can never work. The construction of plans in actual practice do not typically involve the supposition that everything must be spelled out after the fashion of a purportedly scientific model with nothing left implicit. Plans in practical life are

characteristically ‘recipient designed’, to adopt a term from conversational analysis, such that they depend upon the possession of ‘good sense’ and ‘practical judgement’ on the part of those who are to follow them to make appropriate applications of them in actual circumstances. Nor does the making of plans indicate any expectation that the course of action specified in them will, of necessity, follow through. Plans often include ‘fail safe’ devices to cope with situations where things go badly awry, and arrangements to allow for the adaptation of the plan to exceptions, unforeseen circumstances — even its extensive revision — and so forth, as well as mechanisms to oversee the implementation of the plan and to enforce its requirements.

Those who undertake the planning of design development projects are not naïve with respect to the fact that courses of activity will not follow those which they specify in their plans. Projects, after all, involve variable degrees and kinds of risk as we have seen.

In the studies that we have made, the planners were, in completing a plan invariably leaving many matters to be decided. The planners depend upon their typified conceptions of how courses of action are carried out, or upon the presumed grasp that others on the project will have of how things are to be done. They are well aware that the efficacy of methods is itself variable, and that the ways in which those in the organisation work vary from the deployment of ‘sure fire’ techniques to the dependence upon techniques which are less than dependable. Even the more dependable techniques may require skill and experience in their application.

In working out the sequence of operations that the project is to follow, it is appreciated that the project may deviate from the plan because of unforeseen problems, unplanned for contingencies, etc. It was routinely accepted that though there were detectable areas of risk, there were also every prospect of problems arising which could not be foreseen. The employment of methods of ‘risk analysis’ might enable the identification of the most problematical phases of development and might permit, for example, their treatment at an early stage in the process. But ‘risk analysis’ is not supposed to provide a guaranteed way of recognising all the risks that a project runs. The very operation which one might suppose the least problematic might turn out to be the one that gave all the trouble.

In the copier firm, the work was organised on the basis of a problem classification scheme which categorised the problems currently afflicting the project, the main category of concern being the ‘critical’ problems, ones which were explicitly conceived as ‘life threatening’ to the project. At the time they were so classified it was recognised that there might prove to be no practicable solution to them and that their persistence would be such as to render the product uneconomic, inadequate etc. The insertion of milestones and reviews into the project is an explicit recognition that matters may by no means go as planned; that what was planned cannot be realistically implemented and that the project might require abortion.

The planning of a design and development project involves seeking to determine what will have to be done and, crucially, how long it will take to do whatever needs

to be done — by whom and with what resources also requires factoring into the equation. The most effective way to estimate how long a particular activity will take is to know how long it has taken previously, and to assume that it will, other things being equal, take the same amount of time again. Given that design and development work consists in doing many things which have not been done before, the attempt to determine how long they will take is a matter of outright guesswork.

Many of the decisions which have to be made are, further, judgements as to how much is enough. How much is enough is something which can often be determined only retrospectively. How much detail one needs to go into is a question which cannot be given a formulaic answer. Thus, in the preparation of a project involving a collaboration with the Japanese arm of the company, more-than-usually extensive preparations were undertaken to ensure that the largely unprecedented collaboration would be effective. Contacts were established between team members from Japan and the UK to ensure that, for example, the languages would provide no barriers. It was not, however, realised that the team members from Japan were those with a mastery of English, chosen to make contact both practical and possible. However, the actual work of the project would acquire collaboration with engineers who had only Japanese. Nonetheless, though the question 'how much is enough' can only be confidently answered with hindsight

Typically, within organisations a given project is one among a number. Over its course it will involve a varying number of individuals, typically a smaller number during the phase in which the overall design is being worked out, more in the phases during which the design is being implemented and the systems tested. The design work varies not only in terms of numbers employed, but also in terms of the kind of work to be done and the mix of skills required to do it. The projection of the organisation of work was required to determine the nature of the team required and to ensure the availability of appropriate members at the point in time. The basic objective was, of course, to ensure that design and development staff were continuously fully employed whilst also ensuring that they could be rotated from project to project as they were needed. Projects had, therefore, to make specific demands for the time and the scheduling they would require from particular engineers in order that the effort of those engineers could be optimally allocated amongst competing teams.

Foresight was also involved with respect to the dependencies between activities. The variation in personnel over the course of the project was, of course, related to the kinds, as well as the amounts, of work which would be involved in the project and, of course, some of these were interdependent, with some activities being preparatory to others.

Since the projects were directed toward the production of marketable products, the projects were directed toward a 'launch date' and the planning was done, therefore, with respect to the determination of what would need to be done to meet this. The planning of lines of activity, therefore, typically involved determining the latest date by which an activity needed to be undertaken, and the earliest date at which it could begin. This depended upon its relationship to other activities in the

project. Thus, hardware and software testing could only begin when there were combinations of machines and software being produced which were sufficiently robust to serve as test machines, and the expected availability of the test equipment provided a time by which the design of the test programme must be completed. Working out a programme of testing involved specifying a set of procedures to be followed in using the machines and a set of matrices which provided a set of cycles through which the test operators were to put the machines. The provision of a standard set of operations was related to the objective of assembling a body of data which could be systematically analysed, enabling the diagnosis of faults. The co-ordination of software releases with hardware development was another prominent feature: for example, the hardware development proceeded, in part, through the progressive increase in functionality, and the development of software releases was tied to this, such that the software release would support the new functions.

The plan provided, therefore, a scheduled set of activities and these were, effectively, targets for those working on the projects, indicating when things were to be done by. If the target date was to be met, if the testing process was to begin on time, then there were various things to be done. As already noted, there is a certain indefiniteness to the scheduling of project activities, for the time which it takes to carry out some task or to complete some phase of the process may be variable. The policy was to set reasonably tight — or, as they were termed 'aggressive' — schedules, ones which would ensure that those on the project would be kept constantly busy, would be working long hours and that the work would be pervaded with a mild sense of urgency. If the targets were to be met, there was no time to waste.

The scheduling is not, however, typically so tight that any problems will throw the entire plan off schedule. None of the projects studied was set up on the assumption that everything would 'go right first time', but the more usual schedule allows for the prospect that there will be problems and delays in carrying through a task. The plan, as noted, provided a set of targets which could be regarded as more or less reasonable relative to the work that needed doing by those subject to them, and the meeting of those targets was regarded as a relatively high priority matter, not least to avoid the consequences of failing to meet them. As target dates approached, the priority would move to making efforts to meet them, and in some cases, this would involve doing things merely and solely to make the target — such as providing new software releases though the work properly preparatory for them had not been done, because the pressure from hardware to have a release to facilitate their own adherence to schedules was intense. 'Schedule slips' were a preoccupation and every effort was made to avoid them though they would, nonetheless, take place and the project timetable would be revised.

Requirements in Interaction[48]

The second study we report on was done as part of the COMIC Project and involved application designers and potential users jointly testing, evaluating and generating new requirements for a prototype system for design support, namely, DNP. Unlike the studies reported previously, this was part of a research project rather than the development of an industrial application.

The design session focused around a prototype version of Designer's Note Pad (DNP) for the support not only of design activities but of others, too, including the analysis of ethnographic materials. Published accounts of DNP (Twidale *et al*, 1993) emphasise how the system has been designed to allow users considerable flexibility. However, prior to this session, the users were computer scientists who had used DNP as an aid to technical design. On this occasion the work was to examine how ethnographers might use DNP for structuring observational field notes.

The System

The system provides a window, or 'design', in which users may place a number of 'entities'. Entities are the primary basis of DNP and are essentially graphical shapes (circles, square, etc.) taken from a palette. Links may be drawn between these entities in several ways. Pads of 'post-it notes' containing free text may be created and attached to these entities. Post-it notes have to be attached to an existing entity. An entity may be extended 'downwards' into a 'subdesign' shown in a new window. Subdesigns have the same features as designs. A 'report' may be printed of the entities, links and attached post-it notes that the user has defined. A report is a formatted textual view of the relationship between the graphical entities.

Ethnographical materials were used in the session which was video recorded and the talk transcribed. Thus, it offers an insight toward understanding how requirements emerge within interaction.

The Emergence of Requirements

Inspection of the transcripts shows that requirements were rarely directly expressed by the user. Rather it is often the designer who formulates the 'requirement' by responding to the user's troubles and anticipating the 'requirements' without the user directly indicating a want, need or desire. Much of the talk is oriented around 'design commonplaces'; that is, maxims for 'good design', a principle which forms part of the stock of knowledge of system designers. Among these include:

- modularised code facilitates reuse
- it is bad to overload the user with information
- user-interface consistency is desirable
- giving users feedback on the effects of their actions is desirable
- and more.

---

48 This section draws upon and summarises COMIC-MAN-2-3 by Bowers and Pycock.

It is important to note that, at least in this case, the usefulness of the design commonplace has to be worked up. It often comes after the discussions of alternative possibilities and their rationales. This is not so much a design guideline which generates design possibilities as a way of rationalising what has already been discussed.

As explicit requests for design were rare, so were explicit disagreements on the part of the user to suggestions made by designers or vice versa. All parties were attentive to the development status of the DNP and formulate their suggestions so as not to demand too much of the system or its next version.

Some Implications for Design

One major implication of the study is that involving end-users in design is often assumed to be a means of finding out about pre-existing requirements that users might have; hence, the notion of requirements 'capture'. However, in this case, requirements emerged out of the negotiations between designer and user and are not available to be 'read off' ready made from the kind of dialogue that took place here. Indeed, requirements were pulled to and fro between participants as they were constructed into a provisionally stable form.

While there are good arguments for involving users in the process of design, this is clearly not enough. What happens to users and their contributions and how designers respond is a vital matter for consideration. Hitherto, most requirements capture methodologies have only sought to analyse the outcomes of the talk and debate that goes into the formulation of requirements. However, it is important to examine, in some detail, the processes by which they are arrived at as resources for design itself. For one thing, it enhances the traceability of requirements and the exploration of alternative design possibilities.

Concluding Remarks

In this part of the Deliverable we have reviewed what we call a social point of view on design which makes use of ethnography to analyse the real world settings of work and its socially organised activities. We have illustrated this approach by presenting material drawn from two studies of the design process: one of these, research which is ongoing, is from the commercial environment of system design and development, and the other illustrating how requirements may evolve through system use.

The point of focusing on the design process in this way is not, however, simply to illustrate the social point of view on design. The study of the design process is important to the COMIC project in meeting one of the major needs which has been identified, namely, supporting the communication between the rich and detailed analyses of work that ethnography brings to informing requirements and the needs of engineers for highly structured and abstract models of the system they intend to build. A greater awareness of the real world actualities of the design process is important to understanding what a support tool would need to provide in various settings. In this respect, future work within this Strand will be concerned to

develop an existing tool (the DNP) to support the recording and analysing of ethnographic materials of the social organisation of work.

The approach we have taken toward design is to see it is a 'workaday' activity and, as such, imbued with the thoroughly practical concerns that any work exhibits. It seeks to understand the process of design not as Kling's 'system rationalists' might do but as a course of activity which is subject to various contingencies, some outside the control of the designers, some unanticipated problems that arise in any project, some which are a result of the methods of organising the process, and more. The point of this, to repeat, is to display the real world character of design.

None of this is an argument against method. Far from it. It is an argument for understanding what methods can and cannot do in the process of designing large scale systems. We have emphasised throughout the importance of acknowledging both the needs of software engineering and, in the context of CSCW design, those of developing adequate analyses of the real world social organisation of work activities which is essential for CSCW systems.

Design is a complex affair and in the next Part we identify some of the different and varied approaches which can be taken to understanding the process of design. We argue that the complexity of the real world, and consequently of design, is such that a number of different perspectives need to coexist and be managed as part of design. These are discussed as a series of heterogeneities within the methods and models of the design process. The 'workaday' and everyday character of design is such that the uncovering of appropriate means and mechanisms for recording and representing different aspects and perspectives becomes crucial in CSCW.

Part IV
Requirement Methods
and
CSCW Systems Development

Part IV: Requirement Methods and CSCW Systems Development

In considering the various natures of system development and the relationship between cooperative work and the construction of systems we have examined a diverse collection of techniques and methods. These techniques have largely evolved to support the general problems associated with requirements and how these are related to the software development process. Rather than consider the construction of software systems within a particular domain, development is viewed as an abstract and generic set of activities invariably undertaken to realise a software system. Thus, many of the methods and techniques examined in practise take an *a priori* general view of how software should be developed rather than considering how this is done within the development of software systems for a particular application or set of applications. Our examination so far has mainly considered the nature of this abstract consideration of systems development. This has been undertaken both from the perspective of involving a consideration of the social nature of work within system development and the nature of development itself as a social process.

In this final part of the Deliverable we wish to complement our consideration of the general features of software development by an examination of requirements within the context of building large scale cooperative systems. The intent of this examination is threefold:

1. To place the development of CSCW systems requirements within the general context of systems design and development.
2. To uncover the pertinent issues surrounding requirements methods and techniques which have relevance to the construction of CSCW systems.
3. To highlight the key issues which need to be addressed to allow a systematic consideration of CSCW systems development and to support the construction of cooperative software within an industrial context.

This is intended to reflect on the development methods examined and summarised in Part II of the deliverable and the debate resulting from this. In doing so we wish to both summarise and review the nature of these systems before considering what we feel are the minimal characteristics of CSCW systems that development methods and techniques must service. The following section reflects on the disparate nature of current requirements models and methods and the presuppositions reflected in them. This is followed by a consideration of the central features of CSCW systems which requirements must reflect and are crucial to future systematic considerations of requirements development. Finally, we isolate the main research issues arising from this consideration and turn to the attentions which will be the focus of this Strand in the next two years.

The Heterogeneous Nature of Method

One of the major tasks of the first year of COMIC research in this Strand has been to examine most of the extant methods of developing systems requirements. This examination is intended to provide a context for the assessment of empirical studies of work and their relation to the development of CSCW systems. One of the main points we made is that the consideration of method and development techniques is problematic in that, very often, the traditions from which different methods have emerged are disparate. Consequently, when considering method in any systematic way, rather than seeing any extensive commonality, we discern an overwhelming heterogeneity of approach.

We have emphasised throughout that there are no panaceas for the problems of system development, not least because these involve an assortment of difficulties. The evaluation of each of the many methods needs to assess them, first of all, in terms of the problems they are intended to address. It is not, we think, possible to dismiss any of the methods we have described as entirely useless to system development for their creation is commonly in response to some real and important problem in system design. Though they may not resolve those problems, the methods make a constructive contribution to their solution. At the same time, their utility must not be overrated, because the methods, typically, have a specialised nature and are suitable only to some of the problems from across the whole range of system development.

Within this section we wish to reflect on some of the more prominent and relevant forms of heterogeneity evident in requirements methods before considering the implications for the role of requirements in terms of the problems of developing CSCW systems. In our examination of the variety of methods we have detected at least the following sources of heterogeneity

- The origins and presuppositions of the method
- The particular problems they address
- The extent to which they focus on detail
- The objective of the method
- The community the method is intended to serve

The following sections briefly review the ways in which each of these sources are evident within the different methods and techniques examined in Part II of the Deliverable and considers some of the consequences for the applicability of the different methods. In particular, we wish to attempt to outline the means by which methods can, in practise, adequately service the needs of respective parts of the development process. This very ability of methods to meet the actual needs of their community of users within particular domains is precisely the reason why many of these methods will persist despite shortcomings, and why many methods which have such shortcomings have been amended and evolved in attempts to overcome them.

A Heterogeneity of Background

Requirements methods which aim to represent and uncover the needs of users and organisations have arisen from a variety of backgrounds. In Part II we discussed different requirements methods and techniques which have grown from backgrounds which included:-

- Modelling approaches based on systems theory
- A sociotechnical consideration of organisational work
- Process based models of work
- Cognitive theories of the nature of users
- A consideration of the behaviour of software systems.

Obviously, each of these different traditions have strongly influenced the nature of the requirements techniques and development methods associated with them. In addition to these strong historical influences, many of the requirements methods commit their users to a set of theories which are essential to the particular requirements technique. As a result, developers using one of these approaches to requirements capture need to accept a particular theoretical perspective in order to effectively use the method. Examples of methods which exhibit this strong theoretical dependence include:-

- FAOR which assumes an acceptance of soft systems as a means of understanding organisations.
- ETHICS accepts the account of Mumford on the key issues in organisations.
- ICNs assumes a representations of office work as a network of processes.
- Task analysis techniques rely on a cognitive account of the relationship between users and machines.
- SADT promotes a strong functional perspective on the behaviour of systems.

In addition to assuming the acceptance of a particular theoretical stance, these methods are intended to service the needs of users as defined or recognised within that theoretical framework. Problems and breakdowns emerge when either the supporting theory has little to say about a particular set of requirements or a particular domain of user's needs. Alternatively, it may be that the particular supporting theoretical stance is seriously challenged in some way as, for example and to some extent, cognitive theories are being challenged by various kinds of social constructivist theories.[49] Of course, it is also possible that methods can achieve some independence from their initiating theoretical basis.[50] The problem with this is that they then tend to be treated as if they are capable of tackling any problem in design and requirements specification, irrespective of their original theoretical motivation. More generally, given the complex nature of cooperative work and the variety of organisational contexts within which it is situated, it is

49 See, for example, Coulter (1983)

50 This is the case with many of the methods used in social research. Surveys, questionnaires and interviews, for example, are treated as standard research tools, so much so that their original supporting theoretical basis has been forgotten. See Benson and Hughes (1991) for a review of this issue.

unlikely that any single theoretical perspective will prove equal to the task of serving as an overarching framework for CSCW requirements.

In contrast to those approaches which require developers to accept a particular theoretical perspective on the orientation of systems and users, a large number of requirements techniques have emerged which represent a more pragmatic response to a perceived problem and presuppose no particular philosophy or theory. These systems tend to focus, for example, on the solution of particular problems of capturing and codifying information and assessing completeness and consistency for the construction of a software system. However, they have little to say about users and the nature of organisations and how these relate to the constructed systems. This is because they are designed to solve problems which are internal to the development process itself, problems to do with preserving the integrity of information.

Given the lack of any single theoretical perspective which supports uncovering all of the disparate features of work and organisations relevant to CSCW systems requirements, the problem is twofold: first of all, accepting that there are different theoretical perspectives relevant to the nature of work and organisations; second, accepting that not all will, or do, work equally well with respect to specific problems and particular domains.

It means acknowledging, too, that constructing requirements can never be merely a matter of the application of a method or a technique; it also requires judgement.

A Heterogeneity of Problems

In addition to having their roots within a particular background or theoretical perspective, different approaches to requirements often attend to particular aspects of the process, with different methods attending to different aspects. Because the process of developing requirements is multifaceted this means that each method may have validity with respect to the aspect which it treats. CSCW systems development adds new facets to this process by bringing in the problems which are involved in creating cooperative applications. In this section we wish to briefly consider some of the aspects of the development process on which different methods are focused, particularly those which have relevance for the development of CSCW systems.

Support for organisational change

Many of the systems discussed in Part II of the Deliverable are centred on understanding the needs of users in terms of the organisational setting of the system's intended use before attempting to develop an appropriate IT solution. Two distinct approaches are evident within the requirements methods identified in Part II.

- Methods which are based on developing a model of the organisation based on soft systems; for example, FAOR.
- Methods which support the management of organisational change within the broad context of sociotechnical development; for example, ETHICS.

Each of these approaches focus on gaining a general understanding of the many relations between users and the system to be developed rather than directly considering what properties the system needs to have in any detail. The motivation for this stance is that the relationship between the organisation and the developed systems is such that it dominates any other feature of systems development. The relationship between cooperating users and the organisational context is no less important within CSCW. In fact, many argue that this relationship characterises CSCW systems development. This is reflected within the work of COMIC, particularly, but not solely, within Deliverable 1.1. Consequently, the problem of determining the relationship between the organisation and the system will constitute a significant part of the determination of requirements for CSCW systems.

Operation and System Interaction

A prominent perspective directed at uncovering the properties of interactive systems has been based on a cognitive view of the user tasks. This particular perspective sees the central issue as that of ensuring that the goals and tasks of users are sympathetically and systematically aligned with the mechanistic behaviour of the supporting computer system. This has resulted in the development of a wide range of methods aimed at uncovering the tasks a user must undertake to achieve a particular set of goals. These were reviewed in Part II and include:

- Hierarchical Task Analysis
- TKS
- GOMS

As we have already seen in our considerations of HCI and task analysis, a consideration of the relation between HCI and systems development is relatively recent. As a result, current task analysis methods focus on the operational characteristics of users rather than a consideration of what needs to be constructed (Dix *et al*, 1993). Nevertheless, task analysis methods are capable of providing supplementary information on the behaviour of users which is essential to the development of interactive systems at least at the level at which they focus. Future CSCW systems will need to be attentive to the problem of relating the behaviour of users and cooperation applications.

The distinction between cooperative systems and the current considerations of the relationship between users and computers embodied within task analysis lies within the complexity of relationships manifest in CSCW systems. The current division between user and system prominent within HCI, and embodied within task analysis methods, nominates as the core issue of development, the user interface. However, within cooperative applications where a far greater number and variety of interfaces are likely to become more prominent, this particular characterisation of

the configuration between the user and the interface becomes problematic (Bowers and Rodden, 1993). The inability of the standard conception of the interface to reflect the complexity of issues within the real world is reflected within Task analysis methods which poorly reflect the complexity evident in CSCW systems.

Understanding and automating work

A core aspect of cooperative systems is the nature of the work which the application would need to support. This consideration has been reflected in CSCW by the development of a wide range of systems which attempt to capture the nature of work as a shared activity, with constituent roles and goals, in a formal model. Many of these different models of cooperative work have been criticised for failing to consider the situated nature of work (Suchman, 1987). These issues are explored in more detail in Deliverable 3.1.

A central feature of this research is the attempt to develop support systems through an understanding of cooperative work in order to model it within a system which coordinates the activities of users. This has resulted in a number of computational models representing work activities. These include:

- The use of Petri Nets in Domino and CHAOS
- The use of state transition networks to represent speech-acts.
- The use of CSP formalisms in software process models.

Each of these different formalisms have their own particular proponents who argue for their respective, and often competing, merits. In general, however, the perspective adopted by all of these systems is to represent work as a set of processes and flows between interdependent activities. This approach is reflected in many process modelling and enactment systems particularly those aimed at business processes and office automation.

While much recent focus within CSCW has been on the need to provide freedom to support unexpected and spontaneous cooperation, it is important to remember that a considerable proportion of cooperative work has a planned and co-ordinated aspect to it. Thus, although the models referred to above cannot adequately capture the emergent aspects of work, they do offer the prospect of representing and servicing the more planned and stabilised procedures of work environments, and these will need to be considered in future CSCW systems, perhaps particularly for cooperative applications in software development and office domains. Many of the issues involved are notational and are being explored elsewhere in the COMIC project. (See Deliverable 3.1).

Identifying and recording what to develop

Finally, we turn to the development oriented methods reported in the latter portion of Part II of this Deliverable. Without exception, the primary aim of all of these is the need to record the properties of what needs to be developed. The methods focus on the means by which these properties can be represented, and subsequently reasoned about, to ensure the integrity of the systems developed. Techniques for representing the properties of systems include:

- As a set of functions
- In terms of the control effects exhibit by the system
- As a set of predicate based knowledge of the system
- In terms of a set of identified objects
- In terms of some mathematical formalism

In attempting to record and reason about the properties of the system being developed, the focus of most of these methods has been on working out the mechanistic properties a computer system must exhibit, typically characterised in terms of the services offered systems. This approach focuses entirely upon the internal structure of the computer system and thus avoids any consideration of other relationships between the system and the surrounding environment. More recently a range of viewpoint based approaches have enabled a richer reflection on, and thus heightened awareness of, how these systems appear to their users. However the focus is still on the mechanistic behaviour of the system, with only a limited exploration of the system-user relationship. The 'viewpoint' methodology may, however, have the potential to give effective representation to a much more wide ranging awareness of the variety of users in relation to the system and may thus facilitate the development of CSCW systems.

Heterogeneity of Detail

The problem of managing the complexity inherent in constructing a large scale software system is enormous. In the case of cooperative systems where the system involves a need to understand a large set of complex relationships between: users, organisational elements and procedures, and other cooperative applications, the problems seriously proliferate. A crucial component of engineering generally is the necessity to keep track of an immense amount of detailed information, and the objectives of CSCW add to that necessity in respect of system development. CSCW demands that the relation of the intended system to the needs of the users must be understood in a detailed way if system design is to be informed effectively, and that the design incorporates sufficient sensitivity to user activities in a correspondingly detailed way.

Existing approaches to understanding the needs of users vary tremendously in the importance they attach to detail. The degree to which they recognise and respond to the needs to manage detail in many ways reflects the initial perspective of requirements approaches. Development oriented methods such as those based on functional or object approaches exhibit a considerable attention to detail. The properties of each unit of the developed system are reflected within the requirements method and considerable emphasis is placed on issues of traceablity.

In contrast, approaches to requirements such as FAOR and ETHICS focus on an organisational perspective of system development and emphasise the need to develop a clear conceptual overview. The system is reflected upon as a whole and the various relationships evident within the organisation are considered together. Little attention is paid to particular properties of individual units of the developed system.

Striking the right balance between abstraction and specificity is problematic, and approaches will often be criticised for erring in one direction or the other, either sacrificing the clarity of the overall picture to a grasp of detail, or gaining an overall picture at the expense of the detail, as has been the case with approaches we have examined in Part II.

This is a natural tension in the development of requirements, particularly within CSCW systems. Much of the requirements process revolves around the need to uncover the properties to be exhibited by the developed systems. Of necessity this requires a focus on detail and the ability to manage the associated problems of complexity and moving towards well-grounded abstractions useful for system development. Moreover, the impact that CSCW systems will have on the workplace means that this detailed uncovering of the properties of the system has to occur within the context of a whole set of organisational concerns. Ethnographic methods, for example, are more than capable of producing rich, and essential detail, about the social organisation of work activities. It is an important research theme in this Strand to develop ways of using this detail as a basis for producing well-grounded abstractions for use in CSCW system development.

Heterogeneity of Objectives

We have previously considered the way in which different approaches to requirements address themselves to different problems and embody a specific, and commonly distinctive, set of objectives. These different objectives constitute a further form of heterogeneity across methods. As in the previous case each of these different objectives are appropriate considerations for constructing requirements for cooperative systems.

For example, the principle objective of the different task analysis methods considered in Part II is to gain an understanding of the behaviour of users. The methods seek to construct cognitively based models of the actions (or tasks) users will need to undertake in order to reach an identified goal. In uncovering and modelling user behaviour, task analysis techniques reflect the decomposition of goals into subgoals which must be met in order to realise an identified principle goal. A general hierarchical model is adopted to represent the manner in which users plan the tasks at hand. In modelling user behaviour most task analysis techniques focus primarily on the behaviour of users rather than a consideration of user behaviour in relation to the use of computer systems. Often the task analysis techniques follow a procedural consideration of the work at hand without any consideration of the context within which the work is taking place.

In contrast, a number of other methods have as their principle objective the improved understanding of the nature of organisations. For example, the methods based on soft systems and those which have emerged from a sociotechnical background view their principle objective as facilitating an understanding of the organisational context within which IT systems are placed. In doing so these methods focus on uncovering and representing the different type and nature of

relationships between entities within the organisation, including those involving the IT system. While this has the virtue of considering the IT system in relation to the organisation, the development of detailed features of the computer systems is seldom considered by these methods.

A more detailed consideration of the properties of computer systems and the representation of organisations can be seen within process modelling approaches. These approaches represent organisations as networks of activities and an associated work flow. The objective of developing these work flow representations is to mirror them in the construction of computer systems which enact these models. However, this direct mapping between an abstract model of work within an organisation and the behaviour of systems is considered problematic and a more complete discussion of the issues involved is given within Deliverable 1.1.

A further stated objective of process modelling approaches is the improvement of process. For example, Humphrey (1989) characterises processes in terms of a five level Process Maturity Model with the clear intent of improving processes in order to increase their level of maturity. Similar objectives are stated as the motivation for process models within software engineering and within a business context reviewed in Part II. A result of this emphasis on process improvement and work rationalisation is that users are often sceptical of the motives of process modelling approaches and feel that they are seldom involved.

Participative approaches to systems development attempt to balance the effects of less consultative approaches to design by making its objective the emancipation of the user community which the developed system intends to serve. In some cases it is an objective of this approach to make the users more active through ‘consciousness raising’ strategies. The involvement of the user in the design process is not, in this approach, a mere means to the end of a superior system design; it is, in important respects, an end in itself or, if a means, one directed toward politicising the user’s outlook. The aim is to involve the user in design to the extent that the difference between users and developers all but disappears. The possibility of meeting the conditions necessary for this approach in large scale system development has been doubted, as has been the effectiveness of user involvement as a way of transforming worker’s consciousness.

Heterogeneity of Users

Approaches to the development of requirements have evolved in the context of meeting the needs of their own particular user community. Given the multi-disciplinary nature of CSCW systems development, it is worth highlighting the particular users these methods are intended to service. For example, most of the development oriented methods identified in Part II of the Deliverable see as their primary users the developers of information systems. In doing so they identify the need to service the concerns of these developers and this is reflected in the nature of these methods.

Organisational methods based on soft systems identify as their principle clients a set of experts whose role is to analyse and understand organisations. A similar community of organisational analysts is seen as the user of process modelling approaches. However, the experts to which the two methods are addressed are distinct. The experts served by soft systems methods are those with a background in business analysis, and those to which process modelling is oriented experts in computer science and operations research.

In contrast to these approaches to understanding organisations, those based on a socio-technical approach are intended to support the management of information systems. These methods have evolved from earlier considerations of management information systems and the information systems managers within organisations are seen as the principle users of these methods. Consequently, little consideration is given to the development of novel computer systems. The implicit assumption is that the focus is on the procurement of standard software systems and technology. In some ways this contrasts with many classes of cooperative system which exploit highly novel technology to support cooperative work.

A final set of experts are identified as the potential users of the task analysis methods discussed in Part II. These experts have emerged from the increased prominence, during the 1980's, of HCI and are often termed human factors experts. These experts combine an understanding of the cognitive nature of the user and the properties of computer systems. The focus of many human factors experts is in the assessment and evaluation of user interfaces. Often task analysis techniques are used in this light rather than in the uncovering of requirements for new systems.

A characteristic feature of CSCW is the involvement of a wide variety of disciplines and experts. A consequence of their involvement is that each of these experts bring with them an understanding of a particular set of methods for understanding the needs of users, including those of designers and developers. This heterogeneity is an inescapable feature of CSCW systems development and it is important that any future considerations of systematic development is sympathetic to this heterogeneity at least to the extent of being aware that the heterogeneity of problems in CSCW design is likely to require different methods for their 'solution'.

The following section considers in more detail the needs of CSCW systems development. The intent of this examination is to highlight the future challenges which need to be addressed to support the systematic development of CSCW systems.

The Needs of CSCW

Within this Deliverable we have considered the nature of systems development and it relationship to considerations of design within CSCW. As part of this process we have described a wide range of methods which have emerged to support the development of requirements for different computer systems. As we have seen,

these different methods and techniques exhibit tremendous heterogeneity, some of which has been explored in the previous section. Within this section we wish to complement this consideration by examining what we consider to be the pertinent needs of CSCW systems development before considering the development of requirements for CSCW systems in more detail.

Existing cooperative systems have made little progress in gaining widespread acceptance outside research labs despite the considerable growth in network availability during the last decade. A number of researchers have presented different arguments for this apparent failure of CSCW systems to become more acceptable (Grudin 1988). This issue apart, perhaps not surprisingly the main focus of CSCW research has been on small scale systems which are realisable within the context of a research lab. The majority of these systems have limited user groups which are, very often, strongly linked to the development team.

In contrast, we wish to consider an alternative situation where CSCW systems are developed within a more 'industrial context'. Under this arrangement the small scale nature of the software can no longer be assumed. It is more likely that the developed system will constitute a large scale software project with all the associated engineering difficulties. It is worth highlighting three core issues which characterise the nature of CSCW systems development we envisage:

- users within a context of work
- technical difficulty of large scale system development
- support for the management of development

Although each of these issues are distinctly identifiable within CSCW development they are not unrelated. In fact most of these development issues are not unique to CSCW although the interaction between them characterises the key difficulty of cooperative systems development.

Understanding users within a context of work.

One of the more distinctive feature of CSCW is the extent to which it is able to focus much more on work and use than being technologically driven as many previous approaches to systems were and, to some extent, still are. As we have pointed out elsewhere, a number of researchers have argued for the need to treat seriously an understanding of the nature of users and cooperative work as part of the development process (Bannon, 1991; Suchman, 1983; Grudin, 1988). Not surprisingly, however, despite great enthusiasm for such a step there has been considerably more reticence on how it can be achieved. The most notable candidates to emerge have been participative approaches to systems development (Kuhn and Muller, 1993; Greenbaum and Kyng, 1991) and the involvement of ethnographic observation (Bentley *et al*, 1992; Heath and Luff, 1992).

Incorporating an understanding of the nature of a development domain has been an issue which has itself plagued more traditional forms of systems development. Indeed Curtis *et al* (1988) have identified the lack of application domain knowledge as the most significant problem in requirements engineering. The obvious need for

this domain knowledge has motivated much of the work on both user centred and participative design techniques. Within cooperative settings the use of ethnographic approaches has emerged as a prominent candidate. However, little consideration has been given to how this information may be effectively communicated to systems developers.

The technical difficulty of large scale development.

While a minority of CSCW systems place only limited demands on the technical infrastructure used to realise them, a significant proportion of cooperative applications exploit either novel or emerging technology. This is increasingly likely to be the case in the development of large scale cooperative applications. In addition to providing a significant application domain for both computer and communication technologies, CSCW asks challenging questions of these infrastructures in their turn. The majority of existing computer systems have evolved to support an individualistic perspective and many of the design decisions inherent in this philosophy are built into existing infrastructures (Schmidt and Rodden, 1992). The result is that either significant modification or the replication of these existing infrastructure facilities are needed in order to support cooperative applications.

Cooperative systems place a wide array of technical challenges before system developers. They require an understanding of the nature of communication and distributed systems to be combined with a knowledge of users and software systems development. As a result, it is important that appropriate facilities are provided to support the development of these technically challenging applications. A key aspect is the provision of facilities to record the complex and detailed requirements which emerge within technically challenging projects. Any requirements technique which claims to support the development of requirements for CSCW will need to service this attention to detail.

Support for the management of development.

The vast majority of current CSCW systems have been developed in an informal research oriented manner by a handful of people. The complexity involved in realising future large scale cooperative systems is such that they will need to be developed by large teams of programmers. A direct consequence of this observation is that this development will have to be managed and co-ordinated to ensure successful system development. This management will need to take account of a number of issues:-

- The cooperative nature of the development process.
- The need to support the maintenance of the developed system.
- The change in personnel over the lifetime of the project.
- The need to assess the progress of the project.
- The relationship between the developer and procurer of the cooperative system.

Supporting these different features of managing a large scale project places particular emphasis on the need to accurately record and represent user's requirements. Requirements form the basis of the relationship between system procurer and developer and facilitate an assessment of progress in the project. In addition, they allow the aims of the project to be communicated across development teams and allow these aims to persist as team members leave. Finally, a clear record of requirements allows future maintainers to reflect on the reasons for the structure of the software system. Current development oriented methods have focused on meeting many of these needs. However, in doings so they have tended to adopt a prescriptive approach to recording requirements which is insufficiently expressive of the needs of cooperative systems.

Meeting the Challenges of CSCW

Most current CSCW systems are 'concept demonstrator' prototypes which illustrate some basic ideas or which demonstrate how some cooperative support function might be realised. By and large, commercial CSCW products (such as Microsoft Windows for Workgroups™) have been derived from existing systems rather than from an analysis of a work setting which revealed the need for cooperative systems support.

While this approach is an understandable first step, if CSCW systems are to become commercially successful and widely used, a systematic approach to understanding end-user and organisational requirements for CSCW will be necessary. Adding some technical support for more than one user to existing products without an understanding of the work setting where these products will be used will result in products with severely limited applicability.

Furthermore, a disciplined and professional approach to CSCW systems development is required. Considerable attention has been paid to the appropriateness of different process models and many of these have been discussed in Part II of this deliverable. Irrespective of the particular relationship and dependencies between them, a number of separate activities are important to this approach to development :

1. *Requirements analysis and specification*

   This activity involves understanding an existing work setting and specifying the required functions and properties of the supporting CSCW system in sufficient detail that these requirements can form the basis for a contract for system procurement.

2. *System design and development*

   A system must be designed and implemented in such a way that it meets its specification and can be maintained over a long lifetime. This requires the use of standards, well-documented designs, structured programming and extensive system documentation.

3. *System validation against the specification*

   The system must be validated against its specification to demonstrate that the contract to develop the system has been fulfilled.

The model of software procurement which is almost universal is that a specification of the software is defined and a system procurer (who is responsible for developing the specification) lets the contract to develop the software to a system developer (who is responsible for designing, implementing and validating the software). Where software is developed as a product, the system procurer is, in essence, the marketing department of a company; for bespoke software systems, the system procurer and the system developer are usually separate organisations.

The development of a clear specification is central to the software engineering process as the specification, in many cases, acts as a legal document setting out what work will be paid for by the procurer. Techniques such as participative and user-centred design (Greenbaum and Kyng, 1991; Norman and Draper, 1986) are undoubtedly effective in their own terms and have the potential to produce systems which are more attuned to user needs. However, they suffer from the fundamental limitation that there is never a complete system requirements specification to act as a basis for system procurement.

A current research challenge is to develop a model of the software process which allows for the use of user-centred methods during a large systems development project where a legally binding definition of the system must be agreed between system procurer and system developer. At the moment, these methods may be used during the development of a system prototype as part of the requirements analysis phase but, because of cost considerations, this is necessarily limited.

Proponents of user-centred approaches recognise this difficulty and suggest that it might be resolved by modifying the way in which software is procured (Grudin, 1991). Grudin suggests that software procurement should be service-oriented rather than product-oriented i.e. you buy a service which you trust rather than write detailed product specifications. Whilst this is perhaps the best solution from a technical perspective for some classes of interactive system, it is somewhat naïve as it would require immense organisational change to bring about. It would require the notion of a contract between procurer and developer to be redefined. Realistically, this will not happen in the near future and the vast majority of software will continue to be developed according to the traditional model.

In the part of the COMIC project report here we are concerned with the integration of professional software engineering methods and techniques for understanding users within CSCW. We recognise that conventional approaches to requirements engineering are inadequate for CSCW yet recognise also that ‘human-centred’ approaches in themselves do not meet the industrial need for a clear and systematic system specification. Consequently an objective of our work is to investigate how ‘human-centred’ analysis methods such as ethnography which recognise explicit and implicit human cooperation may be integrated with more conventional approaches to system requirements engineering. In doing so a number

of issues which need to be addressed from both a computer science and social science perspective can be identified.

The role of social sciences in uncovering requirements

In Part I of this Deliverable we reviewed some of the problems that have arisen over the introduction of social science, particularly sociology, to system design. The point was made that this has proved to be, to date, a mixed blessing and very much a promissory note that it can deliver appropriate analyses which can inform CSCW system design. As we pointed out, sociology, like other disciplines, is not a unitary endeavour but one which has its own internal debates and disputes which, as we have seen, are more than capable of spilling over into CSCW design.

For our part, and a major focus of this Deliverable, the most promising sociological approach is one which focuses sharply on the details and specificities of the situated, 'real world' social organisation of work and its activities. It is this which brings ethnographic methods of analysis into the requirements process. Some of the advantages such a method can bring to CSCW system design are summarised as follows:[51]

- System building is an intervention in a 'real world' organisation of work which can deal the unaware designer some nasty surprises, many of which can frustrate the effectiveness of the system.
- A well done ethnography can sensitise designers to the 'real world' context and character of work and, in this respect, can be an important educational exercise.
- It offers the opportunity to inform the design of systems which have more resonance with their circumstances of use and, in particular, help to answer vital questions as to what to leave to human skill and experience and what to leave to the computer. In other words, questions about how systems can be designed to support and enhance cooperative work activities.
- Ethnography can offer the opportunity to open up design possibilities. Typically, the motivation for design is to offer a solution to some problem. However, greater knowledge of the socially organised patterns of life within a setting may alter the frame of reference for design by loosening it from subscribing to some simple and decontextualised formula.[52]
- Ethnography is classically the sociological method which has as its especial focus the detailed study of the social organisation of activities and, thus, is capable of meeting one of the acknowledged requirements of CSCW system development, namely, knowledge of the socially situated character of work.

However, realising these potentialities in reference to the kind of conditions of CSCW system development previously outlined is a matter of research. Some of the issues involved here are identified in the next section.

---

51 See COMIC-LANCS-2-4 (Hughes *et al*, 1993) for further discussion.

52 See (Bentley *et al*, 1992)

Incorporation of ethnography in the development process

The problem of understanding the cooperative nature of the domain within which a CSCW system is placed is a key concern in developing an appropriate set of requirements for such systems. The potential value of ethnography in deriving computer system requirements was first identified in seminal work by Suchman (1983, 1987). Subsequently, further studies concerned with air traffic control (Harper *et al,* 1991), police database systems (Ackroyd *et al, 1992*) and underground railway control (Heath and Luff, 1992) have confirmed that an ethnographic study can give real insights into working processes which should be taken into account when deriving computer system requirements.

The notion that there is a fixed process or procedure for most tasks which can be automated is, of course, an over-simplification. The existence of a 'working division of labour' (Anderson *et al,* 1989) rather than the prescribed organisation is one important reason why the requirements for a software system are often such that the system does not meet the real needs of end-users. Too often system requirements are defined according to documented procedures and standards but do not take into account actual working practices.

Involving prospective end-users of a computer system in the requirements analysis does not solve this problem, though it can be of considerable assistance. We know from work in knowledge acquisition that experts find it very difficult to articulate their expertise (McGraw and Harbison-Briggs, 1989). It is equally, if not more, difficult for end-users to describe the working division of labour which is, in fact, informal and dynamic and routinely taken-for-granted. In some cases, the actual work practices may be quite contrary to organisational standards and the end-users of the technology will simply not admit that these practices go on. The general advantages of studying a cooperative application domain which requires some form of CSCW systems support using ethnography include:

1. *An explicit identification of cooperation*

   Ethnography promotes the identification of the subtle, often implicit, cooperation which is central to the functioning of many existing systems. For example, previous ethnographic studies of air traffic controllers, revealed that cooperation involving 'at a glance' understanding of other controller's displays was an essential part of the process. Similar subtle forms of cooperative interaction have been reported by Heath and Luff (1992) in London Underground control room settings.

2. *The identification of process and data variability*

   A central assumption which underlies conventional requirements engineering is that both processes and data change relatively slowly (over months or years). Within a cooperative setting processes particularly are rapidly reconfigured to cope with the variability of work. Even the structure of data may change informally as individuals cope with deficiencies in the given data organisation.

3. *The identification of organisational influences on system requirements*

   Most development oriented approaches to requirements universally ignore the organisation and organisational culture in which the system is to be delivered. The organisational structure and culture is a critical factor in making a CSCW system work and ethnography provides us with some insights into this organisational context.

4. *The provision of an account of the nature of a setting*

   A serious defect of all current requirements methods is that they are predominantly descriptive in nature. System requirements are specified but no record is maintained of why a particular requirement exists. This leads to misunderstandings of the requirements and to significant problems when requirements change. An ethnographic investigation not only records actual activities but also looks for an account of why these activities are they way they are. This information can be associated with specific system requirements.

A positive 'side-effect' of ethnography is that end-users often feel more involved in the process.[53] When structured methods are used, end-users often feel left out of the requirements analysis process as the process is designed to treat them as 'processing stations' rather than as intelligent participants in a complex process. Ethnography has the advantage that is involves users and allows them to volunteer system requirements and to identify shortcomings of existing systems. Despite these advantages several problem areas can be identified which must be tackled before ethnography can be used systematically in the sort of commercial projects needed to realise large-scale CSCW systems:

1. *The ethnographic process*

   Existing experience of understanding system requirements from ethnographic studies have taken place within a research context and the ethnographic study involved took place over an extended period. While a prolonged ethnographic study is possible for very large system development projects where the specification phase may last 2 or 3 years, it is unrealistic for the majority of system specification activities. These must be completed in a relatively short time (typically 2 or 3 months) if the customer's delivery time is to be satisfied. We need to consider the effects of such demands on the ethnographic process if effective results are to be delivered so quickly.

2. *The nature of the ethnographic record*

   The results of an ethnographic analysis are usually recorded as unstructured text with inevitable overlaps, repetitions, etc. It is difficult for this text to be used by anyone apart from the ethnographer who was involved in the process. Furthermore, the collected data may be in the

53 There is no guarantee of this, of course. Much depends upon the deportment of the ethnographer and, quite independently of this, other factors which may be operating in the setting, such as the threat of redundancy.

form of hand-written notes, electronic text, printed documents and diagrams, audio and video tape recordings etc. The heterogeneous nature of this record compounds the problems of finding information.

3. *Inter-disciplinary communications*

   Most ethnographic work is currently carried out by anthropologists or sociologists with most requirements analysis carried out by computer scientists or engineers. These disciplines have little in common in that they adopt quite different methodological approaches to a problem, use mutually incomprehensible jargon and suffer from mutual distrust which, for historical reasons, has developed between 'soft' and 'hard' sciences. It is unlikely that these prejudices will disappear and it is important they are serviced as part of our research. This may involve a consideration of the educational needs of future professionals involved in the requirements process.

An acceptance of multiplicity

As we have seen in this Deliverable the nature of developing requirements for cooperative and interactive systems involves the process of gaining an understanding of the complex alignments between users and the supporting system. A diversity of approach and philosophy was strikingly evident from our examination of the methods and techniques developed to help in gaining an understand of the needs of users. This diversity is not surprising given the complexity of the problem at hand. It is unlikely that any single perspective, method or technique will adequately and completely reflect the nature of work within an organisational context which is why, earlier, we emphasised the importance of taking a pragmatic approach to these matters.

Consider for example the most widely used development oriented approaches to system requirements engineering examined in Part II of the Deliverable. The most prominent techniques are centred on building either data-centred or process-centred models of the system.

1. Data-centred models are concerned with describing the structure of the data that is processed by the system. The most widely used data-centred model is an entity-relationship model (Chen, 1976) which shows the entities in a system, their attributes and their relationships. Other models track the dependencies between data entities and the life history of data in a system.
2. Process-centred models are concerned with describing data processing as a data entity moves from one 'processing station' to another. A 'processing station' may be a program or a person and each 'processing station' is responsible for some data transformation. Thus an approach such as data flow modelling (DeMarco, 1979) shows the processing stations involved in a process and names the data which 'flows' from one processing station to another.

Until recently, a process-centred model was seen as central with the data model as subsidiary to it. However, the development of object-oriented analysis (Coad and Yourdon, 1989, Rumbaugh *et al,* 1991) has reversed this with the objects (data) in the system being identified and specified before the processing steps.

However, none of the widely-used analysis methods, irrespective of whether they are based on a process-centred or a data-centred approach, explicitly allow for the inclusion of negotiation and cooperation. It is left to the judgement of the requirements engineer to determine how much this cooperation should influence the system requirements.

We would argue the industrial development of CSCW systems will fail to meet the real needs of their users if they ignore this essential dimension of cooperation. It really depends on whether or not the individual requirements engineer implicitly applies a user-centred approach and recognises the importance of cooperation and is sufficiently sympathetic and intuitive to understand the cooperation and reflect this in the system requirements. We wish to move beyond this by allowing provision for a more systematic incorporation of the actual needs of users.

Some more recent methods of requirements engineering have adopted the notion of viewpoints (Kotonya and Sommerville, 1992) where it is explicitly recognised that requirements for a system derive from different sources who may have quite different perceptions of the system. These 'viewpoint-based' approaches are principally concerned with looking at viewpoints in isolation rather than as cooperating entities. They are concerned with reconciling conflicting needs (which is a key problem in requirements engineering) rather than identifying viewpoint interactions and cooperation. Proponents of these methods recognise the importance of cooperation (Finkelstein and Fuks, 1989) and these viewpoint-oriented approaches may, in future, include explicit cooperation specifications. However, viewpoint based approaches are focused on a recording of the detailed relationships between the computer system and users rather than a more general examination of the nature of systems within a context of work.

The examination and extension of viewpoint based approaches to requirements will provide a considerable focus for this strand of work over the forthcoming years of the COMIC project. A number of reasons motivate our choice of viewpoints as a means of supporting the development of requirements for CSCW systems.

- Viewpoints are naturally cooperative in that they highlight the multiple orientations people may have to a supporting system.
- Viewpoints are naturally sympathetic to the heterogeneity evident within the development of CSCW requirements
- Viewpoints provide a means of setting the multiplicity of user needs along side each other to inform the construction of requirements.
- Viewpoints are generally understood within the development community and dissemination of information in this community will be enhanced.

An examination of viewpoints within requirements development for CSCW systems involves the consideration of a number of different issues.

- Is there a conceptual basis for informing the identification of appropriate viewpoints?
- How do we handle the massive heterogeneity to allow appropriate viewpoints to be associated when appropriate?
- How do we support multi-disciplinary perspectives within the context of viewpoint oriented approaches to understanding requirements?
- How do we represent viewpoints to developers and users and convey information they contain on the needs of users across a development team?

Many of these issues are central to the concerns of multi-disciplinary working within CSCW. An examination of the concerns they embody through these issues will form a significant portion of the research agenda for Strand 2 of the COMIC project over the forthcoming years.

The exploration of tools to record requirements.

The development of more structured methods and notations for empirical observations of the nature of work in an organisational context is a long-term research goal. In the near future, however, we believe that a way forward is through more pragmatic studies on the use of techniques such as ethnography to inform the requirements engineering process. The development of detailed methods and notations requires a wider corpus of practical experience than is currently available. However, observational techniques such as ethnography can make some contribution now to the requirements engineering process, given that we can effectively organise the ethnographic record.

The need for a more structured ethnographic record is an immediate one if ethnography is to be used effectively in deriving cooperative systems requirements. We are convinced that an effective way to provide this structure is through software tool support. The tool must integrate the requirements of the ethnographer for informal information capture with support for a more structured approach to requirements expression which allows the requirements to be partitioned and analysed.

The Ethnographer (Seidel and Clark, 1984) is one example of a computer-based tool to support ethnographic record management. However, Davies (1990) comments that this system and comparable tools:

> "... impose many unwelcome constraints on the researcher and s/he has to significantly alter the methods and techniques of analysis to fit in with a given system"

As part of this project we have been experimenting with a tool known as the Designer's Notepad (Sommerville *et al,* 1993; Twidale *et al,* 1993) which was originally developed to support the initial phases of the systems design process. This is a hypertext-based system with a simple and neutral vocabulary which allows entities and relationships of interest to be represented within limited commitment (figure 40). The diagrams produced by the DNP are similar to those

produced by processes such as cognitive mapping which have been applied in requirements capture (Brooks and Jones 1993).

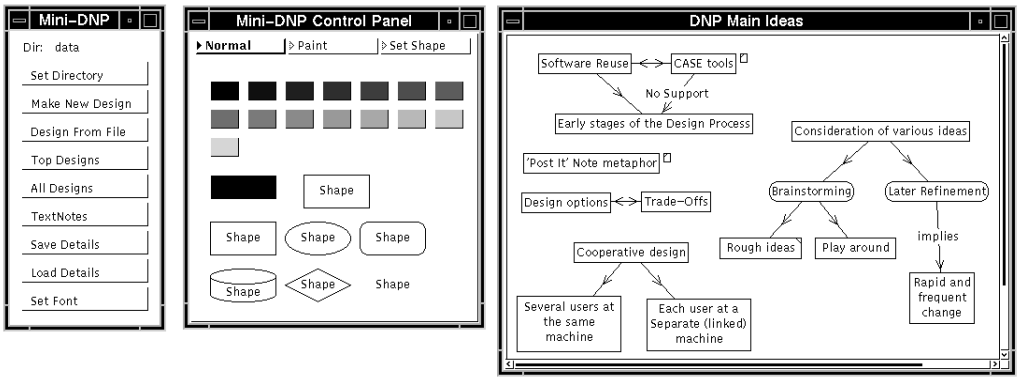

Figure 40 — The designers notepad

A direct result of the DNP's basis in system design is that the same system can be used for both informal information organisation and for supporting more structured requirements engineering methods. Thus, information from an ethnographic study may be integrated with more structured information and links created between formal requirements and the ethnographic observations which serve as rationale for requirements. The emphasis is on supporting the evolution of structure by adding semantics to the notational networks.

The focus on tool support helps us to address the problem of prolonged ethnography. In essence, the ethnographic studies generate 'nuggets' of useful information at unpredictable intervals. In conventional ethnography, these are made explicit in an analysis phase where the ethnographic record is analysed after the field studies have been completed or at least during the later phases of the field study. We wish to examine alternate ways of organising this relationship:-

- We wish to examine a working practice whereby the ethnographer does not work on site for long, uninterrupted periods but returns regularly to report on the progress of the work. Interim records can be entered into the Designer's Notepad. We hope that by entering these into the DNP the ethnographer's attention will focus on the existing records which will often suggest useful structuring. The records then become immediately available to a requirements engineers. Thus, ethnography and conventional analysis can be carried out in parallel with the stream of ethnographic information integrated into the specification as it becomes available.
- We wish to develop different techniques for generating projections of the record held within the DNP that can be shown to different stakeholders in the requirement process. In turn this development will consider:
    - How these projections relate to viewpoints within the requirements process.
    - Making these projections interactive to allow them to be used as an additional tool in gaining requirements.

Given the exploratory nature of the project we have adopted a rapid prototyping approach to the development of tools support. In doing so we have started to drive the development of tools from an observation of the use of these tools in practice. An analysis of one of our initial sessions in adopting the DNP is given in Appendix I. This analysis highlights in itself the social nature of the development of requirements and we believe reflection of this form will be a key characteristic of future design sessions.

The role of notation

As we have already stated, an ethnographic record of work practice is inherently unstructured. It consists of observations of work processes made over an extended period of time. Inevitably, there is a significant amount of duplication and the information collected ranges from specific observations of particular activities to anecdotes and 'war stories' told by workers to the ethnographer. Ethnography is almost completely dependent on natural language for expressing knowledge of a work setting.

When considered in the realm of requirements the ethnographic record takes on a communicative dimension in that it is a means of conveying the needs of users to system developers. However, when an ethnographic record is used by non-specialists who are involved in requirements analysis, a number of problems can arise:

- There is the risk that readers and writers of the description, since they are trained in different techniques, may misunderstand because there may be no shared vocabulary for the work being studied.
- The ethnographic record is not partitioned so the inter-relationships between observations can only be discovered by examining all observations.
- The sequential nature of the record may mean that similar requirements are expressed in completely different ways. The reader has to find related requirements with the consequent likelihood of error and misunderstanding.

By contrast to the natural language used to express ethnographic observations, requirements specification typically uses a mixture of structured diagrams expressed using different graphical notations, natural language, and, increasingly, formal mathematics. The trend away from natural language towards more formal notations is motivated by many of the difficulties identified above.

The richness and flexibility of natural language descriptions while servicing the needs of the observer are problematic for the system developer who must attend to detail. There is a strong case for examining the use of a more structured notation for expressing ethnographic observations. Pragmatically, the development of such a notation would simplify the problem of ethnography becoming acceptable to software engineers. Again, the development of such a notation is a long-term research goal which should be carried out in conjunction with the emerging results on notations for mechanisms of interaction within strand 3.

Summary

This section of the deliverable has examined and reflected upon the experiences of the first year's work in this part of the COMIC project. Within the technical annexe of the project we characterised the principle objectives of this workpackage over the forthcoming years as:

" • The development of techniques and guide-lines to support cross disciplinary working in CSCW systems development.

• The extension of existing requirements capture techniques to incorporate data derived from observational studies.

• An investigation of tools to support the effective translation of observational results into CSCW systems. "

In considering each of these we can see that significant progress has been made in reaching each of these objectives. During the last year we have examined a wide range of different approaches to systems development and debated the many and varied roles of different disciplines within that process. Some conclusions from this examination were reported in Part I of the Deliverable and we believe this will provide a basis for the development of practical techniques and guidelines for the development of future CSCW systems.[54]

As a result of our examination of the nature of different requirements methods we have highlighted the need to provide a framework which supports the evident heterogeneity displayed by these methods. We have also highlighted the use of viewpoints as a means of providing this framework and see this as a significant extension to existing requirements techniques. Over the forthcoming years we hope to examine the conceptual basis for viewpoints and how these may be used to allow different perspectives from different theoretical perspectives to be contrasted and compared in the development of requirements.

A final part of our work has been to investigate the use of the DNP as a tool to support the recording and structuring of observational studies. The use of the DNP, which has already been developed to support the early stages of the design process, allows us to focus on the provision of facilities to support communication of emerging results from observational studies into the development process.

The development of systems requirements is problematic enough for stable and well understood application areas. In CSCW where we have very limited understanding of the nature of the application area and the consequences for systems the problems are exacerbated. However, given the large scale nature of CSCW projects we foresee that the effective development and recording of system requirements is essential to the development process. However, the questions of what to build and how to build it are endemic issues in design. There are no simple panaceas which provide answers, no one method which can, with assurance, provide answers to all the problems that CSCW system design poses. This is the

54 COMIC-LANCS-2-4 (Hughes *et al*, 1993) sets out some practical guidelines for the ethnographic support of the development of CSCW systems.

importance of an approach based on a multiplicity of viewpoints which recognises the manifold problems of design.

Over the forthcoming two years of the project we wish to examine the development of techniques which support the construction and subsequent evolution of requirements employing a viewpoint approach. The focus will be on an evolutionary model of requirements construction since the adoption of requirements development as a one-off process is likely to be less than satisfactory. We also wish to consider the context within which the design is being done. In particular, we wish to further contrast considerations of design and requirements development within research with its commercial counterpart through a series of empirical studies of the design process.

We also wish to examine in more detail the emerging relationships between this an other parts of the COMIC project. In particular, we would like to examine the relation between requirements and the representational models of organisations examined as part of the organisational work reported in Deliverable 1.1. We also wish to examine the notational issues in expressing CSCW systems in terms of the mechanisms of interaction reported in Deliverable 3.1.

Finally, in proposing to provide tool support through the use of a shared repository which represents the evolving results of an empirical study and the emergent requirements, we adopt a model of cooperation through shared information. This has also been a considerable focus of the work on shared objects reported in Deliverable 4.1, and we wish to examine the use of the tools developed there as a practical example of cooperative interaction through shared objects.

Appendix

Appendix: An initial examination of the DNP

This account is an analysis of an initial session of a prototype version of the 'Designer's Note Pad' (DNP). This system has been used to support a variety of activities. Published accounts of DNP (e.g. Twidale *et al,* 1993) emphasise how it has itself been designed to allow users considerable flexibility. However, most of its users prior to the sessions we studied were computer scientists who used DNP as an aid to technical design. Hence, the occasion of our work was to examine how a different set of users (ethnographers) might use DNP for their tasks of structuring observational field notes. The developers of DNP were interested to know how DNP might be used in this situation, what modifications were desirable and what ethnographers' requirements were for a support tool.

The system provides a window or 'design' in which users may place a number of 'entities'. Entities are the primary basis of DNP and are essentially graphical shapes (circle, square, etc.) taken from a palette. Links may be drawn between these entities in several ways. Pads of 'post-it notes' containing free text may be created and attached to the entities. Post-it notes have to be attached to an existing entity and cannot exist alone on the background or on links. An entity can be expanded 'downwards' into a 'subdesign' shown in a new window. Subdesigns have the same features as designs. A 'report' can be printed of the entities, links and attached post-it notes that the user has defined. A report is a formatted textual view of the relationship between the graphical entities.

The design session principally involved three main participants. JB a potential user, TR the project manager, and SM the implementor. During the session, at various moments, TR noted down a series of points arising which were then handed to SM as requirements. JB and TR had agreed that JB would bring a collection of field notes and related ethnographic materials to the session and explore how they might be analysed with support from DNP. Video recordings were made of the (day long) session which have been transcribed and which we have analysed. We present our main findings in the next section.

Analysis

An inspection of the transcripts we have reveals that requirements are very rarely directly expressed. The user (JB) very rarely says *I want X* or *I require Y* or even *I need Z.* More typically, JB will encounter some problem in the use of DNP, the fluency of his work with it will be disrupted and either (i) TR will anticipate what the trouble might be and offer a formulation of it or (ii) JB will ask about what he can do with DNP in a question whose form enables TR to do (i). In other words, JB rarely need have recourse to expressing requirements in any direct form. This means that it is inappropriate analytically to simply 'read-off' requirements from the transcript. Consider Transcript 1:

Transcript 1 (simplified data)

```
TR  right so that's currently generating a report for this ok. so
    you'll see what subdesigns can go ( ) ok
JB  right <JB taps fingers> can we see the report?
TR  you will when its printed ha ha
JB  oh it just prints=
TR  =just print=
JB  =it doesn't
TR  doesn't actually
JB  er show them on screen
TR  doesn't show them on screen
JB  yer
TR  you reckon it should show them on screen before printing
JB  ummm well I could imagine that you you (.) oh yes oh yes
TR  ok then (.) that's not that difficult (to do at all)
...
TR  you could just show show all this text (.) before (printing) it
SM  you mean just as a reinforcement that something is happening or
JB  well=
SM  =to be generally useful?=
JB  =yer I was wondering yer I was wondering
SM  cause maybe it's first time you've seen report and you want (.)
    you're not sure what going to get (.) may be if you use the
    system another time you'd know what sort of report you're going
    to get so you just want confirmation that it's doing something
JB  well there's another way of looking at it and that is that what
    comes out as a report is another view on (.) a more textual view
    on ( ) right whereas here is visual <JB points to screen>
TR  yer yer
JB  and certainly for certain ways of dealing with things it might
    be nice to actually see that kind of textual view
TR  ok
SM  um
JB  also I suppose once you get something quite er tangled erm some
    of the kinds of relations of designs to subdesigns er might
    actually be (.) come to be unclear (.) might actually
    paradoxically be more clear in a textual version that a er
...
JB  <bends around SM to speak to TR> oh and another thing it might
    support is if you want er to write a post-it note which was
    sensitive to say some other other thing which otherwise would
    otherwise be hidden from your view. So if you say wanted to
    write a post-it note to something in the light of another post-
    it note elsewhere I mean obviously you could go and open that
    and then and then stick it but also you might want to (.) you
    know having a report which showed you know say a series of post-
    it notes or something like that (.) might just be an easier way
    of of seeing a you know a a large amount of information which
    actually might not (.) which might be quite far apart in terms
    of the graphic (.) the graphics of it might be actually near
    together in text in the report or something. Sorry I'm I'm
    fishing for reasons here
TR  no no no
SM  I think as many different views on data you can get ( ) as
    possible are a good idea
JB  yer yer that's that's what I'm (attempting) to say
```

This excerpt begins with TR printing out a 'report' of the work JB has so far done with DNP. JB asks whether it is possible to see the report but TR indicates that the report cannot be seen on screen but only in print out form. It is then TR, the designer, who formulates the 'requirement': *you reckon it should show them on screen before printing* which JB agrees to. This is entirely typical of the interaction between JB and TR. The designer formulates the user's troubles and anticipates

'requirements' without the user directly indicating a want, need or desire. That the user may be experiencing trouble is made available to others either by asking a question about his own abilities or those of the machine, or by manifestly coming to a halt in his work with the machine.

Staying with this excerpt a little longer, we see SM formulating a possible rationale for what TR and JB have agreed on: *you mean just as a reinforcement that something is happening or to be generally useful?* SM goes on to give an account of why 'reinforcement' may be of use to provide a new user with feedback from the machine or *confirmation that it's doing something JB* does not explicitly disagree with SM but his *well* prefaces an alternative account which fashions a contrast between graphical and textual 'views'. He then provides an extended account of why textual views may be important: under some circumstances textual views may be clearer than graphical (*paradoxically*): information which is separated graphically may come together textually. Most of this talk is met with tokens such as *yer, um* and *ok* from TR and SM which mark moments where they could intervene and respond but do not, thereby allowing JB to continue to assemble a number of reasons for the utility of a 'textual view'. JB's disclaiming *Sorry I'm I'm fishing for reasons here* prompts a denial from TR whereupon SM offers *I think as many different views on data you can get as possible are a good idea* which JB agrees with.

Several features of this talk are worthy of note and are typical of many of the interactions between designers and users that we have studied. First, we suggest that much of the talk is oriented around what we would like to call design commonplaces. A design commonplace is a maxim for good design, a principle which forms part of the stock of common knowledge of designers. Likely candidates are: modularised code facilitates reuse, it is bad to overload the user with information, user-interface consistency is desirable, giving users feedback on the effects of their actions is desirable, etc. In the above excerpt, SM considers the importance of giving users feedback in his discussion of why JB may want to see reports on screen. Also the excerpt closes with SM suggesting that it is 'a good idea' to give the user many views on data. It is important to note, however, that — in this case — the usefulness of this design commonplace has to be worked up. It comes *after* discussions of alternative possibilities and their rationales. It is not so much a design guideline or a rule of thumb which generates design possibilities as a way of rationalising what has already been discussed and closing the topic down with agreement from participants. The existence of design commonplaces and the significance they have in use may also account for why JB notes that his advocacy of textual views is 'paradoxical': under certain circumstances, he claims, textual views may make for greater visual clarity than the graphical user interface offered by DNP. Here, perhaps, JB is orienting to a design commonplace which celebrates the virtues of graphics for interface design while here suspending its applicability (hence the *paradoxically*).

We have indicated that the explicit issuing of requests for redesign of the DNP is very rare in our materials. Equally rare are explicit disagreements on the part of the

user to suggestions made by designers or vice versa. JB, TR and SM are attentive to each other's skills and abilities and so do not directly confront each other by denying the worth of each other's contributions. Similarly, all parties are attentive to the 'in development' status of the DNP and formulate their suggestions so as not to be demanding too much of the machine or its next version. Refusals, disagreements and resisting suggestions for redesign take on a much more indirect form. It is worth noting again how in Transcript 1, TR and SM do not refuse emerging suggestions for redesign so much as *(re)formulate* them. The possibility that a report might be displayed on screen is formulated as being of use for new users or to give feedback. Displaying the reports in this way is readily 'doable'. Consider, in addition, Transcript 2 where it is indicated that something has been tried already and doesn't work:

Transcript 2 (simplified data)

```
TR  This is use is different from how people di-di-designs with it
SM  yer
TR  its very much different
JB  is it? that's interesting
SM  I mean the entities don't normally get great big long names like
    that as well
TR  yer
JB  right that is something I did think about as to you know
SM  you can actually reshape the entities so that long text is
    spread over several lines
...
JB  have you thought about making enter making it the entity
    (instead of) return?
TR  we've had that at one point and it confused a lot of people
JB  did it?
TR  well the reason for it is that people who did designs generally
    used quite short names and what was important was speed and
    getting things in
JB  right right
TR  and return and enter are a bit confusing
```

As JB has been making DNP entities with *great big long names*, he has pressed the 'return' key a number of times to space out his text within the entity only to find that a new entity is made as soon as he does so. Accordingly, he suggests that pressing the 'enter' key may be an alternative way of making entities which will allow users also to space out the text within entity names by using return. However, the confusion experienced by *a lot of people* who typically used *quite short names* is used by TR to resist JB's suggestion. JB's suggestions and his usage are thereby rendered different and without precedent.

Both Transcripts 1 and 2 suggest that much of JB's contribution to the session involves enquiring about DNP's text handling capabilities. In Transcript 3, JB makes a general point out of this but note how the generality is mitigated by emphasising that there should be *a bit more* 'fluidity/traffic' rather than directly suggesting radical redesign. Again, this enables TR to reformulate JB's generality in terms of extending existing features of the design (entities and post-it notes).

Transcript 3 (simplified data)

```
JB  I guess the general (.) the general kind of way to specify the
    problem that we are encountering here is is that there might
    need to be a bit more fluidity between entities and post-it
    notes right (.) so a bit more free traffic between entities and
    post-it notes (.) between (.) well the text that's in post-it
    notes=
TR  yer=
JB  =and [the text that's in
TR   [and text that's (in) entities need to be moved around and you
    obviously need to be able to reassign and move text and post-it
    notes ( )
JB  ok
TR  fair enough
```

In Transcript 4, JB envisages a specific scenario of use and makes some quite exact suggestions as to how this could be implemented. After he's prompted by JB asking *do you see the idea?* TR takes control of the mouse and keyboard and shows alternative ways of making arbitrary text into entities.

Transcript 4 (simplified data)

```
JB  now another way I mean I could envisage using something like DNP
    would be to just bang in loads of text
TR  um
JB  you know just to you know=
TR  =yer=
JB  =then draw boxes around ( ) and have entities emerge
(.)
JB  do you see the idea?
TR  ( ) that would be quite interesting to do <TR writes on pad>
... <TR tries out some possibilities on screen>
TR  thank god SM's not here cause he's got to do ( ) code it
JB  ( ) ha ha ha
TR  no I think that's doable I don't think that's I don't think
    that's that problematic
...
TR  ok so a means of (forming) entities from large-ish pieces of
    text would be good
JB  yeah
TR  see the other thing which is nice about this as away of doing is
    I think a process ( ) this has been assuming what ethnography
    ethnographers do but one of the things it seems really good at
    is coming out with the phrases the key components
JB  yes
```

In a sense, JB's scenario reverses the order given to text and graphical entities in the version of the DNP he was using. He suggests a scenario in which text is primary or at least can be anterior to making entities out of it. JB's reversal of this has been heard as unusual in a number of the excerpts we have seen. In Transcript 2, SM noted JB's *great big long names* and made some suggestions for how he might manage *long text* with existing facilities. In Transcript 3, JB generalises 'the problem' ultimately into one to do with the relation between *text* and entities. However, there, TR reformulates JB's generalisation into a comment about the relation between *post-it notes* and entities. It is only with Transcript 4 that we see text being considered separately from the existing user interface facility for handling text (post-it notes).

In arguing for the 'free traffic' between text and graphical entities, JB has had to deploy a variety of rhetorical resources. He *listed* several reasons for having a textual view. He has emphasised that this is a *general* (and not a particular) point. He has presented in fairly *concrete* terms how this might appear (drawing boxes around text). TR has said that this is 'doable' but is relieved that SM, who writes the code, was out of the room at the time! Interestingly, it is at this moment that we see an account of how this potentially radical redesign of DNP has come about: the session has revealed the true nature of ethnographers! And they're not like computer scientists, nor like how computer scientists imagine them to be!

Conclusions

We hope that the foregoing analyses have illustrated a number of points. First, users in the kind of user-designer cooperative setting we have studied rarely express or have to express 'requirements' (or wants or needs or desires) directly. Frequently, the designer will anticipate troubles in advance of their occurrence or provide a formulation of them once they have occurred. Similarly, users will offer their contributions as queries as to what is possible, doable, within their capabilities or within those of the designer or machine. This and other phenomena indicate that users and designers are orienting to each other's skills and knowledge in ways which obviate the need for direct requests or refusals. As we remarked at the outset, the involvement of end-users in design is often assumed to be a way of finding out about the pre-existing requirements that users might 'have'. However, the degree of interactivity in the formulation of problems and their solutions, and the mutual orientation that participants display to one another, suggest that the crude attribution of requirements to users is deeply problematic. In a sense, requirements are a negotiated product of argument and resistance and are not available to be read off (as-it-were) 'ready made' from dialogues of the sort we have observed.

Initial expressions of troubles and their possible solutions may subsequently be strengthened by reasons and rationales, reworked, returned to later, explored for consequences, or concretised in terms of existing features of the system or perhaps possible scenarios of use. The requirement may be pulled to and fro between participants as it is constructed into a provisionally stable form. Participants are seen to deploy a mixed range of forces — including the use of design commonplaces — to decide the fate of a suggestion or evaluation.

Attempting to muster enough forces to suggest a radical redesign and to 'undo' some of the effort and decisions embodied in the prototype is here seen to be difficult and demanding relative to the resources available to users. In this design session we have seen how the user builds up to challenging the primacy of graphical entities over text, using a form of argument which the designer might accept without reworking it in terms of existing system features. While Woolgar (1991) usefully suggests that sessions like the one we have looked at involve configuring the user, this is only part of the story. Neither designers nor their machines came out blemish free. Users configure designers too!

Let us briefly examine what the implications of our study may be for methods of requirements capture and analysis. While there are good arguments for involving users in many aspects of design, *simply* prescribing their participation cannot be seen as sufficient. Exactly *what* happens to users and their contributions and *how* designers respond has to be a matter for close consideration. Our study indicates the existence of a number of interactional phenomena lying behind the 'user-requirements' documented by the designers during the session.

It might be suggested that requirements capture methodologies do not deny that requirements are the product of talk and debate, but rather that they have only sought to analyse the *outcomes* of these debates. While these (typically written) outcomes often have value in the management of a design project and are often of legal importance in the contracts which are drawn up between client and developer, this is not an argument for ignoring the processes by which they are arrived at. Indeed, we suggest that materials like the ones we have collected (video, transcriptions, email between participants) and the kinds of analyses presented here may well enhance the 'traceability' of requirements and the exploration of grounded alternative design possibilities if, say, what the designers made of the 'official' requirements turned out to be fruitless.

Bibliography

Ackroyd, S., Harper, R., Hughes, J.A., Shapiro, D., and Soothill, K. (1992), *New Technology and Practical Police Work*, Milton Keynes, UK: Open University Press.

Aglietta, M. (1987), *A Theory of Capitalist Regulation,* London: Verso.

Agresti, W.W. (1986), The Conventional Software Life-Cycle Model: Its Evolution and Assumptions, in *New Paradigms for Software Development,* ed. W.W. Agresti, 2-5, Washington, DC: IEEE Computer Society Press.

Alford, M. (1977), A requirements engineering methodology for real-time processing requirements, *IEEE Transactions on Software Engineering,* 3 (1): 366-374.

Alford, M. (1985), SREM at the age of eight; the distributed computing design system. *IEEE Computer*, 18 (4): 36-46.

Ambriola, V. and Montangero, C. (1992), OIKOS at the age of three, in *Proceedings of 2nd European Workshop on Software Process Technology — EWSPT'92*, ed. J.C.Derniame, 84-94, Berlin: Springer-Verlag.

Anderson, R., Button, G., and Sharrock, W., (1993), Supporting the design process within an organisational context, in *Proceedings of the Third European Conference on Computer Supported Cooperative Work,* ed. G. de Michelis, C. Simone and K. Schmidt, 47-59, Dordrecht: Kluwer.

Anderson, R.J., Hughes, J.A., and Sharrock, W.W. (1989), *Working for Profit: The Social Organisation of Calculation in an Entrepreneurial Firm,* Aldershot, UK: Gower.

Atkinson, J. (1986), *Changing Work Patterns: How Companies Achieve Flexibility to Meet New Needs,* London: National Economic Development Office.

Auramäki, E., Lehtinen, E. and Lyytinen, K. (1988), A speech-act-based office modeling approach, *ACM transactions on Office Information Systems*, 6(2): 126-152

Avison, D.E. and Wood-Harper, A.T. (1990), *Multiview: An exploration in Information Systems Development*, Oxford: Blackwell.

Bagguley, P., Mark-Lawson, J., Shapiro, D., Urry, J., Walby, S., and Warde, A. (1990), *Restructuring: Place, Class and Gender,* London: Sage.

Bailin, S. (1989), An object-oriented requirements specification method, *Communications of the ACM* 32 (5): 608-623.

Balzer, R., Goldman, N. and Wile, D. (1978), Informality in program specifications, *IEEE Transactions on Software Engineering,* 4 (2): 94-103.

Bandinelli, S., Fuggetta, A. and Ghezzi, C. (1991), Software process as real-time systems: A case study using high level petri nets, in *Proceedings of 1st European Workshop on Process Modeling — EWPM'91*, ed. A.Fugetta, R.Conradi and V.Ambriola, Italian Society of Computer Science (AICA) Press.

Bandinelli, S., Fuggetta, A., Ghezzi, C. and Grigolli, S. (1992), Process Enactment in Spade, in *Proceedings of 2nd European Workshop on Software Process Technology — EWSPT'92*, ed. J.C.Derniame, 67-98, Berlin: Springer-Verlag.

Bank, J. (1992), *The Essence Of Total Quality Management*, Prentice Hall.

Bannon, L. (1991), From human factors to human actors: the role of psychology and human-computer interaction studies in systems design, in *Design at Work: Cooperative Design of Computer Systems,* ed. J. Greenbaum and M. Kyng, Hillsdale, NJ: Lawrence Erlbaum.

Bannon, L. (1993), Use, Design, and Evaluation: Steps Towards an Integration, Working Paper, COMIC-RISØ-2-3.

Bannon, L. and Bødker, S. (1990), Beyond the interface: Encountering artifacts in use, in *Designing Interaction: Psychology at the Human-Computer Interface*, ed. J. Carroll, 227-253, Cambridge, UK: Cambridge University Press.

Bannon, L. and Schmidt, K. (1991), CSCW: Four characters in search of a context, in *Studies in Computer Supported Cooperative Work:Theory, Practice and Design,* ed. J. Bowers and S. Benford, Amsterdam: North-Holland.

Bansler, J. (1989), Systems development in Scandinavia: three theoretical schools, *Office: Technology and People,* 4(2): 117-133.

Bansler, J.P. and Bødker, K. (1993), A reappraisal of structured analysis: design in an organizational context. *ACM Transactions on Information Systems*, 11(2): 165-193.

Barber, G.R., de Jong, P. and Hewitt, C. (1983), Semantic support for work in organizations, in *Proceedings of Information Processing 83. The IFIP 9th World Computer Congress*, ed. R.E.A. Mason, 561-566, Paris, France.

Barnard, P. (1991), Bridging between basic theories and the artifacts of human-computer interaction, in *Designing Interaction: Psychology at the Human-Computer Interface*, ed. J. Carroll, 103-127, Cambridge, UK: Cambridge University Press.

Barnes, B. (1977), *Interests and the Growth of Knowledge*, London, UK: Routledge.

Barstow, D.R. (1985), Domain-specific automatic programming, *IEEE Transactions on Software Engineering*, 11(11): 1321-1336.

Belady, L. and Lehman, M. (1976), A model of large program development *IBM Systems Journal,* 15: 225-252.

Belkhatir, N., Estublier, J. and Melo, W.L. (1991), Adele2: A support to large software development process, in *Proceedings of the 1st Conference on Software Process (ICSP1)*, 159-170, Redondo Beach, CA.

Bell, T.E., Bixler, D.C. and Dyer, M.E. (1977), An extendible approach to computer aided software requirements engineering, *IEEE Transactions on Software Engineering*, 3(1): 49-60.

Bengtson, S., Jensen, O. and Hubert, P. (1985), *Analyse og design på kontorområdet — belyst gennem et eksperiment med tre forskellige metoder [Analysis and design in the office domains — An experiment with three different methodologies]*, DAIMI.

Benson, D. and Hughes, J. (1991), Method: evidence and inference — evidence and inference for ethnomethodology, in *Ethnomethodology and the Human Sciences*, ed. G. Button, 77-108, Cambridge, UK: Cambridge University Press.

Bentley, R., Hughes, J., Randall, D., Rodden, T., Sawyer, P., Shapiro, D. and Sommerville, I. (1992), Ethnographically informed systems design for air traffic control, in *Proceedings of CSCW92*, ed. J. Turner and R. Kraut, 123-129, Oct 31-Nov 4, Toronto, Canada: ACM Press.

Berard, E.V. (1993), *Essays on Object-Oriented Software Engineering*, Englewood Cliffs, NJ: Prentice-Hall.

Berger, P., and Luckmann, T. (1966), *The Social Construction of Reality,*, New York: Doubleday.

Bergland, G.D. (1981), A guided tour of program design methodologies, *Computer*, October: 13-37.

Bergstra, J.A. and Klopp, J.W. (1984), Process algebra for synchronous communication, *Information and Control*, 60(1/3): 109-137

Berlinksi, D. (1976), *On Systems Analysis: An Essay Concerning the Limitations of Some Mathematical Methods in the Social, Political and Biological Sciences,* Cambridge, MA: MIT Press.

Bickerton, M. and Siddiqi J. (1993), The Classification of Requirements Engineering Methods, in *Proceedings of RE 93: International Symposium on Requirements Engineering*, Jan 4-6, San Diego: IEEE.

Bittner, E. (1973), Objectivity and realism in sociology, in *Phenomenological Sociology*, ed. G. Psathas, 109-125, New York: Wiley.

Bittner, E. (1974), The concept of organisation, in *Ethnomthodology*, ed. R. Turner, 69-81, Harmondsworth: Penguin.

Bjerknes, G., Ehn, P., and Kyng, M. ed. (1987), *Computers and democracy — a Scandinavian challenge*. Aldershot, UK: Gower.

Bjorner, D. (1987), On the use of formal methods in software development, in *Proceedings of the 9th IEEE International Conference on Software Engineering*, 17-29, Washington, DC: IEEE Computer Society Press.

Bjorner, D. and Jones, C. (1978), *The Vienna Development Method,* New York: Springer-Verlag.

Bødker, S. (1990), *Through the interface: A human activity approach to user interface design.* Hillsdale, NJ: Lawrence Erlbaum

Bødker, S. and Grønbæk, K, (1991), Cooperative Prototyping Studies: Users and Designers Envision a Dental Record System, in *Studies in Computer Supported Cooperative Work:Theory, Practice and Design*, ed. J. Bowers and S. Benford, Amsterdam, North-Holland.

Bødker, S. Ehn, P., Kammersgaard, J., Kyng, M. and Sundblad, Y. (1987), A Utopian Experience, in *Computers and democracy — a Scandinavian challenge,* ed. G. Bjerknes, P. Ehn and M. Kyng, 251-278, Aldershot, UK: Avebury.

Boehm, B. (1976), Software Engineering, *IEEE Transactions on Computers*, 25(12): 1226-1241.

Boehm, B. (1988), A spiral model for software development and enhancement, *Computer,* 21: 61-72.

Bolognesi, T. and Brinksma, E. (1987), Introduction to the ISO dopecification language LOTOS, *Computer Networks and ISDN Systems*, 14: 25-59.

Borgida, A., Greenspan, S. and Mylopoulos, J. (1985), Knowledge representation as the basis for requirements specifications, *IEEE Computer*, 18 (4): 71-80.

Bostrom, R.B. and Heinen, J.S. (1977), MIS problems and failures. A socio-technical perspective: Part II: The application of socio-technical theory, *MIS Quartely*, 1(4): 11-28

Bowers, J. (1991), The janus faces of design: some critical questions for CSCW, in *Studies in Computer Supported Cooperative Work:Theory, Practice and Design*, ed. J. Bowers and S. Benford, Amsterdam: North-Holland.

Bowers, J. (1992), The politics of formalism, in *Contexts of Computer Mediated Communications*, ed. M. Lea, Hassocks, UK: Harvester.

Bowers, J. and Middleton, D.J. (in press), Distributed organizational cognition: An innovative idea? In *The management of intellectual resources*, ed. W. Turner, Paris: Economica.

Bowers, J. and Pycock, J. (1993), Requirements in Interaction: A Study of the Talk between Designers and Users, Working Paper, COMIC-MAN-2-3.

Bowers, J. and Rodden, T. (1993), Exploding the Interface: Experiences of a CSCW Network, In *Proceedings of InterChi 93,* April, Amsterdam: ACM Press.

Bowker, G., Gasser, L., Star, S.L., and Turner, W. (1993), Presentation at Workshop on Social Science Research, Technical Systems and Cooperative Work, Apr 8-10, CRNS, Paris.

Braverman, H. (1974), *Labor and Monopoly Capital*, New York: Monthly Review Press.

Brooks, F.P. (1975), *The Mythical Man-Month*, New York: Addision-Wesley.

Brooks, F.P. (1987), No silver bullet. Essence and accidents of software engineering. *IEEE Computer*. April: 10-19.

Brooks, L. and Jones, M. (1993), Cognitive mapping in requirements analysis, paper at UK CSCW Special Interest Group meeting *Requirements as Cooperation: Requirements for Cooperation*, Jun 15, DTI, London.

Bruyn, W., Jensen, R., Keskar, D. and Ward, P. (1988), ESML: An Extended Systems Modelling Language Based on the data Flow Diagram. *ACM Software Engineering Notes*, 13(1): 58-67.

Bruynooghe, R. F., Parker, J.M. and Rowles, J.S. (1991), PSS: A system for process enactment, *First International Conference On The Software Process*, California.

Bucciarelli, L.L. (1988), An ethnographic perspective on engineering design, *Design Studies*, 9(3): 159-168.

Bullen, C. and Bennett, J. (1991), Groupware in practice: an interpretation of work experience, in *Computerization and Controversy: Value Conflicts and Social Choices*, ed. C. Dunlop and R. Kling, San Diego: Academic Press.

Burke, K. (1935), *Permanence and Change,* New York: New Republic.

Burlon, R., Cardile, B., Conti, M., Pietri, F., Puncello, P. and Torrigiani, P. (1989), A knowledge-based tool for requirements analysis, in *22nd Annual Hawaiian International Conference on System Sciences*, 78-84, Hawaii: IEEE.

Burrell, G. and Morgan, G. (1979), *Sociological Paradigms and Organisational Analysis,* London: Heinemann.

Button, G. and King, V. (1992), *Hanging around is not the point*, paper given to workshop on Ethnography and Design, CSCW92, Toronto, Canada.

Buxton, J.N. (1978), Software Engineering, in *Programming Methodology,* ed. D. Gries, New York: Springer-Verlag.

Card, S., Moran, T.P. and Newell, A. (1983), *The Psychology of Human-Computer Interaction*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Carey, M.S., Stammers, R.B. and Astley, J.A. (1989), Human-computer interaction design: the potential and pitfalls of Hierarchical Task Analysis, in *Task Analysis in Human Computer Interaction*, ed. D. Diaper, Ellis Horwood.

Carlsson, J., Ehn, P., Erlander, B., Perby, M-L. and Sandberg, Å (1978), Planning and control from the perspective of labour: a short presentation of the Demos project, *Accounting, Organizations and Society*, 3(3-4): 249-260.

Carroll, J.M. (1990), Infinite detail and emulation in an ontologically minimized HCI, in *Proceedings of CHI 90.*, Reading, MA: Addison-Wesley.

Carroll, J.M., Kellogg, W.A. and Rosson, M.B. (1990), *The Task-Artifact Cycle*, in *Designing Interaction: Psychology at the Human-Computer Interface*, ed. J. Carroll, 74-102, Cambridge, UK: Cambridge University Press.

Catterall, B.J. (1990), The HUFIT functionality matrix, in *Human-Computer Interaction — INTERACT'90*, ed. D. Diaper, D.Gilmore, G.Cockton and B.Shackel, 377-381, Amsterdam: North Holland.

Checkland, P. (1981), *Systems Thinking, Systems Practice*, Chichester: John Wiley & Sons.

Checkland, P. and Scholes, J. (1990), *Soft Systems Methodology in Action*, Chichester: John Wiley & Sons.

Chen, P. (1976), Entity-relationship approach to data modelling. *ACM Transactions on Database Systems,* 1(1).

Cherns, A. (1987), Principles of sociotechnical design revisited, *Human Relations*, 40: 153 -62.

Clement, A. (1988), Office Automation and the technical control of information workers, in *The Political Economy of Information*, ed. V. Mosko and J. Wasko, Wisconsin: Wisconsin University Press.

Clement, A. and P. Van den Besselaar (1993), Participatory design projects: a retrospective look, *Communications of the ACM*, 36(6).

Coad, P. and Yourdon, E. (1989), OOA-Object-Oriented Analysis. Englewood Cliffs, NJ: Prentice-Hall.

Colbert, E. (1989), The object-oriented software development method: a practical approach to object-oriented development, in *Proceedings of the TRI-Ada 89 — Ada Technology In Context: Application, Development, and Deployment*, 400-415, New York, NY: ACM.

Conradi, R. Jaccheri, M.L., Mazzi, C., Nguyen, M.N. and Aarsten, A. (1991), Design, use and implementation of SPELL, a language for software process modeling and evolution, in *Proceedings of 2nd European Workshop on Software Process Technology — EWSPT'92*, ed. J.C.Derniame, 166-177, Berlin: Springer-Verlag.

Cook, C.L. (1980), Streamlining office procedures — An analysis using the information control net model., in *Proceedings of National Computer Conference*: 555-565.

Cooper & Bowers (forthcoming), Discourse of HCI, in *Social and Interactional Dimensions of Human-Computer Interfaces*, ed. P.Thompson, Cambridge, UK: Cambridge University Press.

Coulter, J. (1979), *The Social Construction of Mind*, London: Macmillan.

Coulter, J. (1983), *Rethinking Cognitive Theory*, London: Macmillan.

Coulter, J. (1989), *Mind in Action*, London: Macmillan.

Curtis, B., Krasner, H., and Iscoe, N. (1988), A field study of the software design process for large systems, *Communications of the ACM,* 31: 1268 — 1289.

Czuchry, A.J.J. and Harris, D.R. (1988), KBRA: A new paradigm for requirements engineering, *IEEE Expert*, 3(4): 21-35.

Davis, A.M. (1982), The role of requirements in the automated sythesis for real-time systems, in *International Symposium on Current Issues of Requirements Environments*, 151-58, Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Davis, A.M. (1990), *Software Requirements Analysis and Specification*, Englewood Cliffs, NJ: Prentice-Hall.

De Cindio, F., De Michelis, G., Simone, C., Vassalo, R. and Zanaboni, A. (1986), CHAOS as a coordinating technology, in *Proceedings of CSCW86*, ed. D. Petersen, 325-342, Austin, Texas: ACM Press.

De Grace, P. and Stahl, L.H. (1990), *Wicked Problems, Righteous Solutions: a catalogue of modern software engineering paradigms*, Englewood Cliffs, NJ:Yourdon Press.

De Michelis, G. (1993), Computer Support for Cooperative Work: Computers between Users and Social Complexity, Worlking paper, COMIC-MILAN-1-1.

De Michelis, G. and Grasso, M.A. (1993), The organizational context of a work process, Working paper, COMIC-MILAN-1-2.

DeMarco, T. (1979), *Structured Analysis and System Specification*, Englewood Cliffs, NJ: Prentice-Hall.

Dewitz, S.K. and Lee, R.M. (1989), Legal procedures as formal coversations: Contracting on a performative network, In *Proceedings of the Tenth International Conference on Information Systems*, ed. J.I. DeGross, J.C. Henderson, and B.R. Konsynski, Boston, MA.

Dieters, W. and Gruhn, V. (1990), Managing software processes in the environment MELMAC, in *Proceedings of the 4th ACM SIGSOFT Symposium on Software Development Environments: In SIGSOFT Software Engineering Notes*, 15(6).

Dietz, J.L.G. (1991), Speech acts or communication action, In *Proceedings of the Second European Conference on CSCW — ECSCW91*, ed. L. Bannon, M. Robinson, and K. Schmidt, 235-248, Dordrecht: Kluwer.

Dietz, J.L.G. (1992a), Modelling the essential activities of an organization, in *Information Modelling and Knowledge Bases III,*, ed. Ohsuga, S. *et al,* 130-140, Amsterdam: IOS Press.

Dietz, J.L.G. (1992b), Subject-oriented modelling of open active systems, in *Information Systems Concepts: Improving the Understanding*, ed. E.D.Falkenberg, C.Rolland and E.N.El-Sayed, 227-238, Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Dix, A., Finlay, J., Abowd, G. and Beale, R. (1993), *Human-Computer Interaction*, New York: Prentice-Hall.

Dobson, J.E., Martin, M.J., Olphert, C.W. and Powrie, S.E. (1991), Determining requirements for CSCW: the ORDIT approach, In *Collaborative Work, Social Communications and Information Systems*, ed. R.K. Stamper, P. Kerola, and K. Lyytinen, Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Dollimore, J. and Wilbur, S. (1991), Experiences in building a configurable CSCW system, in *Studies in Computer Supported Cooperative Work:Theory, Practice and Design*, ed. J. Bowers and S. Benford, Amsterdam: North-Holland.

Donaldson D.S. (1991), Contracting on a performative network: Using information technology as a legal intermediary, In *Collaborative Work, Social Communications and Information Systems*, R.K. Stamper, P. Kerola, and K. Lyytinen, Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Draper, S. and Oatley, K. (1991), Practical methods for measuring the performance of human-computer interfaces, paper to Joint Councils Initiative on Cog Sci/HCI Summer School *Theory and Methodology of Cognitive Science applied to HCI problems,* Queen Mary & Westfield College, London.

Dubois, E., Hagelstein, J. and Rifuat, A. (1988), Formal requirements engineering with ERAE. *Philips Journal of Research*, 43(3/4): 393-414.

Dunlop, C. and Kling, R. ed. (1991), *Computerization and Controversy: Value Conflicts and Social Choices*, San Diego, CA: Academic Press.

Eason, K. (1988), *Information Technology and Organizational Change*, London: Taylor & Francis.

Ehn, P. (1988), *Work-oriented design of computer artifacts*, Falköping, Sweden: Arbetslivscentrum/Almqvist and Wiksell International.

Ehn, P. (1993a), Scandinavian Design: on participation and skill, in *Participatory Design: Perspectives of systems design*, ed. Schuler, D. and Namioko A., 41-77, Hildsdale, NJ: Lawrence Erlbaum.

Ehn, P. (1993b), Contribution to a panel discussion on The Scandinavian Approaches, In *Proceedings of 16th Conference on Information Systems Research in Scandinavia (IRIS)*, Copenhagen, 7-10 August 1993.

Ehn, P. and Kyng, M. (1984), A tool perspective on design of interactive support for skilled workers, in *Proceedings of the Seventh Scandinavian Research Seminar on Sytemeering*, ed. M Sääksjärri, Helsinki.

Ehn, P., and Kyng, M. (1987), The collective resource approach to systems design, in *Computers and democracy — a Scandinavian challenge,* ed. G. Bjerknes, P. Ehn and M. Kyng, Aldershot, UK: Gower.

Ellis, C.A. (1979), Information Control Nets: A mathematical model of office information systems, in *Proceedings of the 1979 ACM Conference on Simulation, Measurement and Modeling of Computer Systems*, 225-239, Boulder, Colorado: ACM Press.

Ellis, C.A. (1983), Formal and informal models of office activity, in *Proceedings of Information Processing 83. The 9th IFIP World Computer Congress*, ed. R.E.A. Mason, 11-22, Paris: North-Holland.

Ellis, C.A. and Nutt, G. (1980), Office information systems and computer science, *ACM Computing Surveys*, 12(1): 27-60.

Ellis, C.A., Gibbons, R. and Morris, P. (1980), Office streamlining, in *Integrated Office Systems — Burotics*, ed. N. Naffah, 111-125, Amsterdam: North-Holland.

Embley, D.W. and Kurtz, B.D. (1992), *Object-oriented Systems Analysis, A Model-Driven Approac*, Englewood Cliffs, NJ: Prentice-Hall.

Essink, L.J.B. (1986), A modelling approach to information system development, In *Information Systems Design Methodologies: Improving the practice*, ed. T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Falkenberg, E, Nijssen, G.M., Adams, A., Bradley, L., Bugeia, P., Campbell, A.L., Carkeet, M., Lehman, G. and Shoesmith, A. (1983), Feature analysis of ACM/PCM, CIAM, ISAC and NIAM, In *Information Systems Design Methodologies: a feature analysis*, ed. T.W. Olle, H.G. Sol, and C.J. Tully, 169-189, Amsterdam: North-Holland.

Feather, M.S. (1993), Requirements reconnoitering at the juncture of domain and instance, in *Proceedings of RE 93: International Symposium on Requirements Engineering*, 73-76, Jan 4-6, San Diego: IEEE.

Fernstrom, C. (1993), PROCESS WEAVER: adding process support to UNIX, *Second International Conference On The Software Process*, Berlin.

Feyerabend, P. (1978), *Against Method*, New York: Verso.

Fickas, S. (1987), Automating analysis, in *4th International Workshop on Software Specification and Design*, Monterey, CA: IEEE Computer Society.

Fickas, S. and Nagarajan, P. (1988), Critiquing Software Specifications. *IEEE Software*, 5(6): 37-47.

Fickas, S., van Lamsweerde, A. and Dardenne, A. (1991), Goal-directed concept acquisition in requirements elicitation, in *6th Intl. Workshop on Software Specification and Design*, 14-21, Como, Italy: IEEE Computer Society Press.

Finkelstein, A. (1992), Software process modelling: a manifesto, *IOPener* (Newsletter of the IOPT Club for the Introduction of Process Technology) 1(4).

Finkelstein, A., Kramer, B., Nuseibeh, B. and Goedicke, M. (1992), Viewpoints: a framework for integrating multiple perspectives in system development, *International Journal of Software Engineering and Knowledge Engineering*, 2(1): 31-58.

Finklestein, A. and Fuks, H. (1989), Multi-party specification, in *5th Intl. Workshop on Software Specification and Design*, 185-195, Pittsburgh, PA: IEEE Computer Society Press.

Finklestein, A., Kramer, J. and Goedicke, J. K. (1990), Viewpoints oriented software specification, in *3rd Intl. Workshop on Software Engineering and its Applications*, 337-351, Toulouse, France: IEEE Computer Society.

Flores, F. and Ludlow, J.J. (1980), Doing and speaking in the office, In *Decision Support Systems: Issues and Challanges*, ed. G. Fick, and R.H. Sprague Jr., Elmsford, NY: Pergamon Press.

Flores, F., Graves, M., Hartfield, B. and Winograd, T. (1988), Computer systems and the design of organizational interaction, *ACM transactions on Office Information Systems*, 6(2): 153-172

Floyd, C. (1984), *A Comparative Evaluation of System Development Methods*, DDC seminar paper.

Floyd, C. (1987), Outline of a paradigm change in software engineering, in *Computers and democracy — a Scandinavian challenge,* ed. G. Bjerknes, P. Ehn and M. Kyng, Aldershot, UK: Gower.

Floyd, C., Mehl, W-F., Reisin, F-M., Schmidt, G., and Wolf, G. (1989), Out of Scandinavia: Alternative approaches to software design and system development. *Human-Computer Interaction*, 4(3): 253-350.

Fok, L.Y., Kumar, K. and Wood-Harper, T. (1987), Methodologies for socio-technical-systems (STS) development: A comparision, In *Proceedings of the Eighth International Conference on Information Systems*, ed. J.I. DeGross, and C.H. Kriebel, Pittsburgh, PA.

Foucault, M. (1979), *Discipline and punish: The birth of the prison,* New York: Random House.

Frese, M. and Sabini, J. ed. (1985), *Goal directed behaviour: The comcept of action in Psychology*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Friedman, A. and Cornford, D.S. (1989), *Computer Systems Development,* Chichester: John Wiley and Sons.

Friedman, A.L. (1992), Four phases of IT — lessons from the past: insights into the future, paper to the *CRICT Conference on Software and Systems Practice: Social science perspectives*, 1 December 1992, Reading.

Galegher, J., Kraut, R. and Egido, C. ed. (1990), *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work,* Norwood, NJ: Lawrence Erlbaum.

Gane, C. and Sarson, T. (1979), *Structured Systems Analysis:Tools and Techniques*, Englewood Cliffs, NJ: Prentice-Hall.

Garfinkel, H. (1967), *Studies in Ethnomethodology,* Englewood Cliffs, NJ: Prentice Hall.

Gerson, E.M. and Star, S.L. (1986), Analyzing due process in the workplace, *ACM Transactions on Office Information Systems*, 4(3): 257-270.

Gibbons, M. and Gummett, P. (1984), *Science, Technology and Society Today*, Manchester, UK: Manchester University Press.

Glaser, B., and Strauss, A. (1967), *The Discovery of Grounded Theory,* Chicago: Aldine.

Goguen, (1993), Social issues in requirements engineering, in *Proceedings of RE 93: International Symposium on Requirements Engineering*, Jan 4-6, San Diego: IEEE.

Goguen, J. and C. Linde (1993), Techniques for requirements elicitation, in *Proceedings of RE 93: International Symposium on Requirements Engineering*, Jan 4-6, San Diego: IEEE.

Goldkuhl, G. and Lyytinen, K. (1982), A language view of information systems, In *Proceedings of the Third International Conference on Information*, ed. C. Ross and M. Ginzberg, Ann Arbor, MI.

Goldkuhl, G. and Lyytinen, K. (1984), Information system specification as rule reconstruction, In *Beyound Productivity, Information Systems Development for Organizational Effectiveness*, ed. Th.M.A. Bemelmans, Amsterdam: North-Holland.

Green, C., Luckhow, R., Balzer, R., Cheatham, T. and Rich, C. (1986), Report on our Knowledge-based Software Assistant. Readings in Artificial Intelligence and Software Engineering: 377-428.

Greenbaum, J. (1979), *In the Name of Efficiency*, Philadelphia: Temple University Press

Greenbaum, J. and M. Kyng, ed. (1991). *Design at Work: Cooperative Design of Computer Systems*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Greenspan, S. (1984), Requirements Modelling: A Knowledge Representation Approach to Software Requirements Definition. Technical CSRG-135, University of Toronto.

Greenspan, S. and Feblowitz, M. (1993), Requirements Engieering Using the SOS Paradigm, in *Proceedings of RE 93: International Symposium on Requirements Engineering*, 260-263, San Diego, 4-6 January 1993, IEEE.

Greenwood, M. (1992), Using CSP and system dynamics as process engineering tools, in *Proceedings of 2nd European Workshop on Software Process Technology — EWSPT'92*, ed. J.C.Derniame, 138-145, Berlin: Springer-Verlag.

Grief, I. ed. (1988), Computer-supported Cooperative Work: A Book Of Readings, San Mateo, CA: Morgan Kaufmann.

Grindley, C.B.B, (1966), SYSTEMATICS- A non-programming language for designing and specifying commercial systems for computers, *Computer Journal*, August: 124-128.

Grønbæk, K., Grudin, J., Bødker, S., and Bannon, L. (1993), Achieving cooperative system design: shifting from a product to a process focus, Background Paper, COMIC-RISØ-2-2. To appear in *Participatory Design: Perspectives of systems design*, ed. D. Schuler and A. Namioko, Hildsdale, NJ: Lawrence Erlbaum.

Grudin, J. (1989), Why groupware applications fail: problems in design and evaluation, *Office: Technology and People*, 4(3): 245-264.

Grudin, J. (1990a), interface, in *Proceedings of CSCW90 Conference on Computer-Supported Cooperative Work*, 269-278, Los Angeles: ACM Press.

Grudin, J. (1990b), The computer reaches out: the historical continuity of interface design, in *Proceedings of CHI 90,* ACM Press.

Grudin, J. (1991), CSCW: The convergence of two development contexts, In *Proceedings of CHI91*, New Orleans: ACM Press.

Grudin, J. (1992), Utility and usability: research issues and development contexts, *Interacting with Computers*, 4(2): 209-217.

Grudin, J., (forthcoming) Evaluating opportunities for design capture in *Design Rationale,* ed. T. Moran and J.Carroll, Hillsdale, NJ: Lawrence Erlbaum Associates.

Guttag, J.V., Horning, J.J. and Wing, J.M. (1985), An overview of the larch family of specification languages, *IEEE Software*, 2(5): 24-36.

Habermas, J. (1971), *Knowledge and Human Interests*, trans. J.J. Shapiro, Boston: Beacon Press.

Hall, P. (1992), What is process modelling really all about?, *IOPener* (Newsletter of the IOPT Club for the Introduction of Process Technology) 1(3).

Hammer, M. (1982), Improving business performance: the real objective of office automation, in *Proceedings of Office Automation Conference*, San Francisco: 247-254.

Hammer, M. (1990), Reengineering work: don't automate, obliterate, *Harvard Business Review*, July-August.

Hammer, M. and Champy, J. (1993),*Re-engineering the Corporation: A Manifesto for Business Revolution*, Nicholas Brealey Publishing.

Hammer, M. and Kunin, J.S. (1980), Design principles of a specification language, in *Proceedings of AFIPS National Computer Conference*: 541-547.

Hammersley, M. and Atkinson, P. (1983), *Ethnography: Principles in Practice*, London: Routledge.

Harel, D. (1988a), On Visual Formalisms. *Communications of the ACM*, 31(5): 514-30.

Harel, D. (1988b), STATEMATE: A working environment for the development of complex reactive systems, in *Proceedings of Tenth IEEE International Conference on Software Engineering*, Washington, DC: IEEE Computer Society Press.

Harper, R. and Hughes, J.A. (1992), "What a F-ing System! Send em all to the same placeand then expect us to stop em hitting": Making technology work in air traffic control, in *Technology and Working Order: Studies of Work, Interaction and Technology*, ed. G. Button, 127-144, London: Routledge.

Harper, R., Hughes, J.A. and Shapiro, D. (1991), Harmonious Working and CSCW: Computer Technology and Air Traffic Control, in *Studies in Computer Supported Cooperative Work:Theory, Practice and Design*, ed. J. Bowers and S. Benford, Amsterdam, North-Holland.

Hatley, D. and Pirbhai, I. (1987), *Strategies for Real-Time Systems Specificiations*, New York: Dorset House.

Heath, C. and Luff, P. (1992), Collaboration and control: Crisis management and multimedia technology in London Underground line control rooms, *Compute Supported Cooperative WOrk*, 1(1-2): 69-94.

Heimbigner, D., Sutton, S.M. and Osterweil, L. (1990), Managing change in process-centred environments, in *Proceedings of the 4th ACM SIGSOFT Symposium on Software Development Environments: In ACM SIGPLAN Notices.*

Heninger, K.L. (1980), Specifying software requirements for complex systems: new techniques and their applications, *IEEE Trans.actions on Software Engineering*, SE-6(1): 2-13.

Hewitt, C. (1986), Offices are open systems, *ACM Transactions on Office Information Systems*, 4(3): 271-287.

Hill, B., Long, J., Smith, W. and Whitefield, A. (1993), Planning for multiple task work -- an analysis of a medical reception worksystem, in *Proceedings of INTERCHI93 Conference on Human Factors in Computing Systems*. 314-320, Amsterdam: ACM Press.

Hirschheim, R. and Klein, H.K. (1989), 4 paradigms of information-system development,*Communications of the ACM*, 32(10): 1199 — 1216.

Hoare, C.A.R. (1985), *Communicating Sequential Processes*, Prentice-Hall.

Howard, R. (1988), Panel Remarks: CSCW: What does it mean? (Bannon moderator), in *Proceedings of CSCW88*, Portland, OR: ACM Press.

Hughes, J.A. and King, V. (1992a), Paperwork, Working Paper, COMIC-LANCS-4-1.

Hughes, J.A. and King, V. (1992b), Sociology for Large Scale System Design, paper to the *CRICT Conference on Software and Systems Practice: Social science perspectives*, 1 December 1992, Reading.

Hughes, J.A., King, V., Randall, D. and Sharrock, W. (1993), *Ethnography for System Design: A Guide*, Working paper, COMIC-LANCS-2-4.

Hughes, J.A., Randall, D. and Shapiro, D. (1991), CSCW: discipline or paradigm? A sociological perspective, in *Proceedings of the Second European Conference on CSCW-ECSCW91*, ed. L. Bannon, M. Robinson and K. Schmidt, Dordrecht: Kluwer.

Hughes, J.A., Randall, D. and Shapiro, D. (1992), Faltering from ethnography to design, In *Proceedings of CSCW92*, Toronto, Canada: ACM.

Hughes, T.P. (1987), The evolution of large technological systems, in *The Social Construction of Technological Systems*, ed. W. Bijker, T.P. Hughes and T. Pinch, 51-82, Cambridge MA: MIT Press.

Humphrey, W. (1989), *Managing the Software Process*, Reading, MA: Addison Wesley.

Hutchins, E. (1990), The Technology of Team Navigation, in *Intellectual Teamwork: The Social and Technological Foundations of Cooperative Work*, ed. J. Galagher, R.E. Kraut, and C. Egido, 191-221, Hillsdale, NJ: Lawrence Erlbaum Associates.

Iivari, J. (1983), Contributions to the theoretical foundations of systemeering research and the PIOCO model, *Acta Universitatis Ouluensis*, Series A, No. 150, Oulu, 1983

Jacobson, I. (1987), Object-oriented development in industrial environment, in *Proceedings of OOPSLA 87*, 183-191, Orlando, Florida.

Johnson, H. and Johnson, P. (1990), Designer-identified requirements for tools to support task analyses, in *Proceedings of Interact 90*, ed. D.Diaper, D. Gilmore and G. Cockton, Amsterdam: North Holland.

Johnson, H. and Johnson, P. (1991a), Task knowledge structures: psychological basis and integration into systems design, *Acta Psychologica,* 78: 3-26.

Johnson, P. (1992), *Human Computer Interaction*., Maidenhead: McGraw-Hill.

Johnson, P. and Johnson, H. (1991b), Knowledge analysis of tasks: task analysis and specification for human-computer systems, in *Engineering the Human Computer Interface*, ed. A. Downton, Maidenhead: McGraw-Hill.

Johnson, P. and Nicolosi, E. (1990), Task based user interface development tools in *Proceedings of INTERACT'90*, ed. D. Diaper, D.Gilmore, G.Cockton and B.Shackel, Amsterdam: North Holland.

Johnson, P., Wilson, S., Markopoulos, P. and Pycock, J. (1993), ADEPT -- Advanced environment for prototyping with task models, in *Proceedings of INTERCHI93 Conference on Human Factors in Computing Systems*, 56, Amsterdam: ACM Press.

Jones, C.B. (1990), *Systematic Software Development Using VDM*, Prentice-Hall.

Jones, M.R. (1991), Post-industrial and post-Fordist perspectives on information systems, *European Journal of Information Systems,* 1(3): 171-182.

Kaiser, G.E., Feiler, P.H. and Popovich, S.S. (1988), Intelligent assistance for software-development and maintenance, *IEEE Software,* 5(3): 40-49.

Kawalek, P. (1993), Coordinating people and technology through process, in *Proceedings of Information Technology and People*, Moscow, May.

Keen, P.W.G. and Scott Morton, M. (1978), *Decision Support Systems: An organizational perspective*, Reading, MA: Addison-Wesley.

Kelley, C. and Colgan, L. (1992), User modelling and user interface design, in *People and Computers VII, Proceedings of HCI92*, ed. A. Monk, D. Diaper and M.D. Harrison, 227-239, Cambridge, UK: Cambridge University Press.

Kellner, M.I. (1989), *Software Process Modelling: Value and Experience*, Software Engineering Institute, Carnegie Mellon University, Technical Review.

Kensing, F. and Winograd, R. (1991), The language/action approach to design of computer-support for cooperative work: A preliminary study in work mapping, in *Collaborative Work, Social Communications and Information Systems*, R.K. Stamper, P. Kerola, and K. Lyytinen, Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

Kerola, P. and Järvinen, P. (1975), Systemointi II, Helsinki, Oy Gaudeamus Ab

Kieras, D.E. and Polson, P.G. (1985), An approach to the formal analysis of user complexity, *International Journal of Man-Machine Studies*, 22(4): 365-394.

Kling, R. (1980), Social analyses of computing: Theoretical perspectives in recent empirical research, *ACM Computing Surveys*, 12(1): 61-110.

Kling, R. (1991), Cooperation, coordination and control in computer-supported work, *Communications of the ACM*, 34(12), 83-88.

Kling, R. and Dunlop, C. (1992), Key Controversies About Computerisation and White Collar Work Life, in *Computer-Human Interaction*, ed. R. Baeker, W. Buxton and J. Grudin, San Mateo, CA: Morgan Kaufman.

Knudsen, T. *et al*. (1993), The Scandinavian Approaches: Theories in Use, of Use and Organization of Interdisciplinarity – a panel introduction, in *Proceedings of the 16th Conference on Information Systems Research in Scandinavia (IRIS)*, ed. J.P. Bansler *et al,* 29-38, 7-10 August 1993, Copenhagen: University of Copenhagen, Department of Computer Science.

Kotonya, G. and Sommerville, I. (1992), Viewpoints for requirements definition, *Software Engineering Journal*, 7(6): 375-387.

Kraft, P. (1979), The Routinizing of Computer Programming, *Sociology of Work and Occupations*, 6(2): 139-155.

Kraft, P. and Bansler, J. (1992) The collective resource approach: the scandinavian experience, in *PDC92: Proceedings of the Participatory Design Conference*, ed. M. Muller, S. Kuhn, and J. Meskill, 127-135, 6-7 November, 1992, MIT, Cambridge, MA.

Krzanik, L. (1989), Enactable models for quantitive evolutionary software processes, *ACM SIGSOFT Software Engineering Notes*, 14(4): 103-111.

Kuhn S. and Muller M.J. (1993), Introduction to the special issue on Participatory Design. *Communications of the ACM*, 36(6): 24-28.

Kuhn, S. (1989), The limits to industrialisation: computer software development in a large commercial bank, in *The Transformation of Work?: skill, flexibility and the labour process*, ed. S. Wood, London: Unwin Hyman.

Kuhn, T. (1970), *The Structure of Scientific Revolutions*, Chicago: University of Chicago Press.

Kuutti, K. (1993) Information systems and the development of work activity, COMIC-OULU-2-6

Kuutti, K. and Bannon, L. (1993), Searching for unity among diversity: exploring the interface concept, in *Proceedings of InterChi 93*. Amsterdam: ACM.

Kyng, M. (1993) Reply to Kraft and Bansler. Electronic mail message circulated to all IRIS 16 participants, August 1993.

Kyng, M. and Matthiassen, L. (1982), Systems development and trade union acitivites, in *Information Society, for Richer for Poorer,* ed. N. Bjørn-Andersen, Amsterdam: North Holland.

Landauer, T.K. (1991), Lets get real: a position paper on the role of cognitive psychology in the design of humanly useful and useable systems, in *Designing Interaction: Psychology at the Human-Computer Interface,* ed. J. Carroll, Cambridge, UK: Cambridge University Press.

Landes, D. (1972), *The Unbound Prometheus*, Cambridge, UK: Cambridge University Press.

Langefors, B. (1966), *Theoretical Analysis of Information Systems*, Lund, Sweden: Studentlitteratur.

Langefors, B. (1974), Information systems, Information Processing 74, 937-945, Amsterdam: North-Holland.

Lappo, P. (1992), Process modelling a quality system, *IOPener* (Newsletter of the IOPT Club for the Introduction of Process Technology) 1(5).

Latour, B. (1990), Drawing things together, in *Representation in Scientific Practice*, ed. M. Lynch and S. Woolgar, London: MIT Press.

Layton, E.T. (1978), Millwrights and engineers: science and social roles, and the evolution of the turbine in america, in *The Dynamics of Science and Technology: Social Values, Technical Norms, and Scientific Criteria in the Development of Knowledge*, ed. W. Krohn, E.T. Layton, and P. Weingart, Dordrecht: D. Reidel.

Lehtinen, E. and Lyytinen, K. (1988), Action based model of information system, *Information Systems*, 11(4).

Leite, J.C.S.P. (1989), Viewpoint analysis: a case study, *ACM Journal of Software Engineering Notes,* 14(3): 111-119.

Lim, K. Y. and Long, J. (1992), Rapid prototyping, structured methods and the incorporation of human factors into system development, in *Proceedings of of East-West International Conference on Human-Computer Interaction — EWHCI 92*, St. Petersburg, Russia.

Luff, P., Heath, C. and Greatbatch, D. (1992), Tasks-in-interaction: paper and screen based documentation in collaborative activity, in *Proceedings of CSCW'92*, Toronto, Canada: ACM Press.

Lynch, M. and Woolgar, S. ed. (1990), *Representation and Scientific Practice*, London: MIT Press.

Lyotard, J-F., (1984), *The Postmodern Condition. A Report on Knowledge*, trans. Billington, G. and Massumi, B., Minneapolis: Minneapolis University Press,

Lyytinen, K., (1987), Different perspectives on information systems: problems and solutions, *ACM Computing Surveys*, 19(1): 5-46

Macaulay, L. (1993), Requirements as a cooperative activity, in *Proceedings of RE 93: International Symposium on Requirements Engineering*, Jan 4-6, San Diego: IEEE.

Macaulay, L., Fowler, C., Kirby, M. and Hutt, A. (1990), USTM: a new approach to requirments specification, *Interacting with Computers*, 2(1): 92-118.

Macaulay, L., O'Hare,G., Viller, S. and Dongha, P. (1993), Cooperative requirements capture, in *Proceedings of the 1993 JFIT Technical Conference*, Keele University: DTI.

MacKenzie, D. and Wajcman, J. ed. (1985), *The Social Shaping of Technology*, Milton Keynes, UK: Open University Press.

Maclaren, R., Hornby, P., Robson., J., OBrien, P., Cleg, C., and Richardson, S., (1993)*System Design Methods — The Human Dimension*, unpublished manuscript.

MacLean, A., Young, R., and Moran, T. (1989), Design rationale: the argument behind the artifact, in *Proceedings of CHI'89*, ed. K. Bice and C.S. Lewis, 247-252, New York: ACM Press.

Malone, T.W. (1987a), Modelling coordination in organizations and markets, *Management Science*, 33(10).

Malone, T.W. (1987b), Computer support for organizations: towards an organizational science, in *Interfacing Thought: Cognitive Aspects of Human Computer Interactions,* ed. J.M. Carroll, Cambridge, MA: MIT Press.

March, J. (1991), How decisions happen in organizations *Human Computer Interaction*, 6: 95-117.

Marcuse, H. (1964), *One Dimensional Man*, Boston: Beacon Press.

Martin, J. and Odell, J.J. (1992), Object-oriented Analysis and Design, Englewood Cliffs, NJ: Prentice-Hall.

Mathews, B. and Ryan, K. (1989), Requirements specifications using conceptual graphs, in *Proceedings of Third Intl. Workshop on Computer Aided Software Engineering*, 186-193, London, UK.

McGraw, K.L. and Harbison-Briggs, K (1989), *Knowledge Acquisition*, Englewood Cliffs, NJ: Prentice-Hall International.

McMenamin, S. and Palmer, J. (1984), *Essential Systems Analysis*, Englewood Cliffs, NJ: Prentice-Hall.

Medina-Mora, R., Winograd, T., Flores, R. and Flores, F. (1992), The Action Workflow approach to workflow management technology, in *Proceedings of CSCW'92*, ed. J.Turner and R.Kraut, 281-288, Toronto, Canada: ACM Press.

Milner, R. (1989), *Communication and Concurrency*, London: Prentice-Hall

Moore, A. P. (1990), The specification and verified decomposition of system requirements using CSP, *IEEE Transactions on Software Engineering*, 16(9): 932-948.

Moran, T. (1981), The command language grammar: a representation for the user interface of interactive computer systems, *International Journal of Man-Machine Studies*, 15(1): 3-50.

Moran, T. and Anderson, R. (1990), The workaday world as a paradigm for CSCW design, in *Proceedings of CSCW'90*, 381-393, Los Angeles: ACM Press.

Morgan, G. (1986), *Images of Organization*, London: Sage.

Mullery, G. P. (1979), A method for controlled requirements specifications, in *Proceedings of 4th Intl. Conference on Software Engineering*, 126-135, Munich, Germany: IEEE Computer Society.

Mumford, E. (1971), *Systems Design for People*, Manchester: NCC.

Mumford, E. (1983), *Designing Human Systems for New Technology*, The ETHICS method, Manchester: Manchester Business School.

Mumford, E. and Henshall, D. (1979), *A Participative Approach to Computer System Design*, London: Associated Business Press.

Mumford, E. and Weir, M. (1979), *Computer Systems in Work Design — The ETHICS method*, London: Associated Business Press.

Musa, J.D. ed. (1983), Stimulating software engineering progress: a report of the software engineering planning group, *ACM Software Engineering Notes*, 8: 29-54.

Norman, D.A. (1986), Cognitive Engineering, in *User Centered System Design*. ed. D. Norman and S. Draper, 31 — 61, Hillsdale, NJ: Lawrence Erlbaum Associates.

Norman. D. and Draper, S. ed. (1986), *User Centred System Design*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Nygaard, K. and Bergo, O. (1975), The Trade Unions – new users of research, *Personnel Review*, 4(2): 5-10.

Olerup, A. (1989), Socio-technical design of computer-assisted work: A discussion of the ETHICS and Tavistock approaches, *Scandinavian Journal of Information Systems*, 1.

Olson, G.M.. and Olson, J.S. (1991), User-centred Design of Collaboration Technology, *Organisational Computing*, 1(1): 61-83.

Orr, K. (1981), *Structured Requirements Definition*, Topeka, Kansas: Ken Orr and Associates.

Osterweil, L. (1987), Software processes are software too, in *Proceedings of the 9th International Conference on Software Engineering*, 2-13, Monterey, CA: IEEE Computer Society Press.

Osterweil, L. (1989), Automated support for the enactment of rigourously described software processess, *ACM SIGSOFT Software Engineering Notes*, 14(4): 122-126.

Ould, M.A. (1992), Process modelling with RADs, *IOPener* (Newsletter of the IOPT Club for the Introduction of Process Technology) 1(5).

Page, D., Williams, P. and Boyd, D. (1993), *Report of the Inquiry into the London Ambulance Service*, February, ISBN. 0 905133 70 6, Communications Directorate, South West Thames Regional Health Authority, London, UK.

Parnas, D., and Clements, P. (1986), A rational design process: how and why to fake it, IEEE *Transactions on Software Engineering*, 12: 251-257.

Pava, C. (1983), *Managing New Office Technology, An organizational strategy*, New York: The Free Press.

Pava, C. (1986), Redesigning sociotechnical systems design: Concepts and methods for the 1990s, *The Journal of Applied Behavioral Science*, 22(3): 201-221.

Payne, S.J. and Green, T.R.G. (1986), Task-action grammars: a model for the mental representation of task languages, *Human-Computer Interaction*. 2(2): 93-133.

Perrolle, J. (1991), Intellectual assembly lines: the rationalization of managerial, professional and technical work, *Computers and the Social Sciences*, 2(3): 111-122.

Pfaff, G.E. (1985), *User Interface Management Systems*, New York: Springer Verlag.

Pressman, R.S. (1992), *A Managers Guide to Software Engineering*, MCGraw-Hill.

Procter, R.N. and Williams, R.A. (1992), HCI: Whose problem is IT anyway?, paper to the *CRICT Conference on Software and Systems Practice: Social science perspectives*, 1 December 1992, Reading.

Quintas, P. and Millar, J. (1992), Towards a framework for innovation and diffusion research in software development, paper to the *CRICT Conference on Software and Systems Practice: Social science perspectives*, 1 December 1992, Reading.

Reisner, P. (1981), Formal grammars and human factors design in an interactive graphics system, *IEEE Transactions on Software Engineering*, 45.

Resnick, L.B., Levine, J.M. and Teasley, S.D. ed. (1991), *Perspectives on Socially Shared Cognition*, Washington, DC: American Psychological Association.

Reubenstein, H.B. and Waters, R.C. (1991), The requirements apprentice: automated assistance for requirements acquisition, *IEEE Transactions on Software Engineering*, 17(3): 226-240.

Rich, C., Waters, R.C. and Reubenstein, H.B. (1987), Toward a requirements apprentice, in *Proceedings of 4th Intl. Workshop on Software Specifications and Design*, 439-446, Monterey, CA: IEEE Computer Society Press.

Rittel. H. and Webber, M. (1973), Dilemmas in a general theory of planning, *Policy Sciences*, 4(2): 155-169.

Robertson, R.E. (1993) What to do with a human factor: a manifesto of sorts, in *American Center for Design Studies*, 7(1): 63-73.

Robinson, M. (1991a), Double-level languages and cooperative working, *AI & Society*, 5: 34-60.

Robinson, M. (1991b), Position paper for formalisation panel, In *Proceedings of the Second European Conference on CSCW — ECSCW91*, ed. L. Bannon, M. Robinson, and K. Schmidt, Dordrecht: Kluwer.

Robinson, M. and Bannon, L. (1991), Questioning representation, in *Proceedings of the 2nd European Conference on CSCW — ECSCW'91*, ed. L. Bannon, M. Robinson and K. Schmidt, 219-233, Amsterdam: Kluwer.

Rose, M. (1988), *Industrial Behaviour: Research and Control*, Harmondsworth: Penguin.

Ross, D. (1977), Structured Analysis: a language for communicating ideas, *IEEE Transactions on Software Engineering*, 3(1): 16-34.

Royce, W., (1970), 'Managing the development of large software systems', *Proceedings of WesCon*.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991), *Object-Oriented Modelling and Design*, Prentice-Hall.

Ryan, K. (1989), Intelligent support for jackson method, in *Proceedings of 3rd Intl. Workshop on Computer Aided Software Engineering*, 194-207, London, UK.

Schael, T and Zeller, B. (1993), Tutorial Description, for *Third European Conference on CSCW-ECSCW 93*, Milan.

Schäfer, G. *et al* (1988), *Functional Analysis of Office Requirements: A multiperspective approach*, Chichester: John Wiley & Sons.

Schlaer, S. and Mellor, S.J. (1988), *Object-Oriented Systems Analysis: Modelling the World in Data*, Englewood Cliffs, NJ: Yourdon Press.

Schmidt, K. (1991a), Cooperative work: a conceptual framework, in *Distributed Decision Making: Cognitive Models for Cooperative Work,* ed. J. Rasmussen, B. Brehmer, and J. Leplat, 75-109, Chichester: Wiley.

Schmidt, K. (1991b), Riding a tiger, or computer supported cooperative work, in *Proceedings of the Second European Conference on CSCW — ECSCW91*, ed. L. Bannon, M. Robinson, and K. Schmidt, Dordrecht: Kluwer.

Schmidt, K. (1993), The Sociological Bonanza?, Working Paper, COMIC-RISØ-2-3.

Schmidt, K. and Bannon, L. (1992), Taking CSCW seriously: supporting articulation work, *CSCW*, 1(1-2): 7-40.

Schmidt, K. and Carstensen, P. (1993), Bridging the Gap: Requirements Analysis for System Design, Working Paper, COMIC-RISØ-2-2.

Schmidt, K. and Rodden, T. (1993), Putting it all together: Requirements for a CSCW platform, in *Design of Computer Supported Cooperative Work and Groupware Systems. 12th International Workshop on Informatics and Psychology,* Schärding, Austria.

Schuler, D. and Namioko A. ed. (1993), *Participatory Design: Perspectives of systems design*, Hildsdale, NJ: Lawrence Erlbaum.

Schutz, A. (1962), Common — sense and scientific interpretation of human action, in *Collect ed Papers,* Volume 2, ed. M. Natanson, The Hague: Martinus Nijhoff.

Scott Morton, M. ed. (1991), *The Corporation of the 1990s: Information Technology and Organizational Transformation,* New York: Oxford University Press.

Searle, J.R. (1969), *Speech Acts, An Essay in the Philosohy of Language*, Cambridge, UK: Cambridge University Press.

Searle, J.R. (1979), *Expression and Meaning, Studies in the Theory of Speech Acts*, Cambridge, UK: Cambridge University Press.

Searle, J.R. and Vanderveken, D. (1985), *Foundations of Illocutionary Logic*, Cambridge, UK: Cambridge University Press.

Shapiro, D. (1993), Ferrets in a sack? Ethnographic studies and task analysis in CSCW, presented at the *12th Schaerding International Workshop on Design Of Computer Supported Cooperative Work And Groupware Systems*, June 1-3, Elsevier Press.

Shapiro, D. (1993), The Boundaries of Design for CSCW, Working Paper, COMIC-LANCS-2-5.

Shapiro, D., Harper, R., Hughes, J.A.. and Randall, D. (1991), Visual Re-representation of data base information: The flight strip in air traffic control, *Co-Tech Working Group Conference*, 21-23 May, Scharding.

Sharrock., W. and Anderson, R. (forthcoming) Organisational innovation and the articulation of the design process, in *Design Rationale*, ed. T. Moran and J. Carroll, Hillsdale, NJ: Lawrence Erlbaum Associates.

Shlaer, S. and Mellor, S. J. (1988), *Object-Oriented Systems Analysis: Modeling the World In Data*, Englewood Cliffs, NJ: Yourdon Press/Prentice-Hall.

Shneiderman, B. (1987), *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Reading, MA: Addison-Wesley.

Shotter, J. (1991), The rhetorical-responsive nature of mind: a social constructionist account in *Against Cognitivism,* ed. A. Still and A. Costall, London: Harvester Wheatsheaf.

Simon, H.A. (1960), *The New Science of Management Decision*, New York: Harper and Row.

Sirbu, M., Schoichet, S., Kunin, J. and Hammer, M (1981), *OAM: An Office Analysis Methodology*, Laboratory for Computer Science, MIT, Cambridge, MA.

Skidmore, S., Farmer, R. and Mills, G. (1992), *SSADM Version 4 Models and Methods*, Manchester: NCC Blackwell.

Sommerville, I. (1992), *Software Engineering*, 4th ed. New York: Addison-Wesley.

Sommerville, I., Rodden, T., Hughes, J.A. (1993), Requirements Engineering for Cooperative Systems, Working Paper, COMIC-LANCS-2-2.

Sommerville, I., Rodden, T., Sawyer, P. and Bentley, R. (1992), Sociologists can be surprisingly useful in interactive systems design, in *People and Computers VII: Proceedings of the HCI'92 conference*, ed. A. Monk, D. Diaper and M. D. Harrison, 341-353, York: Cambridge University Press.

Spivey, J. M. (1988), *Understanding Z: A Specification Language and Its Formal Semantics*, Cambridge, UK: Cambridge University Press.

Suchman, L. (1983), Office procedures as practical action: models of work and system design, *ACM Transactions on Office Information Systems*, 1(4): 320-328.

Suchman, L. (1987), *Plans and Situated Action: The problem of Human — Machine Communication*, Cambridge, UK: Cambridge University Press.

Suchman, L. (1993), Do categories have politics, in *Proceedings of the Third European Conference on Computer Supported Cooperative Work: ECSCW93*, ed. G. de Michaelis, C. Simone and K. Schmidt, 1-14, Dordrecht: Kluwer.

Suchman, L., and Wynn, E., (1984), Procedures and problems in the office, *Office Technology and People*, 2: 133-154.

Taylor, B. (1990), The HUFIT planning analysis and specification toolset, in *Human-Computer Interaction — INTERACT'90*, ed. D. Diaper, D.Gilmore, G.Cockton and B.Shackel, 371-376, Amsterdam: North Holland.

Trist, E.L., Higgin, G.W., Murray, H. and Pollock, A.B. (1963), *Organizational Choice*, London: Tavistock.

Turner, S. (1992), The strange life and hard times of general theory in sociology, in *Postmodernism and Sociological Theory,* ed. S. Seidman and D. Wagner, Oxford: Blackwell.

Twidale, M., Rodden, T. and Sommerville, I. (1993), The designers notepad: Supporting and understanding cooperative design, in *Proceedings of ECSCW93*, ed. G. De Michelis, C. Simone and K. Schmidt, 93-108, Milan: Kluwer.

Vera, A. and Simon, H. (1993), Situated action: a symbolic interpretation, *Cognitive Science*, 17: 7-49.

Viller, S.A. (1993), The group facilitator: A CSCW perspective, in *Readings in Groupware and Computer Supported Cooperative Work*, ed. R.Baeker, 145-152, San Mateo, CA: Morgan Kaufmann.

von Bulow, I. (1990), The bounding of a problem situation and the concept of a system's boundary in soft systems methodology, *Journal of Applied Systems Analysis*, 16: 35-41.

Warboys, B. (1990), The IPSE 2.5 project: process modelling as the basis for a support environment *First International Conference On Software Development, Environments and Factories*, Berlin.

Ward, P. and Mellor, S. (1985), *Structured Development for Real-Time Systems*,, Englewood Cliffs, NJ: Prentice-Hall.

Weber, M. (1930), *The Protestant Ethic and the Spirit of Capitalism*, trans. T. Parsons, London: George Allen and Unwin.

Weber, M. (1949), *The Methodology of the Social Sciences*, trans. E.A. Shils and H.A. Finch, New York: The Free Press.

Welke, R.J. (1977), Current information system analysis and design approaches: framework, overview, comments and conclusions for large-complex information system education, In *Education and large information systems*, ed. R.A. Buckingham, Amsterdam: North-Holland.

Whitaker, R., Essler, U., and Östberg, O. (1991), Participatory Business Modeling, Research Report, TULEA 1991:31, Luleå University, Sweden.

White, P. (1992), Process modelling and process support, *IOPener* (Newsletter of the IOPT Club for the Introduction of Process Technology) 1(3).

Wilden, A. (1980), *System and Structure*, London: Tavistock.

Wilkinson, R. T. (1990), Software requirements specification for the commodity paging system, *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, 457-499, Los Alamitos, CA: IEEE Computer Society Press.

Wilson, B. (1981), *Systems: Concepts, Methodologies, and Applications*, Chichester: John Wiley & Sons.

Wilson, P. (1991), *Computer Supported Coopersative Work: An Introduction*, Oxford: Intellect Books.

Wilson, S., Markopoulos, P., Pycock, J. and Johnson, P. (1992), Modelling perspectives in user interface design, in *Proceedings of of East-West International Conference on Human-Computer Interaction — EWHCI 92*, St. Petersburg, Russia.

Winner, L. (1980), Do artifacts have politics?, *Daedalus*, 109(1): 121-36.

Winograd, T. and Flores, F. (1986), *Understanding Computers and Cognition, A new Foundation for Design*, Norwood, NJ: Ablex.

Wirfs-Brock, R., Wilkerson, B. and Wiener, L. (1990), *Designing Object-Oriented Software*,, Englewood Cliffs, NJ: Prentice-Hall.

Wittgenstein, L. (1974), *On Certainty*, Oxford: Blackwell.

Woo, C. and Lochovsky, F.H. (1986), Supporting distributed office problem solving in organisations, *ACM Transactions on Office Information Systems*, 4(3): 185-204.

Wood, S. (1989), *The Transformation of Work*, London, Unwin Hyman.

Wood-Harper, A.T., Antill, L. and Avison, D.E. (1985), *Information Systems Definition: the Multiview Approach*, Oxford: Blackwell.

Woolgar, S. (1981), Interests and explanation in the social study of science, *Social Studies of Science*, 11(3): 365-394.

Woolgar, S. (1988), *Science:The Very Idea*, London: Tavistock.

Woolgar, S. (1991), Configuring the user: the case of usability trials, in *A Sociology of Monsters: Essays on Power, Technology and Domination*, ed. J. Law, London: Routledge.

Young, J.W. Jr. and Kent, H.K. (1958), Abstract formulation of data processing problem, *Journal of Industrial Engineering*, 1958(Nov-Dec): 471-479.

Young, R.M., Green. T.R.G. and Simon, T. (1989), Programmable user models for predictive evaluation of interface design, in *Proceedings of of CHI 89*, Reading, MA: Addison-Wesley.

Yourdon, E. (1982), *Managing the System Life Cycle. A Software Methodology Overview*, New York: Yourdon Press.

Yourdon, E.N. (1989), *Modern Structured Analysis*, Prentice-Hall.

Zave, P. (1982), An Operational approach to requirements specification for embedded systems, *IEEE Transactions on Software Engineering*, 8(3): 307-320.

Zave, P. and Shell, W. (1986), Salient features of an exceutable specification language and its environment, *IEEE Transactions on Software Engineering,* 12(2): 312-325.

Zetterberg, H. (1965), *On Theory and Verification in Sociology*., 3rd revised and enlarged edition, New Jersey: Bedminster Press.

Zimmerman, D. (1970), The practicalities of rule use, in *Understanding Everyday Life,* ed. J. Douglas, 221- 38, Chicago: Aldine.

Zisman, M.D. (1977), *Representation, Specification and Automation of Office Procedures*, Dept. of Decision Science, The Wharton School, University of Pennsylvania.