

User-developer cooperation in
software development:
building common ground and
usable systems

Eamonn Joseph O'Neill



QUEEN MARY
AND WESTFIELD COLLEGE
UNIVERSITY OF LONDON

1998

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy



IMAGING SERVICES NORTH

Boston Spa, Wetherby
West Yorkshire, LS23 7BQ
www.bl.uk

**PAGINATED BLANK PAGES
ARE SCANNED AS FOUND
IN ORIGINAL THESIS**

**NO INFORMATION IS
MISSING**

He that uses many words for the explaining of any subject,
doth, like the cuttle fish, hide himself for the most part in his own ink.

John Ray, *On the Creation*, 1691

I spent four years prostrate to the higher mind,
Got my paper and I was free.

Emily Saliers, *Closer to Fine*, 1989

Abstract

The topic of this research is direct user participation in the task based development of interactive software systems. Building usable software demands understanding and supporting users and their tasks. Users are a primary source of usability requirements and knowledge, since users can be expected to have intimate and extensive knowledge of themselves, their tasks and their working environment. Task analysis approaches to software development encourage a focus on supporting users and their tasks while participatory design approaches encourage users' direct, active contributions to software development work. However, participatory design approaches often concentrate their efforts on design activities rather than on wider system development activities, while task analysis approaches generally lack active user participation beyond initial data gathering. This research attempts an integration of the strengths of task analysis and user participation within an overall software development process.

This thesis also presents detailed empirical and theoretical analyses of what it is for users and developers to cooperate, of the nature of user-developer interaction in participatory settings. Furthermore, it operationalises and assesses the effectiveness of user participation in development and the impact of user-developer cooperation on the resulting software product. The research addressed these issues through the development and application of an approach to task based participatory development in two real world development projects. In this integrated approach, the respective strengths of task analysis and participatory design methods complemented each other's weaker aspects. The participatory design features encouraged active user participation in the development work while the task analysis features extended this participation upstream from software design activities to include analysis of the users' current work situation and design of an envisioned work situation.

An inductive analysis of user-developer interaction in the software development projects was combined with a theoretical analysis drawing upon work on common ground in communication. This research generated an account of user-developer interaction in terms of the joint construction of two distinct forms of common ground between user and developer: common ground about their present joint development activities and common ground about the objects of those joint activities, work situations and software systems.

The thesis further extended the concept of common ground, assessing user participation in terms of contributions to common ground developed through the user-developer discourse. The thesis then went on to operationalise and to assess the effectiveness of user participation in terms of the assimilation of users' contributions into the artefacts of the development work. Finally, the thesis assessed the value of user participation in terms of the impact of user contributions to the development activities on the usability of the software produced.

Table of Contents

Acknowledgements	13
Chapter 1: Introduction	15
1.1 Background	15
1.2 Problem statement	16
1.3 Research approach	18
1.4 Results and contributions made	19
1.5 Outline of chapters	20
Chapter 2: A longitudinal view on software development and participation	23
2.1 Computer systems development: changing constraints	24
2.1.1 Phase I systems development	24
2.1.2 Phase II systems development	25
2.1.3 Phase III systems development	28
2.2 Participatory design	30
2.3 Giving the users what they want: effective HCI	38
2.3.1 Evolution in HCI	39
2.3.2 Task Analysis and Participatory Design	40
2.4 Where are we now?	44
2.4.1 Listening to developers	48
2.4.2 We are here	50
2.5 Research questions raised	63
Chapter 3: The projects and their analysis	67
3.1 The development method	68
3.2 The development projects studied	74
3.2.1 Pilot project overview	75
3.2.2 CI project overview	81
3.2.3 CS project overview	90
3.3 Data available from the projects	95
3.4 Research method	96
3.4.1 Analysing interaction	102
3.4.2 Approaches to the research questions	109
3.4.3 Issues in performing video based analysis	112
3.4.4 On researcher involvement in the projects	116

Chapter 4: Towards a theory of user-developer cooperation	119
4.1 Developing interaction analysis	124
4.1.1 Identifying process in cooperative development	125
4.1.2 Exploring artefacts and relations in cooperative development	133
4.1.3 Process revisited	151
4.2 The construction of shared understandings	160
4.3 Cooperative development and Common Ground	166
4.4 Conclusion	176
Chapter 5: Effective participation: contributing to discourse and artefacts	181
5.1 The nature of participation: contributing to discourse	182
5.1.1 Making a contribution	182
5.1.2 Types of contribution	188
5.2 Were the studied projects participatory?	189
5.2.1 Expected user contributions to user-developer collaboration	190
5.2.2 Methodology in assessing contributions	195
5.2.3 Observed user contributions to user-developer collaboration	197
5.2.4 Conclusions on participation	201
5.3 Effective participation: contributing to artefacts	205
5.3.1 Contributions and evidence of effectiveness	206
5.3.2 Effectiveness of user participation in the projects studied	210
5.4 Conclusion	220
Chapter 6: User participation and software usability	223
6.1 Evaluating the software	224
6.1.1 Recruiting subjects	228
6.1.1.1 CI subjects	228
6.1.1.2 CS subjects	229
6.1.2 Defining tasks	229
6.1.2.1 CI tasks	229
6.1.2.2 CS tasks	230
6.1.3 Running evaluations	231
6.1.3.1 CI evaluation	231
6.1.3.2 CS evaluation	232
6.2 Summary of the usability evaluation results	233
6.2.1 Summary of the CI evaluation results	234
6.2.2 Summary of the CS evaluation results	236
6.3 Tracing relationships between contributions, software features and usability	238
6.3.1 Tracing upstream	239
6.3.2 Tracing downstream	240
6.4 Results of tracing analysis	242

6.4.1 Results of upstream tracing	242
6.4.2 Results of downstream tracing	245
6.4.2.1 Results of tracing downstream from work analysis activity	246
6.4.2.2 Results of tracing downstream from work design activity	251
6.4.2.3 Results of tracing downstream from software design activity	260
6.5 Discussion	272
6.5.1 Deus ex machina	273
6.5.2 Technology	275
6.5.3 Standard flow	276
6.5.4 Ineffective contributions	278
6.5.5 Unvalued contributions	279
6.5.6 Unrequested features	280
6.5.7 False absence	281
6.6 Conclusion	282
Chapter 7: Task based cooperative development: conclusions	287
7.1 Thesis summary	287
7.2 Lessons and limitations	290
7.2.1 Building theory	290
7.2.2 On research methodology	295
7.2.3 Improving practice	297
7.3 Future work	303
References	307
Appendix 1: CI1 transcript	325
Appendix 2: CS1 transcript	355
Appendix 3: CS5 transcript	361
Appendix 4: CS7 transcript	367
Appendix 5: CS usability evaluation materials	373
Appendix 6: Questionnaire	383

List of Figures

Figure 2.1: Waterfall model of software development	27
Figure 2.2: Research Questions	66
Figure 3.1: Part of a task model on a whiteboard from the CS project	72
Figure 3.2: Analysis component cards in the pilot project	77
Figure 3.3: Part of the task model constructed in the pilot study	78
Figure 3.4: Initial task model sketch from CI project	83
Figure 3.5: Part of a task model on a wallchart from the CI project	84
Figure 3.6: Annotations from a cooperative task modelling session	85
Figure 3.7: Key tasks identified from TM1 in the CI project	87
Figure 3.8: Prepared user interface components for CI paper prototyping work	88
Figure 3.9: Task list for a user performing the indexer role in the CI system	89
Figure 3.10: A task model produced on a whiteboard in the CS project	92
Figure 3.11: Key task-workflow model from the CS project	93
Figure 3.12: Paper prototype using Post-It™ notes in the CS project	94
Figure 3.13: Standard experimental approach	97
Figure 3.14: Observational approach	99
Figure 3.15: Space of systems development research methods	101
Figure 3.16: Locating the research methods of the thesis	102
Figure 3.17: Generic ESDA process	105
Figure 3.18: Three intellectual traditions in ESDA	107
Figure 4.1: Extended instantiation of ESDA process for research question one	120
Figure 4.2: Minneman's [1991] framework of group design interactions	121
Figure 4.3: Schematic representation of the cyclic interaction analysis method	122
Figure 4.4: Software development activities	126
Figure 4.5: Structure of development work activities	128
Figure 4.6: Simple models and relations in work analysis activity	135
Figure 4.7: External shared model as support for and product of interaction	138
Figure 4.8: Simple models and relations in work design activity	138

Figure 4.9: Simple models and relations in software design activity	139
Figure 4.10: Simple models and relations in requirements analysis activity	141
Figure 4.11: Models and relations in current work analysis activity	145
Figure 4.12: User's difficulty with paper prototype dimensions	150
Figure 4.13: Simple structure of development work activities	151
Figure 4.14: Model mismatch monitoring	155
Figure 4.15: Interaction processes within development activities	156
Figure 4.16: User and developer use a task model as a medium for analysis	167
Figure 5.1: Contributing to discourse	184
Figure 5.2: Contributing to common ground	186
Figure 5.3: Successful and effective contributing	187
Figure 5.4: Six areas of knowledge in user-developer communication	193
Figure 5.5: Expected relative frequencies of contributions from users and developers	195
Figure 5.6: Contributions in CS1 meeting to analyse the users' current work situation	197
Figure 5.7: Contributions in CS5 meeting to design a new work situation	198
Figure 5.8: Contributions in CS7 meeting to design a new software system	199
Figure 5.9: Aggregate contributions across three CS meetings	200
Figure 6.1: CI software usability issues at each level of severity	234
Figure 6.2: CS software usability issues at each level of severity	237
Figure 6.3: Sources of CS software usability issues at each level of severity	244
Figure 6.4: Sources of CI software usability issues at each level of severity	245
Figure 6.5: Embodiment of CS1 contributions in subsequent artefacts	250
Figure 6.6: Embodiment of CS5 contributions in subsequent artefacts	256
Figure 6.7: Invoking a 'refer problem' dialogue in the CS paper prototype	258
Figure 6.8: Submitting a query in the CS software	259
Figure 6.9: Embodiment of CS7 contributions in subsequent artefacts	268
Figure 6.10: Customer details fields in the CS paper prototype	270
Figure 6.11: Customer details fields in the CS software	271
Figure 6.12: Contribution routes into software	273

Acknowledgements

Sincere thanks are due to several people who have helped, guided and supported me in the course of this research.

Thanks to my supervisors Prof Peter Johnson and Prof George Coulouris for first giving me the opportunity to begin this work and for then giving me the skills, guidance and experience to finish it.

The HCI Group at QMW has been a wonderful place to be. So many people here have contributed to my enjoyment and learning. Thanks in particular to Hilary Johnson and Tilde Bekker for reading this stuff when it was still forming and helping it on its way.

Mick Smith, Reg Towse and the people at Harlequin deserve my thanks for putting up with a researcher in their midst and, of course, for funding me during this research in partnership with the Engineering and Physical Sciences Research Council.

Anonymous thanks to all those who took part in my projects or answered my questions.

Thanks to Karen for all her help.

Chapter 1

Introduction

This thesis is a study of user participation in the development of interactive software based systems. The literature of interactive software development increasingly promotes user-developer cooperation as a desirable, even necessary, aspect of software development practice. This research first asks how commonly this theme has been taken up in practice by software developers. It then goes on to examine user-developer cooperation in two real world software development projects. It describes the interaction between user and developer in a cooperative development setting, deriving the beginnings of a theory of this interaction. It also examines users' contributions to software systems development and their influence on the artefacts of the development process, including the software produced.

The thesis provides a view of modern software development practice, illustrates how users may participate in a task based approach to software development and presents analyses both of the nature of that participation and its effects on the development process and products.

This chapter presents the background to the research, identifies the problems which the research addresses and the reasons for tackling these problems. It introduces the methodological approaches which the research adopted, presents a brief statement of the research results and outlines the remaining chapters of the thesis.

1.1 Background

In its relatively brief history, software systems development has moved through three identifiable stages [Friedman, 1989]. In the first phase, roughly from the inception of software in the 1940s until the mid 1960s, hardware limitations and costs were the main constraint on systems development. In the second phase, roughly from the mid 1960s until the early 1980s, software complexity became the main constraint on development. In the last of these phases, up to the present, 'user relations' became the main constraint on systems development. That is, system quality problems arising from inadequate understanding and meeting of user requirements. Friedman [1989] suggests three possible strategies for addressing the problems of understanding and designing for user requirements. Of these three

strategies, the approach of promoting user-developer cooperation in systems development has become the most widely recommended route.

User-developer cooperation has been most strongly encouraged in the Scandinavian influenced approach of participatory design (PD). This approach grew out of trade union efforts to promote workplace democracy and from earlier work on sociotechnical design [Mumford, 1987; Mumford, Land and Hawgood, 1978]. PD has focused on techniques such as prototyping to involve users in software design work. However despite increasingly widespread promotion of PD in the interactive software development literature, empirical studies of the real world practice of software developers suggest that the techniques of participatory design are not widely used by practising developers (see chapter two).

User relations issues remain the main constraint in current systems development practice and continue to become more and more significant as types of applications and types of users continue to expand. With the ever increasing importance of relations between user and computer, mediated through the software, has come the expansion of the field of Human-Computer Interaction (HCI). The focus of HCI has tended to shift from perceptual and basic cognitive issues and user interface components to interest in higher cognitive issues such as task planning and execution and human-computer dialogues [Grudin, 1990]. Further movement in this direction has moved the focus of much HCI work to wider task support and work organisation issues.

By now it is received wisdom, at least amongst the academic community, that systems must be built with a clear focus on users and their tasks [Gould and Lewis, 1985]. Task analysis (TA), as well as PD, has been frequently touted in the software development literature, particularly the HCI literature, as an approach to tackling 'user relations' issues. TA, however, lacks a strong reported history of application in substantial real world development projects. On the other hand, while applications of PD techniques are reported frequently, this often is in the context of research projects rather than real world systems development.

1.2 Problem statement

Despite the popularity of both TA and PD within the academic community, there has been very little research effort on systematically integrating task analysis and user participation with each other and with the overall system development process. Both PD and TA approaches often lack consistent user participation across the range of system development activities. Most of the reported techniques of participatory design do indeed focus on design which is, however, only one of many processes which comprise system development. While PD approaches often pay inadequate attention to analysis activities, TA approaches generally do not actively involve users beyond initial data gathering and, sometimes, checking artefacts produced by the developers.

The PD literature lacks detailed analyses of what it is for users and developers to cooperate, of the nature of user-developer interaction in PD settings. Furthermore, there has been little systematic analysis of the *effectiveness* of user participation in such settings. The PD literature is also weak on analyses of the impact of specific features of user-developer cooperation on the resulting implemented systems (i.e. the value of user participation). The literature tends to report PD (i.e. *design*) activities and then give at best a vague assessment of the software produced, without systematically relating features of the PD activities to features of the software (see chapter two).

Despite having been around, in various forms, for many years (see [Diaper, 1989]), task analysis has remained a strongly academic approach to software development, with few attempts at applying a task based approach in real world software development projects. Many professional software development practitioners claim to use both PD and TA. However, empirical studies (see chapter two) suggest that practising developers rarely pay more than lip service to these approaches.

If practitioners are to use such a technique, they must know in detail what it involves and must be convinced of its value. If participatory methods - in all phases of systems development projects - are to be taken up by professional developers, the latter must be clear what these methods consist of and what effects they have. And before recommending such methods, researchers should first gain a more thorough understanding of their practice and results.

This research set out to address these issues, motivating both theoretical and practical contributions to the field. The work reported here attempted to develop an integrated approach to task based participatory development which extended the focus on users' tasks downstream from early analysis activities and extended active user participation in development activities upstream from prototype design. Further, this approach was applied in two substantial real world development projects. In addition to these contributions to practice, this work provides a detailed analysis of user-developer interaction in systems development meetings, laying the groundwork for a theoretical model of user participation (chapter four). Chapter five reports an analysis of user contributions to cooperative development work and the effectiveness of those contributions. Chapter six relates usability features of the implemented systems to user contributions in cooperative development activities.

The work reported here was guided by four research questions which captured the fundamental concerns of the research. The first, and perhaps most fundamental, question asked: what *is* user participation? In posing this question, the thesis considers the processes and activities in which users and developers were collaboratively involved, examining how the participants, especially the users, contributed to those processes and activities.

The second research question asked whether the development projects studied actually were participatory. That is, did the users' presence during development

work result in active contributions by users to the development process; or were the users just used by developers as a resource to be tapped; or was the users' presence simply irrelevant to the developers' performance of the development activities?

The third research question asked: in so far as there was active user participation in the development activities, how *effective* was this participation? In considering this question, it was necessary to operationalise effectiveness in the context of participatory development. Chapter five derives and uses such a definition in terms of the assimilation of users' contributions into the artefacts, both external and internal, of the development process.

The fourth research question asked how valuable user contributions were to the development process. Again, it was necessary to operationalise *valuable* in this context. Chapter six uses a definition of the value of a contribution in terms of its influence on the features of the implemented software to trace relations between specific user contributions to the development process and specific usability issues with the resulting software in each development project.

1.3 Research approach

In order to address the above issues, an integrated approach to task based participatory development was developed and applied across the projects studied. This approach, labelled CUSTARD (Cooperation with USers in Task Analysis for Requirements and Design) [O'Neill, 1995; O'Neill, Johnson and Coulouris, 1995], supported user participation in development activities from initial discussions, through modelling the users' current work situation, modelling envisioned work situations and software design. The approach is of interest in itself, but within this thesis only in so far as it provided the opportunity to address the issues and research questions presented above.

A combination of the research questions and the difficulties inherent in studying real world software development projects suggested an observational, rather than experimental, approach. Having adopted an observational approach, the research drew on both social and cognitive research traditions in choosing and applying specific research techniques.

The approaches adopted in tackling each of the four research questions were based on analysing the interaction between user and developer in the cooperative development settings. The first research question was addressed through an analysis of user-developer interaction in cooperative development meetings across the projects. This analysis was based primarily on video records of the development meetings. Minneman's [1991] framework of group design interaction was used to focus the analysis on three aspects of the cooperative development activities: artefacts, processes and relations (see chapter four). As the analysis

developed, it produced a theoretical account of user-developer cooperation in systems development as the joint construction of common ground.

The second research question asked whether the development projects studied actually were participatory. Chapter five offers a definition of participation in terms of actively contributing to the development discourse and builds on the analysis of chapter four to extend Clark and Schaefer's [1989] notion of how contributions are made in the cooperative development setting. Having operationalised the notion of a contribution to development activity, chapter five assesses user participation through the types and numbers of contributions made.

In tackling research question three, the work reported in chapter five then goes on to operationalise the *effectiveness* of the contributions in terms of their assimilation into the artefacts, both external and internal, of the development process. This assimilation is assessed for the contributions made across the sampled video records.

The fourth research question asked how valuable user contributions were to the development process. Chapter six defines the value of a contribution in terms of its influence on the implemented software. In answering research question four, the work reported in chapter six traces in both directions, upstream and downstream, between contributions to development activities and features of the software's usability. These complementary tracing analyses provide an assessment of the influence of user contributions to development activities on the resulting software.

1.4 Results and contributions made

The thesis explores what happens when user and developer are brought together, with a view to supporting and improving their interaction, and how this may contribute to developing highly usable systems. HCI is a young discipline which has yet to establish its scope and methods so concretely as other disciplines with a longer history. This is apparent in

- (i) the continuing expansion of its remit to include more and more areas of research interest, such as participatory design, computer-supported cooperative work and virtual reality;
- (ii) the continuing expansion of the methods and techniques it adopts from other disciplines, such as mathematics, psychology and sociology;
- (iii) the continuing absence of a systematic theoretical base for much of the work.

This research in part at least addresses all three of these points. It examines an area which in recent years has been swallowed up by the HCI conurbation: Participatory Design (PD) or cooperative software development. In investigating this area, this research applies and integrates a wide range of research methods and techniques to

the analysis of extensive and heterogeneous data. From its analysis, this research attempts, amongst other contributions, to provide theoretical analyses of the nature of user-developer interaction in cooperative systems development and the relationship between user participation and software usability.

Overall, this thesis makes several practical and theoretical contributions. Practically, it developed an approach to task based cooperative software development and applied this approach in two real world development projects. In doing this, it contributed to the development of PD's focus on analysis activities and integration into the overall system development process, contributed to the still meagre stock of reported real world applications of systematic task based development and contributed to the practical integration of TA and PD. The thesis also provides an evaluation of the practical impact of user participation in development on the resulting software.

From a more theoretical perspective, this research provides a rich description of the interaction between user and developer in a cooperative development setting and how users contribute to participatory software development projects. It generates a theoretical account of the nature of PD in terms of the construction of shared understandings or common ground amongst users and developers.

The thesis then goes on to present an analysis of the nature and effectiveness of users' contributions to system development work in a cooperative setting. This effectiveness is assessed in terms of the assimilation of users' contributions into the artefacts of the development process. The thesis provides an example of integration between cognitive and social accounts of human activity.

Finally, the thesis traces the influence on software usability of user participation in the development activities. The analysis examines the relationship between users' contributions to the development activities and usability features of the resulting software. Hence, this work goes some way to explaining what it is to 'do' cooperative development, describing the application of one form of such development and assessing its results in terms of the development artefacts and the software product.

This work raises many questions, not all of which it answers. It is the purpose of an exploratory analysis to generate issues and questions and to frame them within the beginnings of a theoretical context. This work does that, thereby moving the field of HCI a little further from craftwork to scientific discipline.

1.5 Outline of chapters

Chapter two sets the scene for the rest of the thesis. It begins with an overview of software development across the past five decades and introduces PD as the most strongly developed approach to promoting user-developer cooperation in systems development.

The chapter outlines the movement of HCI as a field from an early focus on basic cognitive issues and the components of user interfaces, to a more recent focus on wider task support and collaborative work issues. Chapter two presents the first results of the thesis research with a survey of software systems development practice.

Chapter three describes the software development projects which were studied, the development approach which was adopted and the approach taken to the analysis of these projects. The chapter begins with a description of the task based participatory development approach which was used. It then provides an overview of the two major software development projects studied and of a short pilot study. The chapter summarises the data which were available for analysis from the projects and discusses the research method adopted.

Chapter four addresses research question one: *what is cooperative software development?* This research question was addressed through an analysis of user-developer interaction in cooperative development meetings across the projects.

The chapter presents the results of initial interaction analysis cycles which provided the foundation of an account of user-developer interaction and which suggested common ground [Clark, 1996] as a key theoretical insight into the nature of cooperative development work. The work reported in chapter four goes on to apply Clark's notion of common ground to the developing account of user-developer interaction. This analysis resulted in the generation of a theoretical account of user-developer interaction in terms of the construction of two distinct types of common ground between user and developer.

Chapter four reports a 'macro analysis' while chapter five, in contrast, presents a 'micro analysis', which by its nature is very time consuming and, therefore, more selective. In tackling research questions two and three, the work reported in chapter five develops and applies a framework for assessing user participation in software development in terms of the extent and the effectiveness of user contributions to software development activities.

Chapter five provides a definition in terms of making contributions of what it is for users actively to participate in software development. The first half of the chapter examines user contributions to development meetings in the studied projects.

The second half of chapter five presents a definition of the effectiveness of user participation in terms of the assimilation of users' contributions into the artefacts of the development work. The chapter then examines this assimilation into the artefacts of the development process in the projects studied.

Chapter six addresses research question four, examining the value of user participation in terms of its impact on the usability of the resulting software systems. The chapter traces the relations between user participation and software

usability in two directions, forward from development work contributions to features of the software and backward from features of the software to contributions.

Finally, chapter seven concludes the thesis, summarising the contributions and lessons from previous chapters, assessing the research and pointing towards future work.

Chapter 2

A longitudinal view on software development and participation

This chapter sets the scene for the rest of the thesis. Section 2.1 presents a brief overview of software development across the past five decades. This follows Friedman's [1989] account of three identifiable phases in software development from the 1950s through to the 1990s. In the last of these phases, Friedman identifies 'user relations' as the main constraint on systems development and suggests three possible strategies for addressing the problems of understanding and designing for user requirements. Of these three strategies, the approach of promoting user-developer cooperation in systems development has become the most widely recommended route.

Section 2.2 introduces the most strongly developed approach to promoting user-developer cooperation in systems development: the Scandinavian influenced approach of PD. The section discusses PD's focus on design and the relative lack of well integrated techniques for participatory analysis.

Section 2.3 looks at the growth of HCI as a field addressing the issues of user relations. The section briefly outlines the movement in HCI from a focus on basic cognitive issues and the components of user interfaces, through a focus on higher cognitive issues such as task planning and execution and human-computer dialogues to a focus on wider task support and collaborative work issues. In the course of the latter part of this movement, the domain of HCI has spread to include the issues of interest in PD. However, despite extensive literatures both on task based development and on PD, we have yet to see substantive work on integrating user participation with a task based approach to systems development.

Section 2.4 begins by reporting empirical studies of the real world practice of systems developers. These studies suggest that the techniques of task analysis and participatory design are not widely used by practising developers. The section then reports the results of a survey conducted as part of this research to assess this suggestion.

Section 2.5 reviews this chapter and provides a statement of the research questions which are raised by it.

2.1 Computer systems development: changing constraints

Friedman [1989] presents a history of information systems development which posits three phases from the beginnings of electronic computing to the end of the 1980s. These phases are distinguished by the prevailing constraint on systems development and the spread of computerisation in each phase. Friedman [1989] analyses this history through four models: the position of computer systems development activities within user organisations; management strategies for controlling the development process; the agents of change (e.g. available technology) in the first two models; and the spread of computers and applications to and within organisations. Friedman [1989] identifies three phases dominated respectively by:

- (1) hardware constraints: hardware costs and limitations of hardware capacity and reliability;
- (2) software constraints: productivity of systems developers, difficulties of delivering reliable systems on time and within budget;
- (3) user relations constraints: system quality problems arising from inadequate perception of user demands and inadequate servicing of their needs.

All three constraints have applied in each of the three phases. However, in phase I problems of hardware capacity, reliability and cost were the primary constraint on the development and spread of computerisation. The transition from phase I to phase II came when this constraint had been eased sufficiently (though never wholly eliminated) to reveal software constraints as the primary limitation. Similarly, as hardware and software constraints continued to ease, there was a transition to phase III. This transition, to a phase in which meeting user requirements became the overriding problem, was encouraged also by changes in the types of systems being developed, the types of users and the types of uses.

2.1.1 Phase I systems development

Friedman [1989] identifies the period from the birth of computing in the late 1940s to the mid 1960s as phase I. During this period, 'the important issues in the minds of those who were responsible for selecting a computer were how big is it and how fast will it go? The reason for this emphasis on hardware performance was that the cost of the hardware represented a very high proportion of the cost of the total application' [Friedman 1989, p.73]. However, hardware costs fell throughout this period. Friedman notes that the price of an 'average' computer fell by 75% between 1953 and 1964. Similarly, Boehm [1973] estimates that

hardware costs for the US Air Force fell from 80% of the total costs of an application in 1955 to 50% by 1965.

At the same time, capacity and reliability of both central and peripheral components of the hardware improved throughout this period. Friedman [1989] quotes estimates that between 1956 and 1965 the cost-effectiveness of tape drives and card readers improved by a factor of ten, line printers by a factor of three and CPU and memory by a factor of 400.

From the late 1950s, operating systems became increasingly important until, by 1964 and the IBM 360 series, what the manufacturers were selling was no longer just the hardware but an integrated system of hardware and (operating system) software. This trend continued with the large scale development and sale of increasingly complex application software.

Friedman [1989] summarises the main characteristics of phase I systems as batch systems, operated within organisations' central data processing departments, with no direct user interface to the computer. They were used for cost-saving clerical applications such as accounting, payroll, stock control and sales processing. These were mainly independent replications of existing manual processes. Exceptionally, there were a few on-line real-time systems such as airline reservation and missile defence systems.

The continual easing of the constraints on hardware costs and performance had several important consequences. The number of viable applications grew, computer systems began to integrate previously independent functions and to provide functions which had simply not been available with manual systems. Demand for skilled developers increased markedly above supply, resulting in high staff turnover and the average experience of developers remaining low.

By the mid 1960s, the increasing complexity of systems whose development was attempted, combined with the shortage of experienced developers, resulted in software taking over from hardware as the primary constraint on the further development and spread of computerisation. Software development projects regularly failed to deliver and when they did, they were often over budget and late. According to Friedman [1989], this 'software crisis' marks the transition from phase I to phase II.

2.1.2 Phase II systems development

Awareness of the software crisis had been building in the academic and military-industrial communities at least since the early 1960s (see [Greenberger, 1962]). However, it was openly and generally acknowledged for the first time at the NATO sponsored conference in Garmisch-Partenkirchen in October 1968. The main recommendation to come out of this conference was the adoption of systems engineering methods in the development of large systems development projects.

These methods were: standardisation of program structures via modularisation; specific ordering of the stages in systems development; standardisation of design and programming procedures. According to Friedman [1989], these recommendations were aimed mainly at improving management control over systems developers who were seen as unproductive, over powerful and out of touch with users and their own employers.

The concept of software engineering also predates the Garmisch conference. Canning [1956] was one of the first to suggest the application of systems engineering to the development of computer systems. He defined the following process for the development of a computer-based data processing system.

- 1 Systems study to specify requirements
- 2 Preliminary design
- 3 Detailed design
- 4 Programming/acquisition/installation

As described by Agresti [1986], this model was refined throughout the late 1950s and 1960s. For example, Laden and Gildersleeve [1963] identified the following stages in the systems development 'life-cycle' model.

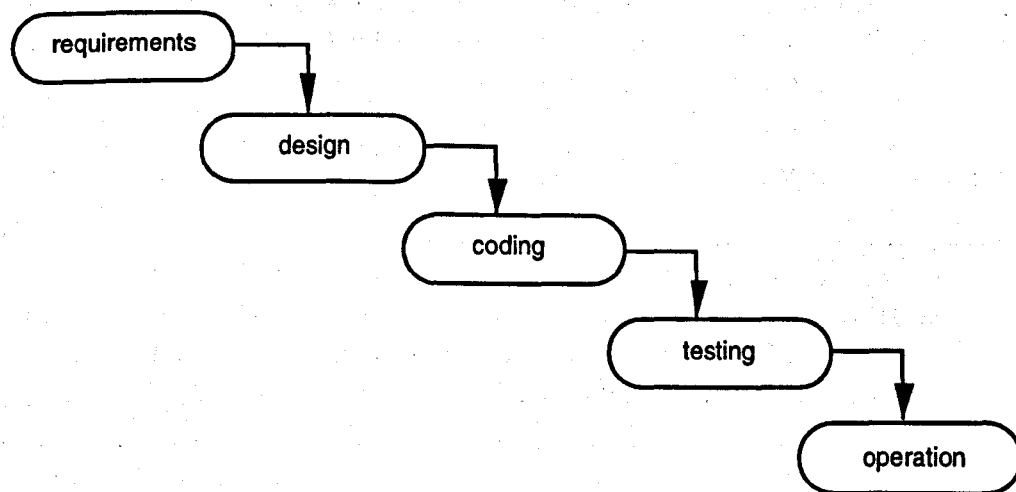
- 1 Survey (feasibility)
- 2 System investigation (definition of objectives)
- 3 System design (relationships between all parts of the system)
- 4 Programming
- 5 File making
- 6 Preparation of clerical procedures (documentation)
- 7 Program testing

These systems engineering models were not primarily intended as descriptive of how software was being developed. They were from the beginning *prescriptive* of how software engineering theorists believed software should be developed. They had strong influence through the dominance of the underlying systems engineering paradigm on which they are based. This paradigm had its roots in general systems theory and operational research which developed during the 1940s as part of the scientific war effort in logistics and planning. The general systems engineering model typically has the following stages.

1. Information gathering and problem identification. A need is perceived, background information is gathered on this perceived need and an initial assessment of the economic criteria is made. The objectives of the project are then identified.
2. Systems analysis and modelling. The 'problem-solving' phase in which mathematical modelling techniques are often used to simulate and to explore alternative means to the ends identified in 1, with a means chosen based on a trade-off amongst viable alternatives.
3. Design. This stage involves the design of the engineered system (e.g., a manufacturing plant) which will realise the chosen means.
4. Implementation. The system designed at the previous stage is constructed.
5. Operation. The implemented system begins operation (e.g., the plant produces its first product). The operation is monitored and, if necessary, modified to ensure that the system runs as expected.

Perhaps the classic systems engineering model of software development from the phase II period is Royce's [1970] 'Waterfall model' (see Figure 2.1).

Figure 2.1: Waterfall model of software development



By 1972, Bauer could begin his preface to an advanced course in software engineering with the claim that 'it is not necessary to start with a definition of Software Engineering. ... In 1967 and 1968, the word "Software Engineering" [*sic*] has been used in a provocative way in order to demonstrate that something was wrong in the existing design, production and servicing of software. The situation has considerably changed since then; many people show concern about the problems of software engineering and some of the manufacturers, to which the provocation was mainly addressed, claim that they already obey the principles of

software engineering, whatever this may mean' [Bauer, 1977; 1972 preface]. Bauer's lack of concern explicitly to define the meaning of 'Software Engineering' may be explained by the pervasiveness of the paradigm underlying his thinking. Software engineering had very quickly been accepted by theoreticians, within both academia and large manufacturing organisations, as the way to build software.

During the quarter century since Bauer wrote his preface, systems engineering has continued to be applied to the development of computer systems, both hardware and software. Software Engineering models have dominated academic and corporate thinking on software development. The original Waterfall model has been refined many times. Tools and techniques to aid software production have been constructed within this paradigm. The past quarter century has also seen massive improvements in computer hardware. This success illustrates the strengths of the systems engineering approach when the problem and the solution can be clearly defined and agreed upon and a readily identifiable sequential movement can be made through subgoals to the solution.

2.1.3 Phase III systems development

Friedman [1989] argues that the methods and techniques which were developed and applied during phase II did not entirely solve the software crisis but did ease the problems. Factors such as greater experience with computer systems development in user organisations, more realistic ambitions of systems developers, advancing hardware and software technology (notably high level programming languages) and standardised development methods contributed to the delivery of somewhat more reliable systems at more acceptable cost and occasionally even on schedule.

The easing of hardware and software constraints allowed the already present user relations constraints to come to the fore. Friedman [1989] places the transition from phase II to phase III at the beginning of the 1980s. This transition, to a phase in which understanding and meeting user requirements became the overriding constraint, was encouraged by changes in the types of systems being developed, the types of users and the types of uses. The organisation of systems developers made difficult the understanding and meeting of user needs. During the 1960s and 1970s, developers had become concentrated in 'data processing' (DP) departments within large user organisations. Throughout this period, DP workers became more and more isolated from the workers in other departments of the organisation who were the users of the systems which the DP department provided.

As user experience with computer systems increased, so did their demands for ever more complex applications. As systems grew beyond mere automated replications of previously manual tasks, systems began to be judged on how useful they were in supporting users in doing their jobs and in achieving their overall

ambitions. The sheer complexity of systems and requirements was constantly increasing. Common applications were changing to less structured, more management and planning oriented functions. Consequently, user requirements became more unstable and more difficult to specify. At the same time, users became more knowledgeable, more powerful and more likely to express dissatisfaction.

Writing in the late 1980s, Friedman [1989] noted a number of strategies commonly pursued during phase III for dealing with what he calls user relations problems, that is the problems of meeting user requirements for useful and usable work support. These strategies fall within three broad approaches: allowing users to cross the user-developer barrier; allowing developers to cross the user-developer barrier; and directly breaking down the user-developer barrier.

The first approach described by Friedman [1989] involves what he calls 'end-user computing'. By this he means that users take over the roles of developing or, at least, customising their own applications. Friedman describes this as an old dream within computing. In the early days of computing, most users did have to write their own applications. In the late 1990s, for most users it remains largely a dream. Apart from very limited user customisation facilities in mass market applications and user development in a few specialist applications, software development has remained largely the task of the professional developer. For a discussion of ways in which user development might be supported, see [Henderson and Kyng, 1991].

The second approach to dealing with the user relations problem is for developers to become more sensitive to users, better equipped to understand their needs and more flexible in response to those needs. Friedman [1989] suggests that this strategy was adopted by many organisations in the 1980s. At an individual level, this involved developers' recruiting so called 'renaissance people', non-computer specialists with educational backgrounds in disciplines such as business studies and humanities. How much more effective such an education makes people in determining user requirements is a matter of some doubt. In any case, as Friedman [1989] notes several times, no matter what an individual's background, once she has been recruited into a systems development department or group, she quickly aligns with and becomes a part of the developer culture, so losing the renaissance touch.

The third approach to dealing with the user relations problem is to 'concentrate on directly breaking down the [user-developer] barrier by creating arenas where internal interaction between computer specialists and users can occur' [Friedman, 1989, pp.301-2]. Friedman notes that during the 1980s, user-developer cooperation in systems development began to be seen as a panacea, with 'writers contributing to most of the lines of literature ... coalescing around this recommendation for solving user relations problems' [Friedman, 1989, p.271].

2.2 Participatory design

Since the 1980s, academic interest in user-developer cooperation has continued to grow. The literature on approaches to and techniques for user-developer cooperation has blossomed. The first major conference solely devoted to direct user participation in systems development was held in Denmark in 1985. Floyd [1987] reflected the tone of the conference, distinguishing between the established software engineering paradigm, which she described as 'product oriented', and an alternative 'process oriented' paradigm of software development. The software engineering paradigm focuses on the *product* in design and use. Floyd's alternative paradigm focuses on the *processes* of design and use. Floyd argued that within the established software engineering paradigm, user relations issues 'must be considered as additional aspects outside the realm of systematic treatment' [Floyd, 1987, p.195]. A product oriented approach views a software product as an object derived by formalised procedures and 'the user organisation as static, with the interaction between the person and the computer as something fixed and predefined, and the designer appearing as an outside observer' [Bannon and Bødker, 1991, p.248]. In contrast, a process oriented approach views organisations as continuously changing, prompting corresponding changes in human computer interaction. The software designer necessarily becomes involved in these changes and 'design is a learning and change process for all the involved parties, both designers and users' [Bannon and Bødker, 1991, p.248].

The notion of paradigm shift is due to Kuhn [1962]. He describes a paradigm as a shared framework of theories, exemplars and practical norms which forms the basis of scientific research. Research consists in interactions between the paradigm and the aspects of the world to which it is relevant. A paradigm's imperfect modelling of the world permits anomalies between the paradigm and real world results. However, in the long term a mass of unresolved anomalies calls into question the utility of the paradigm. New paradigms may be proposed. Kuhn [1962] argues that 'scientific revolutions' occur when a new paradigm is generally accepted and support for an old paradigm collapses. One of the clearest examples of this revolutionary paradigm shift is the 'Copernican Revolution' in which the scientific community's mainstream view of the physical Universe changed radically.

Paradigm shift may, however, be less revolutionary and not entirely displace the old paradigm. This may happen when a new paradigm is accepted as resolving anomalies apparent in the old, while the old still has considerable utility. An example of this is when Einsteinian Relativity Theory was generally accepted as resolving anomalies within classic Newtonian mechanics. The latter did not disappear from widespread use because it remained valid except in the extreme

conditions which produced the anomalies resolved by Relativity Theory. It is this type of evolutionary paradigm shift which Floyd [1987] advocates.

As described above, established software engineering has its roots firmly in the systems engineering paradigm. Naughton [1984] discusses a potential shift from this 'hard' to a 'soft' systems paradigm. The soft systems paradigm has perhaps its fullest expression in Soft Systems Methodology (SSM) [Checkland, 1981; Checkland and Scholes, 1990]. SSM was developed from research on the application of systems engineering to what Ackoff [1974] has called a 'mess'. A mess is a situation of several, interacting problems, i.e. a system of problems. Ackoff argued that no problem ever exists in isolation and solution of one problem may in turn cause other problems. A mess is, therefore, particularly unsuitable for resolution by systems engineering with its emphasis on developing a system based on a clear, unitary problem definition.

Floyd [1987] identified the limitations of the hard software engineering paradigm in analysing and designing to meet 'messy' user requirements. However, she failed to distinguish between a hard approach to what Checkland [1981] calls 'the situation of concern' - in this case, the users' work situation - and a hard approach to the management of the software development project. As noted by Friedman [1989], one of the main reasons for the success of software engineering in easing the problems of the software crisis was improved management control over development projects.

Software development is both a technical task and a social process. Similarly, the work for which the software support is being developed is both. Historically, both the development work and the work situation developed for have been treated, in the hard systems engineering approach, as technical tasks. A paradigm shift may be needed to an approach which treats both the development work and the users' work situation more as social processes, while retaining a distinction between a hard approach to the management of development projects and a soft approach to understanding and designing for users' work requirements. The work of this thesis concentrates on the latter. A strength of work such as MUSE [Lim and Long, 1994], for example, lies in its potential for combining soft approaches to the situation of concern with hard approaches to project management.

The location of the 1985 conference in Denmark reflected a strong Scandinavian influence on the movement towards directly involving users in software development work. The Scandinavian school of Cooperative or Participatory Design¹ (PD) represented at the conference emerged from Scandinavian legislation and research in the 1970s and early 1980s (e.g. [Bødker, Ehn, Kammersgaard, Kyng, and Sundblad, 1987]). The Scandinavian codetermination laws of the 1970s were intended to increase worker influence in decisions

¹The terms are often used interchangeably. The author prefers to refer to cooperative development.

affecting their working environment and practices. With the support of this legislation, trade unions sponsored several projects which examined how workers might exert most influence on the development of computer systems for the workplace (e.g. [Nygaard and Bergo, 1975]).

These projects and fresh legislation covering workers' influence on the introduction of new technology into the workplace continued through the early 1980s. Trade union sponsored action research emphasised the participation of the intended users, i.e. the workforce, in computer systems development. This 'collective resource approach' emphasised that 'in democratisation of design and use of new technology in Scandinavia, trade unions - especially on local level - should play an active role' [Ehn and Kyng, 1987, p. 38; *sic*]. Thus, the motivating concern of the collective resource approach was not for improvement in the usefulness and usability of the resulting systems *per se*, but for trade union influence and 'workplace democracy'. While this concern does provide the approach with a clear and much needed focus on the social and political context of a systems development project, it does not necessarily promote usability of the resulting system (although it may enhance its acceptability to the users).

The collective resource approach grew out of earlier attempts to apply the sociotechnical approach pioneered by the Tavistock Institute (see [Mumford, 1993]). Proponents of the collective resource approach criticised applications of the sociotechnical approach for only superficially democratising the development process [Ehn and Kyng, 1987; Floyd, Mehl, Reisin, Schmidt and Wolf, 1989]. Despite this criticism, the collective resource school continued to borrow from user participation methods developed within the sociotechnical approach.

During the 1980s, the collective resource approach grew from encouraging trade union influence on systems development projects to developing participatory design techniques. Again this was conducted primarily through action research such as the oft cited UTOPIA project [Ehn, 1989]. A fundamental tenet of these projects was that 'the design of computer support for labour processes calls for professional experience and know-how, both with respect to the labour process in which the computer is to be used and to the field of computer technology and its design. The design of computer support for labour processes must therefore be carried out with the users; it cannot be performed either for or by them' [Floyd *et al*, 1989, p.288].

Throughout the 1980s and 1990s, work has been done on encouraging and supporting the active involvement of users in software development activities. Introducing a book which reviewed the state of the Scandinavian influenced 'cooperative design' work at the beginning of the 1990s, Greenbaum and Kyng [1991] state that 'although the identification of user issues now dominates the computer management and system development literature, the majority of books, articles and seminars addresses the issue of how best to "integrate the user" into

the system development process. ... Our interest is not in fitting users into an already existing system development process, but in creating new ways of working together. For us, user participation does not mean interviewing a sample of potential users or getting them to rubber stamp a set of system specifications. It is, rather, the active involvement of users in the creative process we call design'.

There is by now a large and growing body of PD literature, with a biennial conference on PD and many publications in other conferences and journals. PD has adopted from its inception an action research approach and much of this literature provides accounts of PD methods and techniques being applied in system development projects. Despite the oft avowed goal of understanding the users, their tasks and work situation, much of the published work on user involvement has discussed techniques and tools for 'participatory' or 'cooperative' *design* of new computer systems, with little focus on what by analogy may be termed cooperative analysis. Hence, a large part of the PD literature describes the use of design mock-ups (e.g. [Bødker *et al*, 1987; Bødker, Ehn, Knudsen, Kyng and Madsen, 1988; Ehn and Kyng 1991]) and software prototypes (e.g. [Thoresen, 1993]).

For example, Bødker describes six design situations from three projects. 'Common to the projects was that they made use of a prototyping design strategy, where the users took part in constructing and evaluating the prototypes' [Bødker, 1991, p.58]. Ehn and Kyng also describe the use of mock-ups in the UTOPIA project, stating that 'with cardboard mock-ups it's simple: the purpose is *design* and the mock-ups are used to evaluate a design, to get ideas for modifications or maybe even radical new designs, and to have a medium for collaborative changes' [Ehn and Kyng, 1991, p.192; emphasis added]. Again, Bødker and Grønback [1991] describe an approach to 'cooperative prototyping'.

Within PD approaches, old problems with prototyping remain. As Muller [1993, p.213] notes, 'software rapid prototyping technology contains implicit politics'. The developer and user do not have equal access to or expertise in the software technology which is the vehicle for prototyping. Thus, the developer can much more readily express her ideas in the prototype. The user often can incorporate her ideas in the prototype only through negotiation with and interpretation by the developer. 'The prototype system reflects the developer's interpretation of the user's needs' [Carey and Mason, 1983, p.51].

The developer often has a large investment of programming time and effort in the software prototype and may be tempted to interpret the user's ideas in ways which protect that investment. Therefore, 'the user is dependent upon a software professional whose personal or organisational agenda may be quite different from the user's. As a result, the user's ideas may be distorted, and the social or organisational effort required from the user to correct the distortion may be prohibitive. The user may thus be alienated from the design process and from the

artefact produced by the process' [Muller, 1993, p.213]. Attempting to reduce this dependence by training a user in the software development technology can lead to her adopting in part a developer's perspective and no longer being representative of her erstwhile colleagues (see [Bødker *et al*, 1987]).

Paper based mockups may be less intimidating to users than software based prototypes. Mockups, however, tend, like software prototypes, to be 'owned' by the developer. Nielsen [1990, p.316] describes that 'a paper mockup consists of a series of screen designs only, and needs a human to "play computer" and present the screens to the user in the right order according to the user's (simulated) actions'. Paper mockups, therefore, gain little over software based prototypes. The screens are produced by the developer and presented to the user by the developer 'playing computer'. Feedback from the user does not directly change the design, but is mediated through and interpreted by the developer, perhaps by creating modified screen mockups to present again to the user. In addition, paper mockups lack some of the strengths which software based prototypes have in simulating the real time behaviour of the system in interaction with the user. Neither does the use of mockups remove the difficulty associated with software prototyping of representing the usability requirements which emerge in a form which may readily be integrated into the overall system design.

Friedman [1989, p.271] reported prototyping as the PD technique which gained greatest currency throughout the 1980s. Grønæk, Kyng and Mogensen [1993] note the inconsistent coverage of development activities in PD and its strong focus on prototyping, proposing a development method which they claim extends cooperative development into the implementation phase. There remains, however, relatively little literature on 'participatory analysis' methods or techniques (but see [Mogensen and Trigg, 1992]) or reports of PD based projects which perform in depth cooperative analyses before leaping into prototyping activities. Kjær and Madsen [1995] and Grønæk and Mogensen [1994] do present rare examples of participatory analysis work being practised.

Muller, Wildman and White [1993] present a taxonomy of PD practices which includes many of the prototyping techniques but also includes practices which focus more on analysis than on design, such as Contextual Inquiry [Holtzblatt and Jones, 1993] and ethnographic methods. Contextual Inquiry is founded on the premise that 'fundamentally, usability derives from the optimum match between users' work intentions, concepts and work flow, and the work expectations that designers build into the system. ... The match between what the system provides and how users want to do their work determines whether the system is experienced as disruptive or usable. Because much of the quality of this match has to do with how users experience their work and system interaction, we need to involve users in the design of an appropriate system work model and user interface' [Holtzblatt and Jones, 1993, pp.179-80].

However, Contextual Inquiry's attempt to achieve user involvement in the development process amounts to developers' interviewing users in their workplace about their tasks as they carry them out, essentially the concurrent protocol technique of task knowledge elicitation [Ericsson and Simon, 1993]. Contextual Inquiry recommends the subsequent use of 'affinity diagrams' to assist the developers to '*interpret customer data together, as a team*' [Holtzblatt and Beyer, 1993, p.95].

Thus, Contextual Inquiry simply uses conventional task analysis techniques, such as concurrent protocols, for gathering data about users' tasks. There is some interpretation and analysis in the dialogue between developer and user as the user describes her task: Holtzblatt and Jones [1993, p.199] claim that 'together with users, we cointerpret their experience of work and usability'. However, despite the explicit recognition of the need to involve users in the development process, the emphasis in Contextual Inquiry remains on developers' gathering data on users' tasks and subsequently, in isolation from the users, modelling the developers' interpretations of the data. As discussed below, this is typical of many approaches to task analysis.

Some research (e.g. [Heath and Luff, 1991; Luff, Jirotko, Heath and Greatbach, 1993]) has explored the utility of applying methods from the social sciences to the elicitation of user requirements. In particular, techniques from ethnomethodology and sociolinguistics have addressed user requirements by examining the social interactions in the users' organisation [Goguen and Linde, 1993].

These techniques however address the analysis of users' work settings at a sometimes painfully detailed level. Such an approach may be useful, indeed often necessary, in research but is rarely appropriate in pragmatic development work. They also lack one of the strengths of the software engineering approach: a clear recognition of the importance of transmitting requirements into design. The sociological techniques applied have tended to gather large amounts of detailed data about the users and their environment with little or no indication of how this data may inform the software development process. Anderson [1994] challenges the view of ethnography as a form of data collection in systems development and suggests wider analytical uses for it.

Furthermore, ethnographic approaches to analysing users' work situations and requirements are no more participatory, in the sense of involving users directly in the analysis work, than the hardest of software engineering approaches.

As PD has broadened its practices, it has also broadened its motivating concerns. As noted above, in its Scandinavian origins, PD was motivated by a concern for workplace democracy and trade union involvement in system development. However, in the later literature we begin to see the motivating concern moving from these considerations to a belief that active user involvement in software development leads to more useful and usable software products. For example,

Holtzblatt and Jones [1993, p.179] 'recognise that in order to design a usable system, we must involve users in design'. Grønbæk, Grudin, Bødker and Bannon [1993, p.79] state that 'cooperative development - full participation by both developers and users - requires rethinking the tools and techniques used in systems development. Motives for doing this range from simple cost-benefit arguments ... to concern for democracy in working life. Our point here is that user involvement is needed to achieve quality: better products, such as computer applications, will result when the developers have a knowledge about users' practice and future use situations that can only be obtained through cooperation with users'.

Gärtner and Wagner [1996] describe a shift in the emphasis of Scandinavian PD from political issues to techniques for user-developer cooperation in producing usable software. This changing concern within PD may have infiltrated along with techniques borrowed from the more North American approaches to user involvement in systems development. At the same time as concerns for workplace democracy were motivating cooperative development in Scandinavia, concerns with the failure of systems development projects to meet user requirements were motivating approaches to 'user centred design' in North America (e.g. [Norman and Draper, 1986]). Carroll [1996] notes that in the 1990s there has been considerable interaction between the PD and UCD traditions. However, while PD has adopted the notion that user-developer cooperation promotes software usability, non-Scandinavian practitioners of forms of UCD often have not adopted the Scandinavian emphasis on workplace democracy, viewing PD simply as another means to improving software quality; although the latter view is not ubiquitous [Muller, 1991a; Greenbaum, 1993a]

Despite the growing claims that PD aids usability, the PD literature lacks systematic attempts to relate participation to product. In a rare example of such an effort, Baroudi, Olson and Ives [1986] present a survey of user involvement in development projects and user satisfaction with the resulting systems. This study 'tentatively' concludes that 'user involvement in system development leads to increased user information satisfaction and increased system usage' [Baroudi, Olson and Ives, 1986, p.237]. More commonly, reviews of PD projects report the success of the project not in terms of satisfaction with the software produced but rather satisfaction with the cooperative development activities. For example, in reporting projects using the PICTIVE approach, Muller [1991b, 1992, 1993] consistently presents 'assessments' which focus on the users' satisfaction with the development process rather than with the product. Many reviews of PD projects, e.g. [Beck, 1993; Blomberg, Suchman and Trigg, 1996; Bloomer, Croft and Wright, 1997; Bødker, 1996; Greenbaum and Madsen, 1993; Grønbæk, Grudin, Bødker and Bannon, 1993; Hornby and Clegg, 1992; Madsen and Aiken, 1993; Noyes, Starr and Frankish, 1996; Thoresen, 1993; Wong and Tate, 1994], have no explicit assessment of the success of the PD work, simply listing the stages of the project, describing the techniques employed in the various stages, the artefacts

delivered by each stage and the gross level of participation which each group of stakeholders enjoyed. In so far as success is considered, it is often in terms of implications *for the PD approach used*, rather than for the users, either as users of the system produced or as participants in the development process. For example, Bødker [1996] addresses the 'concept of quality of the design process' and Muller [1992] presents an assessment of a PD method with no assessment of the quality of the software product.

This is not to suggest that PD researchers are unaware of the importance of product quality. Thoresen [1993], for example, stresses the need for iterative evaluation of software prototypes. However, there is a lack of published work relating the quality of systems produced to the nature of users' participation in the work which led to the products. In order to develop a systematic analysis of the relationship between participation and product, one requires a detailed, fine grain analysis of users' participation in the development work. However, this, too, is lacking in the PD literature. As Minneman [1991, p.171] notes, 'participatory design ... methods are primarily about intervention, not about what those interventions reveal about the situations in which the interventions are applied'.

The paucity of fine grain analyses of user-developer interaction in PD settings may be linked to a lack of analytic theory in PD. Greenbaum [1993b] outlines three perspectives on the need for PD approaches: political, pragmatic and theoretical. Pragmatically, Greenbaum argues (without offering evidence), PD reduces errors, builds better systems and increases product quality. Politically, PD offers workplace democracy and a better quality of working life. Finally, Greenbaum argues that 'there are many theoretical arguments supporting the need for PD'. She compares users and developers to Wittgenstein's lions and humans, arguing that 'a PD approach to prototyping' provides users and developers with a basis for shared experience and understanding. But even this short and undeveloped piece uses theory to argue the need for PD approaches rather than to examine the practice of PD. Similarly, in a stronger piece, Ehn [1989] develops a theory of PD but provides only a *post hoc* theoretical justification for the use of PD rather than a theoretical analysis of PD practices.

The relatively few attempts at theoretical analyses of PD practice tend to be at the level of work organisation rather than individual interaction and so lead to correspondingly coarse grained analyses. For example, Gärtner and Wagner [1996] analyse the political and organisational context of PD projects using actor-network theory. Hornby and Clegg [1992] apply theories from work organisation to a study of a PD project. They state the objectives of their study as to describe the way in which user participation was enacted in a particular organisational context and to examine whether they could explain the forms and outcomes of these processes of participation in organisational terms.

Bødker and Grønbæk [1991] do, however, produce quite a fine grain analysis in applying activity theory [Nardi, 1996a] to a study of cooperative prototyping work and it seems that activity theory may become a dominant theoretical approach to the study and practice of PD.

2.3 Giving the users what they want: effective HCI

The concern for usability has developed within a broader field of concern for issues of human-computer interaction. The growth of HCI as a field has reflected the growth of concern for what Friedman [1989] called 'user relations' issues. Influenced by fields such as psychology and ergonomics, much research in human-computer interaction has focused largely on issues of system usability. Indeed, Carroll and Campbell [1989] defined the role of HCI as a discipline in the following terms: 'HCI exists to provide an understanding of usability and of how to design usable computer artifacts'. Shackel [1990, p.248] lists 'five fundamental features of design for usability' as:

1. User centred design
2. Participatory design
3. Experimental design
4. Iterative design
5. User supportive design

When the majority of a product's users have received little or no training in its use, the success of the product depends heavily on its usability. However, while usability has grown as an area of research and of commercial interest, there is as yet little agreement on the definition of usability. As noted by Tetzlaff and Mack [1991], many researchers side-step this omission by treating the meaning of 'usable' as self-evident. It is, however, worth offering at least a working definition of usability.

Brooke, Bevan, Brigham, Harker and Youmans [1990] argue that usability cannot be defined except in operational terms and that to define usability one must consider

- the users who will use the product
- the tasks for which the product will be employed
- the conditions under which those users will perform the tasks.

The ISO standards committee on which Brooke worked used the following definition of measures of usability and usability attributes:

- usability measures: the effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in a particular environment
- usability attributes: the features and characteristics of a product which influence the effectiveness, efficiency and satisfaction with which particular users can achieve specified goals in a particular environment [Brooke et al, 1990].

These definitions emphasise the situated nature of any notion of usability. A system may be usable by a particular user for a particular task in a particular environment but unusable or poorly usable with another combination of user, task and environment. This is echoed by Shackel's definition of the usability of a system as 'the capability in human functional terms to be used easily and effectively by the specified range of users, given specified training and user support, to fulfil the specified range of tasks, within the specified range of environmental scenarios' [Shackel, 1991, p.24].

2.3.1 Evolution in HCI

Grudin [1990] presents a history of human-computer interface development which suggests that the focus of interest in HCI has moved ever outwards from human interaction with the hardware, to interaction between programmers and their software tools, to perceptual and basic cognitive aspects of the interaction between software and nonspecialist users, to higher cognitive, task and dialogue issues of HCI. Grudin [1990] predicted, quite accurately, that the focus of HCI research in the 1990s should expand further to include the social or work setting of interaction between possibly more than one user and more than one computer.

HCI as a coherent field dates from around the beginning of the 1980s. Early work concentrated on areas such as empirical studies of user interface components and cognitive modelling (see, e.g., [Card, Moran and Newell, 1983]). Early task analysis (TA) work in HCI reflected the field's more general focus on cognitive modelling and quite low level specification and evaluation of interfaces (e.g. Task Action Language [Reisner, 1981]).

Some work at this time (e.g. Gould and Lewis [1985]) presaged later HCI concerns with understanding and designing for users' higher level work tasks. As this concern has grown, so too has the application and development of TA within HCI. TA approaches span a wide range of research and development activities, with different granularities of task considered, from the detailed cognitive features of task performance to the daily workplace tasks which computer systems are designed to support.

Johnson [1992, p.165] notes that 'any TA is comprised of three major activities; first, the collection of data; second, the analysis of that data; and third, the

modelling of the task domain'. In the application of TA to system development, the activities are extended to include: collecting data on users' (current) tasks, analysing the data, modelling users' (current) tasks, deriving design requirements from the models and modelling users' envisioned tasks with a designed system.

2.3.2 Task Analysis and Participatory Design

Johnson and Johnson [1990, p.259] argue that 'using task analysis in conjunction with rapid prototyping is tackling the usability issues from a more informed and hopefully principled position of design generation followed by design development'. TA based approaches to system development have tended to support the involvement of users in the processes of gathering data on their current tasks. This compares favourably with much system development practice in which users often are not involved in these activities. However, while TA has often involved users in data gathering, the activities of analysing the data and modelling users' tasks have remained almost exclusively the preserve of the analyst (who may also be the software designer) and the derivation of design requirements and envisioned systems from task models remains something of a black art.

TA practitioners have tended to follow the conventional approach of gathering data from subjects, analysing and interpreting the data and modelling the analysts' interpretations. Even when, as in many cases, iterative validation of the model with users is recommended, task analysts use their own representations rather than involving users in the construction and definition of the representations. Several difficulties arise from this approach. Even given an accurate and comprehensive set of gathered data, it is possible that the analysts' interpretation of the data may be substantially different from an interpretation which users might have made. The model derived from the analysts' interpretation may not, therefore, accurately reflect the users' tasks. Indeed, it may prove difficult to validate the accuracy of the task model because, even if accurate, it may be based on an interpretation with which the users are unfamiliar. Validation of the task model may also prove difficult and time consuming because the modelling notations and media are not familiar to the users. Amending and validating an unsatisfactory task model may also take considerable time and effort because of the need again to move through the cycle of assimilating, assessing and interpreting gathered data, constructing a fresh model and returning to validation.

Thus, while PD approaches often seem to lack adequate attention to analysis of users' current work, TA approaches often lack consistent user participation across the range of system development activities. It seems, then, that a synthesis of TA's focus on analysis and PD's emphasis on user participation may prove useful. However, despite the seemingly inexorable expansion of the field of HCI to include such areas as PD and the popularity amongst the HCI research community of PD and TA as potential amelioratives to what Friedman [1989] has called user relations constraints, we have seen very few attempts at systematically integrating

practices and techniques from participatory and task analytical development with each other and within the overall system development process. Such an integration should foster the participation of users in the data gathering, analysis and modelling activities of TA and extend the participation of users upstream from envisioning and design activities to contribute to the overall development process.

A rare example of such an attempt is presented by Muller and colleagues (see, e.g., [Muller, Tudor, Wildman, White, Root, Dayton, Carr, Diekmann and Dykstra-Erickson, 1995]) in their description of PICTIVE and CARD. PICTIVE is a set of PD techniques which addresses the problems associated with rapid software prototypes and mockups and offers an alternative approach with the goals:

'to empower users to act as full participants in the design of systems that will have impact on their jobs and their work-lives

'to improve knowledge acquisition for design, and the quality of the resulting system, by involving the people with job expertise (the people who do the job) in the design process

'to improve the flow of the software engineering process by bringing representatives from major components of that process into the design phase as co-owners of the design ...

'to explore video technology as a means for recording and communicating design specifications' [Muller, 1991b].

PICTIVE provides a 'low-tech' prototyping technology: typical office stationery and 'design objects' made from card and plastic. These design objects may be generic (e.g. query fields, menu bars, dialogue boxes, etc) or may be prepared specifically for a single project. Examples of project-specific design objects are those produced by Muller, Smith, Goldberg and Shoher [1990] for a PICTIVE-based design of a project management groupware prototype: a set of coloured plastic squares with domain-specific icons drawn on them, a set of paper images of pop-up events and several sheets of desk-blotter paper which represented large, fixed areas of a presentation surface.

The PICTIVE design team includes both developers and users. Other stakeholders in the project may be included, such as marketing staff, technical writers or human factors specialists. While the design objects are initially made and provided by the developers, the objects are manipulated by all the members of the design team on a shared design surface to which all members of the design team have equal access. Tools such as pens, scissors and tape are available to allow the members of the design team directly to modify the design.

'The general notion of PICTIVE is that the design team manipulates concrete objects to explore a number of scenarios about how the target system is supposed to support the user's job or task' [Muller, 1993, p.215]. The use of familiar office stationery attempts to alleviate the technological intimidation of computer naïve users, while the PICTIVE process of involving users in manipulating the objects attempts to avoid the limitation of most mockup approaches that users simply react to and critique mockups produced by developers. PICTIVE allows more equal contribution to the software development process than does software based rapid prototyping or mockups. The low-tech approach facilitates the direct participation of users who may not be computer literate. Equality of access encourages the modification of the design in real time by the users without a software prototype or mockup implementor as intermediary.

The 'high-tech' element of PICTIVE is the video-recording of the sessions. The camera is focused on the shared design surface and records the voices of those working at the surface. The use of video in PICTIVE is intended to facilitate the communication of designs to system implementors, again without a loss of information through translation and to provide a history of design decisions. 'The video record of the design session will serve as a dynamic presentation of the design, as a conversational rationale for that design and for the decisions underlying the design, and as a means for communicating concretely the different views of different participants' [Muller, 1991b, p.226].

Video recording of the session provides a means of record keeping which tend to equalise participants' access to the record and which produce a very comprehensive, yet informal, record of the design and of the decisions which led to the design. Given the present state of the art, however, PICTIVE's reliance on video as a recording medium for both the design and the design rationale may be a weakness. A video recording contains a large amount of unstructured data which can be difficult and time consuming to analyse. Muller acknowledges this weakness and the difficulty of analysing video records when he reports that 'the most important problem was the poor transfer of the system concept from the design sessions to the developer This occurred because the developer was not involved in the participatory design sessions. ... The design sessions produced over a dozen hours of videotape, and the developer simply didn't have the time to comb through those records to collect information for implementation' [1992, p.456].

The relationship of PICTIVE to the upstream activities of users' task analysis and synthesis of design requirements remains unclear. At separate points in one paper, Muller [1993] claims that PICTIVE 'may be used for early requirements gathering and analysis' [p. 215], reports that 'with a high-level [requirements] specification, the [PICTIVE] session becomes an extended requirements analysis ... and little concrete design work is accomplished' [p. 223] and concedes that

'PICTIVE appears to be most useful in projects and products for which the requirements analysis has already been completed' [p.224].

PICTIVE's position in the camp of process oriented approaches [Floyd, 1987] may go some way to explaining the lack of clarity on the nature of representations which PICTIVE produces and how these representations might fit into an overall software development process. Process oriented approaches, challenging the traditional product oriented, specification driven approaches, have tended to deny the utility of, even the need for, any explicit specifications. For example, Kyng [1995, p.88] promotes a process oriented approach, explaining that, 'where traditional methods treat descriptions of ... work tasks as the primary input from the work context to the design process, our approach is based on the belief that the complex texture of workplace life should be handled primarily through action based techniques. Such tools and techniques must encourage users to bring their knowledge and skill to bear on design, without forcing on them mediation through explicit description'.

The suggestion that explicit descriptions of requirements and designs should, or indeed can, be dispensed with may be somewhat over optimistic. In criticising available techniques and representations for describing and specifying requirements and designs as not useful in supporting communication between users and developers, it is possibly more effective to find alternative representations which do support this communication than to abandon attempts to describe what it is one wishes to communicate.

Recognising the difficulties in applying PICTIVE to requirements analysis, Muller *et al* [1995] developed CARD (Collaborative Analysis of Requirements and Design), in order to address the problem that 'much of participatory work is concerned directly with the process of design. This had led ... to complaints that it was too easy to dive into the details. There was insufficient support for big-picture or work-level analysis, representation, and design'. [Muller *et al*, 1995, 142]. In the CARD method, components of users' tasks - based on a preliminary task analysis by the developers - are represented on cards. The cards are then sorted and ordered by the participants in the session to describe at a high level the task to be supported by the software system. The sorted cards represent an event flow of task components. Brief textual notes are taken, usually on the cards, of requirements which emerge from this manipulation.

Tudor, Muller, Dayton and Root [1993] argue that both PICTIVE and CARD provide three layers of design representation. Both techniques use low-tech paper-and-pencil artefacts to mediate design discussions. These artefacts also serve as a 'first level of design representation', capturing the appearance of the components of the design. These components must, however, be integrated into a complete design. Tudor *et al* [1993] suggest that the video recording of the session and hand-written annotations on the physical artefacts provide a 'second

level of design representation' to support this integration. Questions on the rationale for a design may be answered, according to Tudor *et al* [1993], by recovering the rationale from the video record of the sessions. This provides a 'third order of design representation'. (As noted above, however, and as conceded by Tudor and his colleagues, analysis of a video record can be difficult and time consuming.)

CARD has claimed some success [Muller *et al*, 1995] but has weaknesses from both the task analysis and participatory design perspectives. From the participatory design perspective, user-developer cooperation is compromised in so far as the task components manipulated by the user have been determined and presented in advance by the developer. The user does not participate in the initial choice and description of task components. From a task analytical perspective, simply asking the user to sort a set of prescribed task components does not represent an adequate exploration and analysis of the user's situated tasks. As with PICTIVE, the authors of CARD seem not to be clear on its role. They have used it 'to explore the requirements for a graphical layout system', as 'part of a task analysis of the job of directory assistance operators' and 'to design certain subsystems of a voice messaging product' [Muller *et al*, 1993].

Chin's work, e.g. [Chin, Rosson and Carroll, 1997], is another example of an attempt to integrate participatory and task analytical methods. This work builds on Carroll's well developed work on the 'task-artefact' cycle (see [Carroll, Kellogg and Rosson, 1991] and on claims analysis (see [Carroll and Rosson, 1992])). However, this work is still at a quite early stage and little has so far been published.

2.4 Where are we now?

Quite a lot of literature is produced by proponents of particular development approaches or methods or techniques to review the application of their favoured approach to systems development (see, for examples, section 2.3 above). In contrast, there are relatively few empirical studies which seek to provide an unvarnished description of what practitioners in general are using. Rosson, Maass and Kellogg [1988, p.1289] comment that 'many researchers have written about how the design of interactive systems *ought* to take place but virtually none have made a serious attempt to describe how it *does*'.

In developing new methods, techniques and tools, the tenet 'Know thy user' applies no less strongly when the intended users are software developers. When software developers are the intended users of a proposed new tool or technique, it is no less important than with other kinds of users to inform the design with an understanding of the tasks which are to be supported and of ways in which current support may be improved. Brooks [1990] notes that 'practitioner experiences are an extremely important source of information to the researcher about the often

complex interaction between the new technology they advocate and the organisational context in which it will be used'. Offering prescriptions for practice is of little use without first taking the trouble to develop descriptive models of the process one hopes to understand and to improve. However, as Minneman [1991, p.8] argues 'design researchers largely persist in ... building design tools without an understanding of the context of their use (or even studying their use after deployment)'.

There have, however, been some descriptive studies of the real world practice of systems developers. Rosson, Maass and Kellogg [1988] interviewed twenty-two software developers about twenty-two separate projects developing a wide range of applications. Each project resulted in a running system. Rosson *et al* reported that the strongest factor related to the development approach was whether or not the project was a research project. Six of the seven research projects studied adopted an incremental development approach with design and implementation occurring simultaneously and iteratively. None of the product development projects followed this approach, instead designing before implementing with iteration within each phase. This distinction in development approach was also related to size of development team, with smaller teams favouring the incremental approach. Rosson *et al* [p.1292] suggest that 'larger groups require more control and coordination to keep the design process intact'.

Three of the developers reported that they made no effort to get feedback from users. Of the rest, eight reported getting feedback from users at the design stage and eleven after the system had been completed. Of the eight, five gained only informal feedback from people in the work situation who were willing to look at a paper screen mockup or partial system demonstration.

Rosson *et al* asked the developers to report the methods they used for generation and subsequent development of design ideas. The most common means of idea generation was some form of user task analysis, reported by twenty-one developers. However, only seven of them interviewed or observed users in performing this analysis and only two development teams included a user. The remaining twelve developers relied on such techniques as 'taking a user's perspective' and 'generating task scenarios'. Only two developers reported using a form of user task analysis to flesh out design ideas. In both cases, this was done through the developers' 'taking a user's perspective'. Very few of the developers interviewed by Rosson *et al* reported systematic recording of design ideas or of progress made. Fifteen made at least some reference to informal records - notes, drawings, diagrams or lists. 'To the extent that formal documentation was used, it was often associated with later phases of the project (e.g. documentation of the system, a design rationale created *after* the design had been finalised)' [Rosson, Maass and Kellogg, 1988; emphasis added].

Johnson and Johnson [1989, 1990] conducted an interview based survey of three software developers' views on how, why and where task analysis might contribute to software design and on the requirements for tools to support task analysis. All the developers interviewed by Johnson and Johnson [1989] believed that knowledge of users' tasks would contribute benefits to the design process. Two of the developers suggested that task analysis could be used to provide initial input to interface prototyping. All of the developers suggested that the results of user task analysis should be presented in a range of formats to support validation by users, mapping and checking between user tasks and proposed designs and validation of the design specification and implementation.

However, the developers reported in practice an absence of user involvement in development processes with the developers generally not knowing how users perform their tasks or even in some cases what those tasks are. 'Neither were TA or HCI methods and principles used' [Johnson and Johnson, 1989, p.1460]. The developers reported that they did not follow a structured development method and even with individual development teams there was no unified view of development practice. Individual elements of structured development methods, such as SSADM [Downs, Clare and Coe, 1988], were used in isolation for specific pieces of development work.

Terrins-Rudge and Jørgensen [1993] studied ten system development projects, through participant-observation, structured interviews, questionnaires and video analysis. They found that the most common approach to system development was one of 'muddling through'. Formal or structured methods were not employed, developers preferring selectively and opportunistically to use individual parts of such methods in the course of muddling through. This approach was predicted to continue for so long as it is easy to use, time saving and politically feasible

Bellotti [1988, 1990] interviewed eight developers, four in academic organisations and four in commercial organisations, with a view to determining the extent to which HCI task analysis techniques were employed in systems development practice. All of the interviewed developers claimed never to have used such a technique and some claimed never to have heard of them. Bellotti [1988] reported several practical problems identified by the developers. These included poor communication, exacerbated by a lack of shared terminology between developers and users; developers' uncertainty about users' requirements; exclusion of users from the development process, leading to misinformation about and misunderstanding of users' requirements; developers' unfamiliarity with the task domain; and overly casual evaluation.

Walz, Elam and Curtis [1993] analysed video records of nineteen meetings during the course of a system development project. They were interested in the participation of development team members in knowledge acquisition, sharing and integration. 'No specific design techniques or disciplined development

methodology was forced upon the project team' [p.63]. The development team involved eight developers, a project manager and a representative customer. Two other customers attended the third and fourth meetings and presented conflicting requirements. The developers responded by attempting to address the requirements proposed by only one of the customers, effectively ignoring the others. The developers were then unhappy when the original customer representative tried later to involve other customers in discussing the requirements.

Developers made only brief notes from their interactions with customers. Most of the requirements information from customers was given orally and a large amount of it was lost. Walz *et al* also argue that developers were capable of integrating only 'a *design bite's* worth of information into their current understanding of the design task, based on the ability of the new information to "attach" to that already integrated into the design' [p.71]. Again, this led to a large amount of information from customers being lost or unnoticed. Such information often had to be reconstructed (and not always successfully) at later stages of the development project.

Myers and Rosson [1992] report the results of a survey of user interface programming to which they received 74 responses from developers in software development companies (44%), software research laboratories (29%) and universities (27%). Part of the survey gathered information on the development process followed by the respondents. Seven developers reported no interaction with users while 89% claimed to have made 'some effort aimed at gathering and responding to user input'. 43% reported 'some level of formal testing with end-users prior to release'. 46% of projects involved the development of one or more prototypes and these were 'often offered to users for comments or testing'. 11% of developers claimed to have used participatory design, with users contributing directly to interface design. Other developers interviewed users or observed them at work, although no figures are given for this. The nature of the participatory design activities, interviews or observations was not reported.

These general empirical studies of software development provide a clearer view of the actual takeup of approaches from PD and TA than does literature devoted to reporting a particular application of one of these approaches. In the latter literature, the particular approach is often applied at the instigation of the author of the report and generally in a research driven rather than commercially driven project. The empirical studies reviewed here suggest that, despite heavy promotion by the research communities of methods from PD and TA, these methods are not widely used by system development practitioners. Techniques from structured development methods are occasionally used but generally opportunistically and not as part of an integrated approach to development. This research first set out to check these assessments against what real world system developers claim to be using in current practice.

2.4.1 Listening to developers

The initial phase of research began with a series of semi-structured interviews with software developers from both academia and the private industrial sector. Nineteen developers were interviewed, two from a university and seventeen from six industrial organisations. The interviewees were software developers with many years experience of software development projects. The commercial interviewees described projects ranging from a hand-held point-of-sale device for insurance sellers to a computer aided design application for the aerospace industry. One academic interviewee described the development of a course directory and room allocation application. The other academic interviewee described the development of an interactive application for deriving mathematical proofs. Each interview focused on a single software development project, although at times the response to a particular question on a project included a reference to a similar or relevant issue in another project. The interviews were based around the following questions.

1. Why was there a project at all? (i.e. was the project developer-driven or user-driven?)
2. How was it decided (and by whom) what the product was to be? (i.e. application requirements)
3. How, and by whom, was the user group defined?
4. Who, if anyone, was consulted for user requirements?
5. How were user requirements
 - (a) determined;
 - (b) recorded;
 - (c) used in the development process?
6. How (if at all) were the user requirements validated or evaluated?
7. How (if at all) were the design and implementation evaluated against user requirements?
8. Was the product right?
 - (a) how did it compare with the user requirements?
 - (b) how happy were the users with the product?
 - (c) how happy were the developers with the product?
9. Would you have done it the same way again?

Unstructured interviews complemented the more formal, semi-structured interviews. The author also had frequent contact, both semi-structured interviews and conversations, with a senior developer at the software development company which sponsored this research. In addition, the author attended as an observer a product definition workshop for a potential new software project and an ongoing development meeting for a new release of an existing software product.

The product definition workshop for a potential new application development project was called by the project leader, workshop facilitator and chief designer to explore a possible opportunity to develop a product to complement the company's current document analysis software product. Those present included the project leader, the author (as observer), two members of the proposed development team, the Development Manager for the current product and a developer with working knowledge of a third-party software development environment which had been suggested as a possible implementation platform.

At the software development meeting, the author observed three software developers discuss and develop the design of an extension to one of the company's existing mass-market interactive software applications. In addition to interviewing and observing developers at work, a questionnaire (see Appendix 6) was developed from the questions asked in the semistructured interviews and from issues raised in the interviews and workshops.

Sixty copies of the questionnaire were sent by standard post (snail-mail). However, the departmental database of software developers on which this mailshot was based seems to have been less than wholly accurate: ten of the postal questionnaires were returned with the addressee not known at the address. No responses were received to the postal questionnaire. It seems likely that more than the returned ten failed to reach their target. (One email response confirmed that at least one other potential respondent was no longer at the address used.) Other discouraging factors may have been the length of the questionnaire and a reluctance amongst 'computer people' to use snail-mail.

The questionnaire was also sent by email to thirty individuals from the departmental database, to a 'CHI' mailing list and to the general mailing list of the company sponsoring this research. The electronic survey produced a greater response. Forty-nine replies were received, twenty-eight of which were complete, i.e. provided answers to every question.

The first part of the questionnaire sought background information on the respondent's experience in software development. Part two asked about the nature of the group in which the respondent worked. Part three investigated the use of prescribed software development methods by the respondent's group. Finally, part four asked for information on a single software development project on which the respondent had worked (rather than from the respondent's more general experiences of software development).

A questionnaire does not provide direct observation. It can elicit only the respondent's account of the topic under consideration. By restricting the respondents' answers in part four to the work associated with one project, it was hoped to avoid the worst of their providing a generalised, and perhaps idealised, account of what they do.

2.4.2 We are here

The meetings provided qualitative, observational data while the interviews and questionnaires provided more focused and more quantitative data. From another perspective, the interviews and questionnaire provided data on what developers claimed to do while the meetings provided data on what they are actually observed to do.

The workshop facilitator proposed to base the product definition workshop around an iteration through the first stage of USTM, a method proposed by Hutt and colleagues [Hutt, Donnelly, Macaulay, Fowler and Twigger, 1987; Hutt and Flower, 1990]. The workshop had been intended to run over two days but was cut to one when the Sales Manager objected to the 'loss' of his staff for more than one day. The course of the workshop was adversely affected by the need to move in one day through a process which had been planned to take two days. Most of those present were not familiar with the USTM method and not enough time was available clearly to explain what was sought from each stage in the process. Partly as a result, the method contributed little to the early part of the workshop and was almost entirely abandoned during the second half of the workshop.

Aspects of the political environment within the company and the economic environment in which the company operates laid the ground for difficulties experienced in the workshop. Management were reluctant to invite customers to attend the workshop, seemingly because of an assumption that customers' attendance is useful only when development has progressed to the stage at which a concrete design has been produced.

Users were not welcome at a very early product development meeting for two main reasons: users were viewed as having little useful to say about a potential product without first having seen an implementation of some kind; and it was strongly feared that customers' exposure to the directionless, chaotic start to a project could damage the developers' image as 'experts' and cause customer defection to a competitor with an intact image of omniscience.

There was further management reluctance to allowing three of the sales force to spend two days in a workshop. It is an indication of the significance of understanding users to a software development project that, in the absence of real users, those in frequent contact with customers were so important to the workshop that it was reduced to one day, with the concomitant difficulties, rather than lose their potential contribution. The absence of users and the reluctance to release sales people indicate how much real influence this significance often has.

The process ran into difficulties when the workshop attempted to identify user roles and tasks within potential customer organisations. In the absence of people from these organisations, the workshop organiser had invited company sales people with experience of the application area, a history of working in a major

potential customer organisation and frequent ongoing contact with customers. It was hoped that these sales people could provide the workshop with the desired elucidation of user roles and tasks.

In the event, it proved difficult to gain a clear understanding of user roles and tasks from the sales people at the workshop. They had great difficulty in communicating their knowledge of users' roles, tasks and requirements to the workshop.

One of the most intriguing aspects of the communication difficulties in the requirements synthesis meeting was that the sales people understood and could speak the developers' language, understood and could speak the users' language, but could not translate between the two. The sales people can (and frequently do) speak to customers and to developers in their respective languages about their respective worlds. But when asked to speak to developers, in the language of developers, about the world of users, they flailed between using the vocabulary and conceptual framework of the users' world (which the developers did not grasp) and using the vocabulary and conceptual framework of the developers' world (which were unfamiliar and awkward in thinking about the users' world).

The nature of this communication difficulty suggests that the synthesis of user requirements demands a shared vocabulary between users and developers. The user cannot be expected to provide this and so it must be introduced and used with guidance from the developer to produce a directed dialogue between developer and user. The difficulties of the sales people in communicating an understanding of users' tasks and roles to the developers illustrate that this communication is an ongoing, emergent process which cannot reasonably be assumed to fit into a pre-allocated slot in any software development method. The short time available in a one-day workshop served only to emphasise this.

In the software design meeting, there was a great deal of debate on the structure and contents of menus and on the underlying means of specifying the types of external database with which the product should synchronise. There was a lot of speculation about how users might perform tasks with the proposed product. This speculation was, however, couched in terms of: 'We can do *A*. How will the user react to *A*? What do we put as *B* to deal with this reaction?' *B* was mainly couched in terms of constraining what the user *can* do as a reaction to *A*. Almost all of what was to be presented to the user (in the form of menus, dialogue boxes, lists, etc) was determined by design decisions on how technically to achieve the desired functionality.

Validating the speculation by asking users what they did and what they wanted was ruled out. Management of a non-development area of the company sought to maintain an image of the company as experts, as knowing everything, and also sought to maintain exclusive contact with customers. There was a deep reluctance to allow developers to talk to users or to let customers see incomplete versions of

products. The company's relationship with users (or customers, as they were always called) was very much sales driven rather than cooperative.

Some requirements for the design under discussion were clearly defined, since this design was for an extension to an existing product. However, the discussion was aimed at finding a technical solution to a technical problem, with the user interface depending almost entirely on what solution was determined. This approach also implied that the technical *problem* had been clearly defined. It was not at all clear that it had.

One of the designers, responsible for a particular aspect of the design, had forgotten a previous design decision which affected the design. It was agreed that failure to track design issues and decisions was a problem. A user interface design was developed and agreed but then abruptly changed towards the end of the meeting when it was decided that the designer responsible for much of it had not got sufficient time to produce an implementation before a deadline.

The interviews with developers provided insights into a range of different types and sizes of projects. It is interesting then that there were several common threads in development practices. The two developers in the academic environment produced relatively small systems. These were not, however, research projects and had identified users for whom the systems were being developed. In each of these projects, the developer produced a first version of the system without ever talking to the intended users. The design of the initial version was based on the intuitions and experience of the developers. In one project, the first version was presented to and discussed with users. Lessons learned from these discussions were incorporated into redesigns and the system evolved rapidly and iteratively. In the other project, The first version was presented to users as a delivered system to be used. It was largely rejected by users and a radically different design was subsequently produced, again without user involvement.

A very different project used a very similar approach. In a project to produce a CAD system for a large company, prototypes were again produced by developers on the basis of very sparse initial analysis. The developers then showed the prototypes to users and elicited their comments. The developers then redesigned in isolation. The interviewee commented that there was little point in talking to users because they very rarely say clearly what they want.

Throughout these developers' experiences, the little user involvement which occurred was in the assessment and redesign of initial versions of the system. There was no user involvement in upstream analysis activities or in design of the first prototypes. As described earlier in this chapter, most of the PD literature describes and promotes practices which increase user involvement at the redesign stage, and often introduces user involvement in initial prototype design, but does not extend user involvement upstream to the predesign activities.

Convincing developers of the value of user participation in upstream analysis activities may be an uphill struggle. As one interviewee argued, 'talking to customers early in a project is too abstract. They need to see a product'. Indeed, the usefulness of having a concrete prototype with which to work is apparent if the desire is to involve users in design and redesign of envisioned systems. But the point remains to involve users also in the analysis and understanding of their current tasks - and this is an activity most efficiently conducted in the early stages of a development project. None of the developers interviewed claimed to have used any systematic form of task analysis, although several claimed to recognise the need to understand users' tasks as a prerequisite for developing systems.

Task analysis in all the projects reported was at best *ad hoc*. For example, in a project to develop a document digitising and storage system, the developer started knowing nothing about the users' work. The company secretary explained the background, the users' work and the requirements. The developer then wandered around the workplace, 'looking for scale and scope', talked to users at random, asking them what they did. Then, in isolation, the developer produced an informal description of the users' work. This was text based, using diagrams 'when it became complex'. The developer then looked at what technology was available and useful, discussed some design ideas with the company secretary and then designed a system in isolation. The developer claimed to have built the interface with users sitting beside him but conceded that in fact they were 'two senior henchmen and a guy who was going to be in charge of half a dozen users'.

In another project for a very large company, the developer interviewed had talked to users initially, analysing the current system, 'because you can't upset current practice too much'. But the developers then produced requirements specifications in isolation and never returned to validate them with users. In another large company, with many in-house development projects, the only contact between development teams and users was through their respective managers. In fact, project managers themselves had contact with users' managers only through the project managers' boss. The developer interviewed here argued that 'the customer shouldn't always be given what he wants; it should be what he needs', with the implication that only the developers knew what the users needed.

The interviews suggested that few developers use conventional SE structured methods. One very experienced developer claimed that commercial mass market software was never produced using a structured method. This lack of use is often because structured methods are seen as time consuming, too constraining, not adaptable, imposing a large burden of administrative and documentary overheads and having few benefits to offer in addressing and meeting user requirements. The implementation of structured methods was restricted to the larger organisations and within these was applied half-heartedly. Developers often 'threw away' steps of the structured method. Several developers complained that SSADM [Downs, Clare and Coe, 1988] was very difficult to use because of lack

of CASE tool support. SSADM was also disliked because it increased pressure from managers who wanted to see 'lines of code, not stacks of paper'.

Structured methods were not considered for smaller projects, where development practices tended to be *ad hoc* and experimental. In larger projects where structured methods were demanded by management, the elements of structured methods which were used tended to be imposed *around* the day to day detailed development work which remained *ad hoc* and experimental. The overheads of structured methods conflicted with developers' eagerness, and frequent management pressure, to produce running prototypes as quickly as possible and documentation and administration was often done after rather than during development projects.

The interviewees described only one project which had explicitly set out to promote participatory development. This was the development of portable system for sales staff as an in-house project within a very large company. There had been two earlier projects which attempted to produce the same product. Each had in turn been abandoned as a failure. The failures had been viewed as the result of lack of user involvement leading to an inability to determine and design for user requirements. Hence, the third project began with the intention of pursuing a participatory approach to the development.

Unfortunately, the developers were forbidden by senior management from talking to real users. The only reason made explicit for this was that contact with users should build up their hopes and later disappoint them. Consequently, the development team took two ex-sales staff as representative users. These people had had no experience as users for two years, during which period details of the users' work had changed. In addition, it was rather unkindly suggested that they had moved from sales because they were not very good at it in any case.

The two representative users worked with the developers two days per week for several months. As described by the interviewees, they became jaded and fed up with looking at screen after screen. They were disgruntled at losing commissions because of the time taken away from their own jobs. Over time, they became part of the development team, identified with the product and became very reluctant to have it criticised. They were seen to have lost any real relation to users, being much more familiar with the product than new users could ever have been. The project was completed but the product never used because it was found to take twice as long as the old manual system.

Respondents to the questionnaire tended to be experienced software developers. Fourteen of the twenty-eight described themselves as 'senior', 'chief' or 'project leader'. Practical experience ranged from 18 months to 30 years with a mean of eight years. The size of their current development teams ranged from 2 to 20 members with a mean of 7.75.

Thirteen of the respondents received their training in software development from a first degree which was the sole training for five. Ten of the respondents received their training from a postgraduate degree which was the sole training for four. Thirteen received employment based training which was the sole training for six. Four of the respondents were self taught of whom two had received no formal training.

The application domain for respondents' current development teams was wide, ranging from device drivers through software development tools to educational games software. Eight of the respondents were working in academic research groups, the rest in commercial software development.

In response to the question '*does your group have a defined software development policy?*', six of the respondents in academic research groups replied *no*. One replied that '*User Centred System Design and Cognitive Engineering guide our development*'. The other academic respondent adhered to an in-house development method comprising prototyping with close user involvement following object-oriented analysis. Amongst the commercial developers, one did not understand the question, thirteen replied *no*. Of the remaining six, one claimed adherence to the ISO 9000 standard, one to User Centred System Design, two to the BS5750/ISO 9001 standard, one claimed to '*gather resources, information, design, produce a prototype, test, alfa, test, beta, test, final version*', and one claimed to be working towards conformance to the ISO standard.

Thus, 67% of commercial developers who replied to this question claimed not to follow a defined software development policy, while 22% claimed current or future conformance to a version of the ISO 9000 standard. 75% of the academic research groups did not follow a defined software development policy. 7.7% of those who replied, academic and commercial, claimed adherence to User Centred System Design while another 7.7% used prototyping.

The next three questions investigated the history of use by the respondent's development group of various prescribed development methods and techniques. The respondent was first asked '*does your group use one or more of the following?*': SSADM, JSD, CAP, RAD, JAD, VDM, DFDs, Z, SSM, Multiview, Yourdon-deMarco, Prototyping, Participatory Design. Respondents indicated for each of the methods whether her group used it *never, occasionally, often, always* or the respondent had *never heard of it*. The next question asked '*where you have used one or more of [the] above, would you in general characterise its use by your group as?*: *strict adherence, loose but complete adherence, using appropriate parts as desired, not used correctly*. The final question in this part asked '*for each of [the methods or techniques] which you or your group has used, please indicate how useful you found it to be*'. Here the choices were *indispensable, very useful, useful, unhelpful, hindering*.

There were several incomplete responses to these questions. Two of the replies from commercial respondents to these three linked questions were dropped from analysis as unreliable. A further one of the commercial respondents made no reply to any of these questions. Another commercial respondent claimed never to have heard of any of the listed methods. Two of the academic respondents made no reply to any of these questions. One of these two did, however, add the comment, 'no proprietary systems used, we apply the technology which is suitable to the problem, ER modelling, predicate calculus specifications, etc. Heard of most of [them]. Mostly they restrict and constrain the problem solving process'.

One academic and one commercial respondent replied that they *always* used *prototyping* and *participatory design* and left blank the boxes for all the other methods. The commercial respondent claimed to use *appropriate parts as desired* of both prototyping and participatory design and to find prototyping *indispensable* and participatory design *very useful*. The academic respondent claimed *loose but complete adherence* to both prototyping and participatory design and to find both *very useful*. This left five academic and fifteen commercial respondents who gave complete answers to these three linked questions. These twenty responses were broken down as follows.

SSADM One commercial respondent used it occasionally; eleven respondents had never used it; a further eight had never heard of it.

JSD One academic respondent used it occasionally; eleven respondents had never used it; a further eight had never heard of it.

CAP seven respondents had never used it; a further thirteen had never heard of it. (Not an altogether surprising result since this was the European Union's Common Agricultural Policy.)

RAD one commercial respondent used it often; seven respondents had never used it; a further twelve had never heard of it.

JAD eight respondents had never used it; a further twelve had never heard of it.

VDM one commercial respondent used it often; eight respondents had never used it; a further eleven had never heard of it.

DFDs one commercial and one academic respondent used it occasionally; one commercial and one academic respondent used it often; six respondents had never used it; a further ten had never heard of it.

Z one commercial respondent used it often; ten respondents had never used it; a further nine had never heard of it.

SSM seven respondents had never used it; a further thirteen had never heard of it.

Multiview one academic respondent used it occasionally; five respondents had never used it; a further fourteen had never heard of it.

Yourdon-deMarco one academic respondent used it occasionally; one academic respondent used it often; nine respondents had never used it; a further nine had never heard of it.

Prototyping one commercial respondent had never heard of prototyping; one academic and six commercial respondents used it occasionally; two academic and four commercial respondents used it often; two academic and four commercial respondents always used it.

Participatory design one academic and two commercial respondents had never used participatory design; a further two commercial respondents had never heard of it; one academic and three commercial respondents used it occasionally; one academic and seven commercial respondents used it often; one commercial and two academic respondents always used it.

Part four of the questionnaire asked the respondent to describe a single project on which she had worked. Question nineteen of this part asked, for the project in question, 'was any structured development method (e.g., SSADM, JSD) used?'. Twenty respondents (71%) replied *no* to this question. seventeen respondents replied *no* to both this question and question 10, 'does your group have a defined software development policy?'. One commercial respondent who replied *no* to question nineteen had failed to understand question ten, another claimed in question ten that his group in general followed a structured method conforming to the ISO 9000 standard and a third claimed in question ten to follow structured procedures with the aim of conformance to the ISO standard.

Two commercial and two academic respondents replied *yes* to question nineteen. Academic A claimed in question ten that '*User Centred System Design and Cognitive Engineering* guide our development'. The other academic respondent adhered to an in-house development method comprising prototyping with close user involvement following object-oriented analysis. Their responses to question nineteen referred to these methods.

In response to question twenty, which asked how the method referred to in question nineteen had been employed, Academic A claimed 'very rigorous adherence to *User Centred System Design* in an informal way'. Academic B qualified his *yes* to question nineteen with the comment that they had applied no method directly but had loosely but consistently employed object-oriented analysis, partly employed object-oriented design and intensively used prototyping. In response to question twenty-one, which asked how useful the method had

proved, Academic A claimed that 'as an *ad hoc* approach, it was absolutely essential'. Academic B claimed that his approach had been partly useful but should 'never be used to handcuff developers'.

Both commercial respondents who replied *yes* to question nineteen had claimed in question ten to follow the BS5750/ISO 9001 standard. In response to question twenty, Commercial A claimed that their project had used JSD, with poor adherence, and informal inspections. Commercial B claimed to have used his group's in-house method conformant to ISO 9001 'fairly loosely - although most steps were covered in some way'. In response to question twenty-one, Commercial A claimed that JSD had proved itself 'not at all useful' while the informal inspections had proved 'very useful'. Commercial B noted that their method had proved to be 'in this case, not particularly useful'.

Questions twenty-two to thirty-three focused on the synthesis, recording and use of user requirements within the reported projects. Two commercial respondents replied to none of these questions (nor, indeed, to any of the remaining questions), leaving twenty-six responses.

One commercial and one academic respondent claimed in response to question twenty-two, that the user group for their product was determined by preliminary analysis, including interviewing potential users. One commercial respondent did not understand this question. The remaining twenty-three projects fell into one of two camps. In five academic and ten commercial projects, the users were assumed to be predefined at the outset of the project, involving no analysis. In two academic and six commercial projects, the user group was decided on the intuition of the development group members without input from any source outside the development group. Comments typical of the first camp included, 'it's not a decision which needed to be made' and 'there was a definite target population decided by the situation we were designing to meet'. Typical of the second camp were the comments, '[users were] decided in general discussion in accordance with our own plans' and '[the user group was] negotiated with marketing on an ongoing basis'.

Questions twenty-three to twenty-six investigated how the respondents discovered and recorded user requirements and used these requirements in the design process. For four academic and four commercial respondents, the main source of user requirements was the intended user group. For four commercial respondents, users' managers were consulted rather than the users themselves. For two academic and seven commercial respondents, user requirements were based on the developers' own knowledge and experience, with no one consulted outside the development group.

One academic and one commercial respondent relied on literature surveys to determine user requirements. Two commercial respondents received 'user requirements' mainly from the development organisation's own marketing group.

One academic and one commercial project drafted typical users into the development team to provide user requirements.

Questions twenty-seven to thirty-one investigated the respondents' use of user requirements specifications. Question twenty-seven asked, for the respondent's project, '*was an explicit user requirements specification produced?*'. Of the eight academic respondents, five replied *no* and three *yes*. Of the eighteen commercial respondents, eight replied *no* and ten *yes*.

Of the three academic respondents who claimed to have produced an explicit user requirements specification, one replied to question twenty-eight that '*the user interface prototype was used as the specification*'. This requirements specification was evaluated by '*testing the user interface prototype with users*'. Consequent changes to user requirements were incorporated into the prototype.

Another academic respondent's specification consisted of '*an explicit list of goals that the software should address*'. This specification was evaluated by '*informal user testing*' of prototypes. The specification was renegotiated amongst the developers as user requirements changed.

The third academic respondent's user requirements specification consisted of '*a paper [giving] a rough overview of use requirements*'. This respondent claimed that '*there was no explicit user requirements specification validation*' and noted that a user requirements specification was '*found to be rather useless, since we have the users at hand. Fast prototyping and testing changed everything [quickly]*' and the paper requirements specification was not explicitly updated to reflect changes.

The ten commercial respondents who claimed to have produced an explicit user requirements specification described a variety of forms of this specification. Five of these were straightforward textual descriptions of the requirements. The first of these five was evaluated by prototype testing and 'expert opinion' within the development group. The text of the specification was amended to incorporate changing requirements.

A second textual requirements specification was evaluated through 'user trials with simulations and prototypes ... based on implemented design guidelines from the user requirements specification'. Changes were not incorporated into the textual specification. A third textual requirements specification was evaluated through discussion with senior management of the customer organisation. The specification was edited following changes in the development organisation's marketing strategy. A fourth textual requirements specification was evaluated by asking users to read and to comment on the text. The text was then amended accordingly. The fifth textual requirements specification was written by the development team leader and 'distributed for comments within the [development]

company'. No potential users outside the company were consulted. No changes were made to the requirements specification.

Two of the commercial respondents described more formal textual user requirements specifications. One of these specifications consisted of a several hundred page document providing a list of individually specified requirements with a rationale and a priority for each requirement. The other more formal textual specification consisted of 'a text document specifying all elements of the look and feel with accompanying sample screen designs, flow charts and story boards'.

The user requirements specification for another commercial respondent consisted of an extremely simple list of requirements 'jotted on a piece of paper and checked off as completed'. This requirements specification was evaluated through extensive user testing with versions of the software. Consequent amendments to the requirements specification were simply jotted down.

Another commercial respondent used a prototype, with some accompanying documentation, as requirements specification. This specification was evaluated by running the prototype with focus groups and individual subjects. The prototype and documentation were amended to take account of the results of these tests.

For the remaining commercial respondent, the requirements specification was taken to be the contract between the developer and customer. This contract was a result of competitive tenders and was evaluated by 'members of the development team verifying, before the contract was completed, whether it was possible and how much effort it would take'. Users were not consulted at any time about user requirements. Changes to the requirements specification were not made because 'any changes would mean that the entire contract (including pay) had to be reworked and usually meant that other tenders would have to be considered at the same time'.

Question thirty-two asked, '*how (if at all) was the design evaluated against user requirements?*'. A total of six academic and sixteen commercial respondents answered this question. Two academic and four commercial respondents replied that the design was not evaluated against user requirements. Three of each group of respondents conducted this evaluation by informal trials with prototypes. One academic and three commercial respondents conducted more formal prototyping tests, gathering feedback from users through interviewing and questionnaires. Six commercial respondents conducted this evaluation by review meetings of the development group. One commercial respondent used formal tests of the software as specified in the requirements for the project.

In response to question thirty-three, two commercial respondents replied that the product failed to meet user requirements. One academic respondent claimed that

the final product did not meet user requirements so well as prototypes because project funding did not allow full development of the prototypes. Two academic and two commercial respondents claimed that the product not only met all user requirements but gave users more usable functionality than the requirements specification demanded. One academic and seven commercial respondents said that the product compared satisfactorily with user requirements, while two commercial respondents claimed that they compared very well. Two academic and four commercial respondents noted that user requirements had changed very much during the project and that the product reflected the new requirements.

Two academic and two commercial products had not yet been delivered, while one commercial project had its funding cut off before a product could be delivered.

Questions thirty-four to thirty-seven explored the satisfaction of developers, users and customers with the product. One academic respondent claimed that her development group was unhappy because some of the best aspects of their prototypes were cut from the final product for lack of project finance. Otherwise, all the respondents claimed that their respective development teams' satisfaction with the product ranged from quite happy to ecstatic.

In response to the question, *'how happy were the users with the product?'*, one academic respondent gave no reply and another admitted that she did not know. The remaining respondent replied that their users' happiness ranged from quite to very. No respondent admitted to having unhappy users. This may be viewed in the context of replies to the next question, *'how was user satisfaction with the product ascertained/measured?'*. Two academic and four commercial respondents conceded that user satisfaction was not investigated at all.

Three academic and two commercial respondents informally asked users how happy they were with the product. Two academic and three commercial respondents investigated user satisfaction with questionnaires and interviews. Four commercial respondents relied on the volume of user complaints to gauge user satisfaction. Two commercial respondents conducted formal usability tests with users throughout the project while one commercial respondent relied on the customers' signing off formal acceptance tests agreed in the project specification. The remaining five of the twenty-eight respondents did not answer this question. Thus, 75% of developers claimed no systematic assessment of user satisfaction with the product.

The results reported above suggest that very few developers use established software engineering structured methods. This is often because such methods are seen as too constraining, not adaptable, imposing a large administrative burden and having few benefits to offer in addressing and meeting user requirements. There are heavy overheads, particularly in the demands for extensive documentation and often obscure notations, and there is a lack of perceived

benefits in developing modern interactive systems. The few developers who espouse structured methods do not often find them useful and often find them difficult to apply rigorously, largely because the methods tend to constrain the development process rather than to support it.

The survey suggests that, in practice, developers follow unstructured *ad hoc* approaches to systems development, relying on intuition and experience and a certain arrogance that they know what users want. Developers rarely seek users' views, let alone their involvement in the development. They often do not construct requirements specifications in the ways, at the times or with the content that structured methods prescribe, again because of a strong perception of a negative cost-benefit balance.

There have been so many attempts at prescribing formal or structured methods of engineering software that most developers have heard of only a small proportion of them. They have generally never used those of which they have heard. In contrast, a large majority (95%) of developers have not only heard of *rapid prototyping*, but claim to use it, 35% occasionally, 30% often and 30% always. Similarly, only 10% of developers have never heard of participatory design, while a further 15% have never used it. Most developers (75%) claim to have used it, 20% occasionally, 40% often and 15% always.

On the other hand, only 30% of developers claimed to have consulted the intended users as the main source of user requirements for the single project on which each developer reported. Many of the respondents to the questionnaire (15%) took user requirements only from the users' managers and not from the users themselves. Developers in interviews and conversations confirmed that the management of customer organisations frequently forbids the developers from speaking directly to users. 35% of developers relied on their own knowledge and ideas to determine user requirements, consulting no one outside the development team.

In some cases, this approach was explained by developers as appropriate when the intended users of the product are themselves software developers. The prevailing attitude amongst developers is that they are similar enough to the users to determine requirements without consultation. This approach is entirely inappropriate when the product is not a software development tool. In any case, even when the intended users are software developers, the developers of the product cannot assess its functionality and usability objectively. They are too close to the product, know it too well and cannot approach it as novice users. Nor are they necessarily a good sample.

Neither were the surveyed developers rigorous in consulting users about how the delivered product met the users' requirements. 22% of developers investigated user satisfaction with the product by interviews and questionnaires. A further 8% conducted systematic usability tests with users. However, 70% of developers carried out no systematic assessment of user satisfaction with the product. Yet

only 4% of developers admitted that they did not know how happy users were with the product. No developer conceded that the users were unhappy with the product. This may be an indication of complacency amongst developers that they get the product right. It is, however, likely that those who contributed to the survey were unwilling for personal or professional reasons to admit to having unhappy users.

However, many developers confidential comments suggested that software products generally have not attained high standards of usability. Projects in which structured methods have been employed generally had not resulted in highly usable systems, but neither had developers systematically applied other approaches which resulted in high usability. The surveyed developers produced systems which ranged from adequately to barely usable. Two caveats are worth noting: (i) these are subjective evaluations by the developers themselves of the usability of their products; little or no objective usability evaluation was performed on the systems they developed; (ii) if anything, the developers were reluctant harshly to criticise their products (mainly for reasons of personal and organisational politics) and usually became really scathing only off the record.

2.5 Research questions raised

The chapter began with a look at the evolution of computer based systems and the practice of their development. For modern interactive systems, 'user relations' have come to the fore as the primary constraint on systems development. In response, three broad strategies have been proposed better to understand and to meet user requirements for functionality and usability. The strategy which has received most attention, at least in the research community, is facilitating direct cooperation between users and developers in the development process.

This work accepts the premise that modern interactive systems must be built with a clear focus throughout the development process on supporting users' work tasks [Gould and Lewis, 1985]. Hence, the system development process must be based on a thorough analysis and, where necessary, redesign of those tasks. Users are the primary source of extensive and intimate knowledge of themselves, their tasks and their working environment. Hence, it is a further assumption of this work that there should be as few layers of mediation as possible between users, developers and the emerging artefacts of the development process. The more layers of mediation which exist, the more opportunities there are for ambiguity and error in communicating and translating between representations and between people.

Established systems development methods tend to impose many layers of mediation between users and developers and between users and development artefacts. Such layers of mediation consist of both people and representations. For example, when users are consulted at all, what they have to say may be recorded and interpreted by an analyst who may pass a representation of her

findings to a designer. The designer may reinterpret this representation of the user's thoughts and present a representation of this reinterpretation to an implementor. The implementor may then attempt to represent her interpretation of the user's thoughts in an implementation of the software. The software may then be presented to the user who, it is hoped, finds it to be just what she wanted.

Given that a user is a primary source of task knowledge and requirements, one of the jobs of the professional developer is to draw out those requirements, regardless of how capable the user initially may be of expressing her requirements in terms which the developer can understand. The developer must act as a catalyst for the synthesis of requirements from the user's situated task knowledge. For the developer, requirements analysis depends upon eliciting, validating and representing information about the situation for which the system is being developed. In order to derive an understanding of the requirements for the system which arise from the situation, the developer must first derive an understanding of the situation itself. While simply giving developers access to users, as a passive on-line source of information, may well go some way towards facilitating this process, substantive improvement may require more direct and active user cooperation with developers.

TA and PD each are frequently touted in the software development literature, particularly the HCI literature, as potential solutions to 'user relations' issues. TA, however, lacks a strong reported history of application in substantial real world development projects. While applications of PD techniques are reported frequently, this often is in the context of research projects and studies specific to the use of these techniques.

Many practitioners are at least vaguely aware of the popularity of both PD and TA within the research community and make positive noises about such approaches. However, it seems from empirical studies that practising developers have heard of and pay lip service to the techniques of PD and TA, but rarely use them. When they are used, it is spasmodically and opportunistically rather than systematically. If practitioners are to use such a technique, they must know in detail what it involves and must be convinced of its value.

However, the PD literature lacks systematic analyses of what it is for users and developers to cooperate, of the nature of user-developer interaction in 'participatory' settings, of the *effectiveness* of user participation in such settings and of the impact of user-developer cooperation on the resulting systems which are produced.

The PD literature often reports and promotes PD (i.e. design) activities and provides only vague assessments of the software produced. But they do not systematically relate features of PD activities to features of software usability and they do not establish that PD work has had a positive effect on software usability.

Both PD and TA approaches often lack consistent user participation across the range of system development activities. Most of the reported techniques of participatory design do indeed focus on design which is, however, only one of many processes which comprise system development. While PD approaches often pay inadequate attention to analysis activities, TA approaches generally do not actively involve users beyond initial data gathering and, at times, checking artefacts produced by the developers. Neither PD nor TA approaches generally pay adequate attention to the integration of the diverse aspects of system development.

Hence, the development and application of techniques which embody both TA's focus on analysis and PD's emphasis on user participation may prove useful. However, we so far have seen very few attempts at systematically integrating the practices of participatory and task analytical development with each other and within the overall system development process.

This work set out to address the issues summarised here. The author proposed an approach to direct user participation in task analysis and design activities and applied this approach in two substantial real world software development projects. Hence, this work contributed to the development of PD's focus on analysis and integration into the overall system development process, contributed to the still meagre stock of real world applications of systematic task based development and contributed to the theoretical and practical integration of TA and PD.

However, it was not the aim of this work simply to add yet another method to the largely undisturbed arsenal of system development methods. Rather, the application of a participatory task based approach provided an opportunity to ask fundamental questions about the nature and impact of user 'participation' in systems development work.

This research included an analysis of the nature of user-developer interaction in participatory development settings and assessed the impact of user contributions to the development process on development artefacts and on the usability of the resulting software. Hence, this work goes some way to explaining what it is to 'do' participatory development, describing one form of such development and assessing its results. The work reported here addressed four related research questions (see Figure 2.2) which captured the fundamental concerns of the research.

The first, and perhaps most fundamental, question asked: what *is* user participation? In posing this question, the thesis considered the processes and activities in which users and developers were collaboratively involved, examining how the participants interacted in performing those processes and activities.

The second research question asked whether the development projects studied actually were participatory. That is, did the users' presence during development

work result in active contributions by users to the development process; or were the users just used by developers as a resource to be tapped; or was the users' presence simply irrelevant to the developers' performance of the development activities?

The third research question asked: in so far as there was active user participation in the development activities, how *effective* was this participation? In considering this question, it was necessary to define effectiveness in the context of participatory development. Chapter five develops and applies such a definition in terms of the assimilation of users' contributions into the artefacts, both external and internal, of the development process.

The fourth research question asked how valuable user contributions were to the development process. Again, it was necessary to define *valuable* in this context. Chapter six uses a definition of the value of a contribution in terms of its influence on the features of the implemented software to trace relations between specific user contributions to the development process and specific usability issues with the resulting software in each development project.

The remainder of this thesis is devoted to tackling these research questions.

Figure 2.2: Research Questions

RQ1 What is user participation in software development, i.e. what are the processes and activities in which users and developers are involved and how do the participants, especially users, contribute to those processes and activities?

RQ2 Were the projects studied participatory, i.e. did users actively contribute to the interaction, rather than being passively present or simply a source of on-line data to be tapped by developers?

RQ3 Was user participation effective in the studied projects, i.e. were user contributions assimilated into the external and internal artefacts of the development process?

RQ4 How valuable was user participation in the studied projects, i.e. what relations were there between user contributions to development activities and features of the software product's usability?

Chapter 3

The projects and their analysis

The previous chapter argued that 'user relations' remain the main constraint in software systems development practice and continue to become more and more significant as types of applications and users continue to expand. Chapter two also identified several key issues in the proposition of task analysis (TA) and participatory design (PD) methods to tackle user relations constraints in modern systems development practice. These issues may be characterised as follows.

- TA and PD are widely promoted within the academic community as effective means of ameliorating user relations constraints. However, there is a dearth of reported applications of TA in substantial real world development projects. Reported applications of PD are often in the context of research projects. Professional systems development practitioners are positive about TA and PD but rarely choose to apply them in practice.
- Approaches to development in the PD camp tend to lack a strong focus on analysis of the users' current (and envisioned) tasks, while TA approaches generally lack active user participation beyond initial data gathering, if at all.
- Despite the popularity of both TA and PD within the academic community, there has been very little research effort on systematically integrating task analysis and user participation with each other and with the overall system development process.
- PD literature is not strong on theory. The PD literature lacks detailed analyses of what it is for users and developers to cooperate, of the nature of user-developer interaction in PD settings. Furthermore, there has been little systematic analysis of the effectiveness of user participation in such settings. The PD literature is also weak on analyses of the impact of specific features of user-developer cooperation on the resulting implemented systems.

This research addressed these issues through the development and application of an integrated approach to task based participatory development in two real world development projects. This chapter describes the development approach which was adopted, the projects which were studied and the approach taken to the analyses.

The chapter begins in section 3.1 with a description of the task based participatory development approach which was applied in the projects studied.

Section 3.2 provides an overview of the two major system development projects studied and of a short pilot study. Section 3.3 summarises the types of data which were available for analysis from the projects studied and section 3.4 describes the research method adopted in analysing the projects.

3.1 The development method

It was not a goal of this research to propose or to promote a prescribed systems development method. As noted in chapter two, there is already a large number of such methods, often promoted only by their devisors and used in practice by nobody. However, in order to address the issues of interest here, it was necessary to adopt in the projects a systematic approach to development work which was task based and participatory. Given the weaknesses of existing TA and PD methods, as outlined at the beginning of this chapter, it was in turn necessary to devise a systematic approach to task based participatory development which could be applied across the projects studied.

The approach used, labelled CUSTARD (Cooperation with USers in Task Analysis for Requirements and Design) [O'Neill, 1995; O'Neill, Johnson and Coulouris, 1995], supported user participation in development activities from initial discussions, through modelling the users' current work situation, modelling envisioned work situations and software design. The approach is of interest in itself, but within this research only in so far as it provided the opportunity to address the issues and research questions presented above. It is not the intention of this work to promote the approach adopted here as the way to do participatory development. It is likely that another approach could have worked at least as well for the participants and it is possible that this approach may not have worked in the hands of other participants.

The description of CUSTARD in this section applies to the two full development projects and the pilot study. Differences in the application of the approach between each of them are described in the overviews of each project in section 3.2 below. Drawing on lessons from the real world application of development methods (see chapter two), CUSTARD provided recommendations of which activities to perform, in which order and with which tools, but allowed the participants to refine any and all of these as they required. A clear example of this is in the use of task modelling notations. These were based on previous work on TKS [Johnson and Johnson, 1991] and EuroHelp [Breuker, 1990]. However, as an integral part of the cooperative development work, the notations were refined and agreed by the participants as the task modelling activities progressed. As described in more detail later in the thesis, this helped to promote the users' understanding and feelings of ownership of the task models. Similarly, the 'steps' of CUSTARD itself were applied, ignored, refined and reapplied on a contingent basis throughout the two

projects. Hence, the cooperative development method is itself cooperatively developed.

CUSTARD took task based development as a sound foundation for building useful, usable systems because of its strong focus on users' tasks [Gould and Lewis, 1985]. The approach extended active user involvement in the system development process from, at one end, data gathering for task analysis and, at the other end, evaluation of software designs, to include participation in all the phases of data gathering, analysis and modelling of user tasks, derivation of requirements from the resulting task models, envisioned task modelling and software prototype design and evaluation. The approach was intended to maximise user participation without user dependence and to minimise layers of mediation between users, developers and the artefacts of the development work.

The first cooperative task analysis phase began with initial data gathering. This was conducted through semistructured discussions amongst the participants, observation of the users at work and collection of documentation from the work place. Other forms of data gathering may also be used, such as concurrent and retrospective protocols; for a summary of data gathering techniques, see [Johnson and Johnson, 1989].

In the first phase of analysis, the participants provisionally identified

- the work situation of concern (i.e. the domain into which a putative system was to be introduced);
- actors who currently fulfilled roles in the work situation; this provided an indication of the people in the work situation whose cooperation in subsequent development work should be particularly valuable;
- roles within the situation of concern which should potentially be supported by a new system;
- a preliminary description of the primary tasks associated with each role to be supported; (this served as a starting point for the cooperative construction of a model of the tasks performed in the role;)
- a preliminary statement of the desired system (without reference to envisioned technology or possible designs) to support the attainment of the identified tasks.

At this stage, the developers were interested in gaining a preliminary understanding of the work situation and identifying the main roles and tasks therein. The developers needed to come up to speed on knowledge of the domain effectively to communicate with users. The users also needed to develop some understanding of the design process, its tools and artefacts and the technical possibilities in order effectively to communicate with developers.

This next phase of development work saw the participants engaged conjointly in the development of an external shared model of the users' current work situation. Building on the initial analysis phase, this work constructed a task model which covered the overall work situation. The roles identified by users in the work situation served as the unit of analysis.

Where there was a small number of highly interactive roles in the work situation, one or more of the actors who fulfilled each role became involved together in the cooperative construction of a model of the tasks which were performed across the work situation. Each user participant had special insight into the role which she performed and often also provided new and fresh insights into other roles in the work situation. Where there was a larger number of roles than could be effectively analysed in a single group session or where there was little interaction between roles, representatives of each role became involved in the cooperative development of a model of the tasks which were performed in their own role. For example, one session was devoted to cooperative modelling of a technical consultant's tasks, while another session was devoted to cooperative modelling of a receptionist's tasks. In these cases, further sessions were then required to integrate the individual task models into one comprehensive model and to model the interactions between the roles.

Some authors (e.g. [Bannon and Bødker, 1991; Benyon, 1992]) have criticised task analysis and task models on the grounds of what they do not achieve. In many cases, such criticisms stem from a failure to appreciate the purposes of a task model and the nature of a task analysis. In turn this may be due at least in part to the changing scope and resolution of task analysis. As described in chapter two, task analysis has in recent years moved its level of focus from cognitive activities to users' work tasks and working environment.

In what has become known as task analysis, an understanding is required not only of the tasks but also of the domain in which those tasks are performed. The position adopted in the work reported here was that domain information can and should be represented in the task model. The task models produced in the development projects studied here included not only decompositions of the tasks, but representations of the roles in which tasks were performed and of workflows and relationships between roles and tasks. Thus, the task models included information on the domain in which the modelled tasks were performed. Inevitably, there will have been aspects of the domain which were not represented in the task models. Indeed, some such unmodelled aspects of the domain may have been relevant to the performance of the modelled tasks. However, it is not the purpose of task modelling to provide a perfect representation of every aspect of the users' tasks and work environment, but to present a *sufficiently* accurate and complete representation to support the progress of the development work. The participants are the arbiters of sufficiency in this regard at any given period in the development work. Iteration is inevitable.

Advantages to this approach included having a single, comprehensive model of the work situation, the users, their tasks, objects and workflows, current or envisioned, with layers of mediation between representations minimised. The only additional representations produced to complement the task models were models of the work objects, their decompositions and relations.

The media used for constructing the task models varied across the projects. The differences in the application of the approach across the pilot study and the two major projects are described in sections 3.2.1 to 3.2.3. In each case, active user participation was encouraged throughout the development of the task models.

The task models represented the work situation in terms of roles, tasks associated with each role and flows of work objects amongst the roles. Tasks were represented in a primarily hierarchical structure of subtasks with notions of sequence, selection and iteration. The task models in each project were represented through a graphical notation. The model of the users' work situation began in each project with a rough schematic representation of the users' physical working environment. The sketch included stick figure representations of people who worked in the offices, their relative working locations (represented by their relative placement in the sketch) and flows (represented by arrowed lines) of work objects, documents and so on, amongst the people.

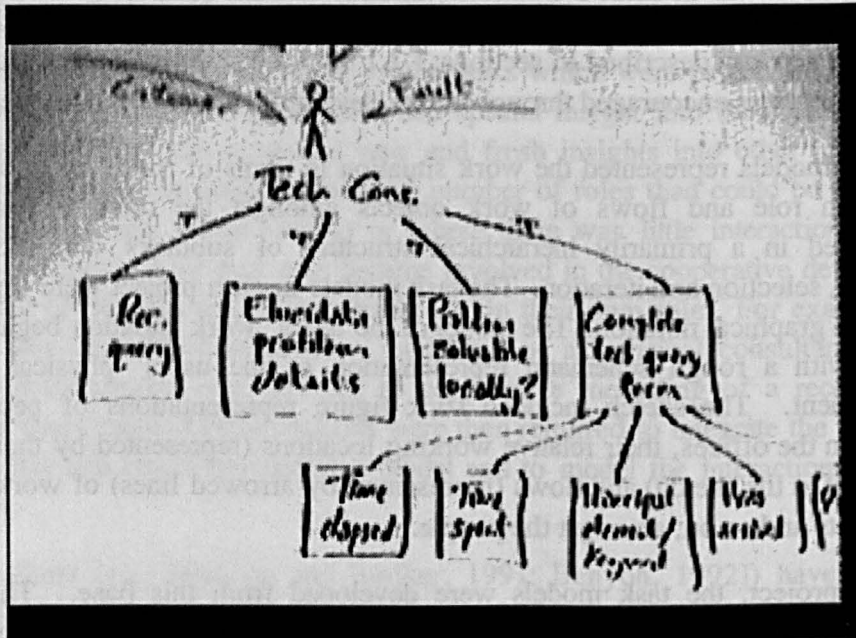
In each project, the task models were developed from this base. The people identified in the work situation were formalised as roles. The notational device for a role was slightly different in each project due to the medium used for the representation. In the CI project a 'head-and-shoulders' icon denoted a role because this icon represented a person in the software used to produce the task model charts. In the CS project it was a full body stick figure because that was the symbol which participants found easy to draw. In each case, the graphical role notation was labelled with a textual title, such as 'Tech. Cons' (Technical Consultant) in Figure 3.1 below.

Tasks associated with a role were denoted by boxes connected to the role by lines. Similarly, subtask boxes were connected by lines to the task into which they were composed. Task decompositions were represented below their associated role with leaf nodes at the bottom, furthest from the role annotation. Task and subtask boxes contained a brief textual description; for example, 'log fault'. The connecting lines were also labelled.

Some boxes were used to represent conditional tests which the user applied to determine which actions to carry out next. To represent this, a box contained the conditional test (e.g. 'Item in database?'), lines from the box were labelled with the range of possible responses (e.g. 'Branch YES', 'Branch NO' and 'Branch DON'T KNOW') and the box to which each line ran contained the task with was carried out following each response. Each of these boxes in turn might have subtasks. Another specialised notation covered cases where a task (or subtask) took different

forms depending upon the object on which the task was performed. In these cases, the line between the main task or subtask box and the variants was labelled with 'Variant' and the name of the object on which the specific task variant was performed.

Figure 3.1: Part of a task model on a whiteboard from the CS project



The model of users' current tasks developed in the first cooperative task modelling phase served as a starting point for the second phase. The participants cooperated in amending and refining the model of current user tasks to synthesise a model of envisioned user tasks. The users contributed to this work both knowledge of their current tasks and their wishes for enhancements to those tasks. The developers contributed their knowledge and experience of computer based support for tasks.

In the development projects, a direct route across the 'design chasm' from the task model of the users' current work situation (TM1) to a task model of an envisioned work situation (TM2) was developed. This route involved (i) mapping work flows in TM1, (ii) identifying key tasks in the current work situation which demanded support in an envisioned system, (iii) mapping envisioned work flows through the key tasks, (iv) extending this mapping into an envisioned task model (TM2).

The primary criteria for identifying key tasks which should be carried forward from the current situation to the new design were (a) tasks necessary to goal achievement; i.e. tasks without which the system's job could not be done; and (b) the user's (mental) model of his tasks. In general, workflow or 'housekeeping' tasks were not considered as key tasks for this purpose. These were, for example, simple physical tasks of moving a work object from one location to another. Workflow

and housekeeping tasks for the envisioned system were identified while extending the work flow-key task mapping into the envisioned task model.

Hence, an envisioned task model, again covering the overall work situation and including roles, tasks and workflows, was derived in further cooperative task modelling sessions. Thus, the second cooperative analysis phase delivered a model of envisioned user tasks.

In turn, in pursuit of the objectives of minimising layers of mediation and bridging the design chasm, envisioned software components were mapped directly from envisioned task model components. Two facets of the task model - its elements and the structure of those elements - mapped, respectively, to two elements of the software - again, elements and structure. The elements mapped from the object classification and decomposition within the model of the envisioned work situation, while the structure mapped from the envisioned task decomposition structure.

The design phase of CUSTARD also promoted active user participation, with the cooperative development and construction of paper prototypes, similar to the approach taken in PICTIVE [Muller, 1993]. CUSTARD benefited from the strengths of the PICTIVE method and had the additional advantage of a clear basis in user tasks. The ongoing involvement of users in the development process avoided the limitation of most mockup or prototyping approaches that users simply react to and critique developers' mockups [Nielsen, 1990].

However, the paper based design prototypes lacked some of the strengths of software prototypes. Specifically, a software prototype may, through user testing, provide a model of interaction behaviour which a static paper based prototype cannot provide. Unlike a software prototype, the design representation in paper prototypes cannot simulate the real time behaviour of the implemented system in interaction with the users. Also, while participatory prototyping removes much of the need for translation and interpretation between user and designer (and, if applicable, implementor) [Bødker and Grønbæk, 1990], the translation between paper based representation and implemented software offers opportunities for ambiguity and error.

These weaknesses were addressed by replication of the paper prototypes as running software prototypes, allowing users to interact with the designs which they had helped to create. User based evaluation of the software prototypes (see chapter six) was facilitated by the users familiarity with the layout of the interface. The user's familiarity with and sense of ownership of the design helped to alleviate the technological intimidation often felt by users faced by software prototypes. As an embodiment of the design from the cooperative paper prototyping session, the software prototypes also avoided the problem with most software prototypes that they represent solely the developer's interpretation of the user's needs.

As noted by Carroll, Kellogg and Rosson [1991], the introduction of an artefact designed to support user tasks may alter the nature of those tasks. This in turn

alters the requirements for a task supporting artefact. Thus, in the approach described here, evaluation of user interaction with a software prototype resulted in iteration through the processes of cooperatively modelling envisioned enhanced user tasks, specification of requirements, cooperative design of paper based prototypes and construction and evaluation of software prototypes.

3.2 The development projects studied

A short pilot study and two substantial system development projects were conducted during the course of this research. The general approach described above to task based participatory development was applied in each. The pilot study involved a first pass at cooperative analysis and modelling of users' current tasks. This study had no design remit or intention and was aimed only at piloting the cooperative analysis method. The two systems development projects each involved user participation through all the development activities from analysis of current tasks to evaluation of implemented software. The following subsections provide overviews of the pilot study and system development projects. Section 3.2.1 provides an overview of the pilot study and lessons learned from it. The mechanics of the cooperative task analysis process were changed considerably for the two main development projects as a result of lessons learned from the pilot study. Sections 3.2.2 and 3.2.3 respectively provide overviews of the work which was conducted in the two development projects. Much of the rest of the thesis is devoted to presenting lessons from these projects.

The first system development project was the development of a commercial software system in the field of criminal intelligence. This project is referred to below as the CI project. The second was the development of a query logging system for the Computing Services department of Queen Mary and Westfield College and is referred to as the CS project. Studying these two projects allowed comparison of the application of CUSTARD in two very different application domains and with two very different teams. The CI project was aimed at a mass-market commercial package while the CS project was aimed at an in-house custom system. Keil and Carmel [1995] contrast user-developer links in participatory activities in these two types of development projects. Grudin [1991a] identifies three types of development project: product development, in-house development and contract development. The CI and CS projects represented, respectively, the first two of these types.

An advantage of the CI project was that it provided a test of development work in a highly commercially motivated real world development situation. However, the corresponding commercial sensitivity led to issues of access and publication. Minneman [1991] reports similar difficulties and argues that his study of two different projects provided complementary publishable data. Similarly, here the CS project provided material which was less sensitive and more easily publishable than material from the CI project. The CS project was also a real project but lacked the

strong commercial sensitivities of the CI project - in fact the department actively demonstrated the system to others trying to develop similar systems. In the CS project, multiple, real users were involved throughout the project. Video recording of development activities was more extensive in the CS project (see section 3.3).

3.2.1 Pilot project overview

The pilot study set out to realise an initial approach to the analysis and modelling of a user's work situation which supported the active participation of the user in the analysis and modelling activities. Work in the pilot study did not go beyond this initial modelling. In the pilot study, the author, *qua* developer, worked with a solicitor from a large law firm. It was assumed in the first instance that a system was required to support the lawyers' work. Initial data gathering in the form of semistructured discussions allowed the developer to gain a preliminary understanding of the work situation and the lawyer to gain a preliminary feeling of what might be required and what feasible.

For the purposes of the pilot study, the only actor identified in the task analysis was the lawyer herself. At the user's insistence, two distinct roles were noted which she fulfilled in her work. Role one was a pensions solicitor whose goal was to provide cost effective advice and support to clients who were pension scheme trustees or employers. Role two was an employee of the law firm whose goal was to make money for the firm.

Primary tasks associated with these roles were identified as follows:

Role 1 Pensions solicitor

Task 1.1 To research legal questions

Task 1.2 To draft legal documents

Task 1.3 To advise clients

Task 1.4 To support clients' transactions

Role 2 Employee of law firm

Task 2.1 To record time worked *per* client

Task 2.2 To promote good relations with clients.

In addition, the initial analysis identified several other roles in the work situation whose tasks should need to be supported by any envisioned system and whose actors interacted with the user participating in the analysis. Further analysis of these roles was beyond the scope of the pilot study. The roles are simply listed here:

Role 3 Client of law firm

Role 4 Client's actuary

Role 5 Other side's solicitor

Role 6 Pensions partner/senior solicitor

Role 7 Corporate solicitor/partner

Role 8 Personal secretary

Role 9 Internal post messenger

Role 10 Departmental 'know-how' officer

Role 11 Trainee solicitor.

A preliminary statement of the desired system was produced as follows: a system to support a pensions solicitor in researching legal questions, drafting legal documents, advising clients and supporting clients' transactions, in order to provide cost effective advice and support to clients; and to support a CityLaw¹ employee in recording her work-time in order to make money for CityLaw.

User and developer then moved on in the pilot study to the cooperative development of an external shared model of the users' current work situation. The task model which was built covered only the tasks of the single user. The work of other users was not modelled and the task model covered in detail only one area of the user's work.

A model of the user's current tasks was constructed through user-developer cooperation using card-based task components. 'Analysis components' were used to support the cooperative analysis and modelling of user tasks. Generic analysis components took the form of blank cards representing generic tasks and subtasks (rectangular cards), roles (schematic person shaped cards), actions² (square cards) and objects (triangular cards) (see Figure 3.2). Specific analysis components were generic analysis components which were labelled by the session participants to fit the particular work situation being modelled. In addition, participants could introduce other analysis components for the purposes of a particular project on a contingent basis.

The proposed advantages of the use of analysis component cards to support cooperative task analysis were

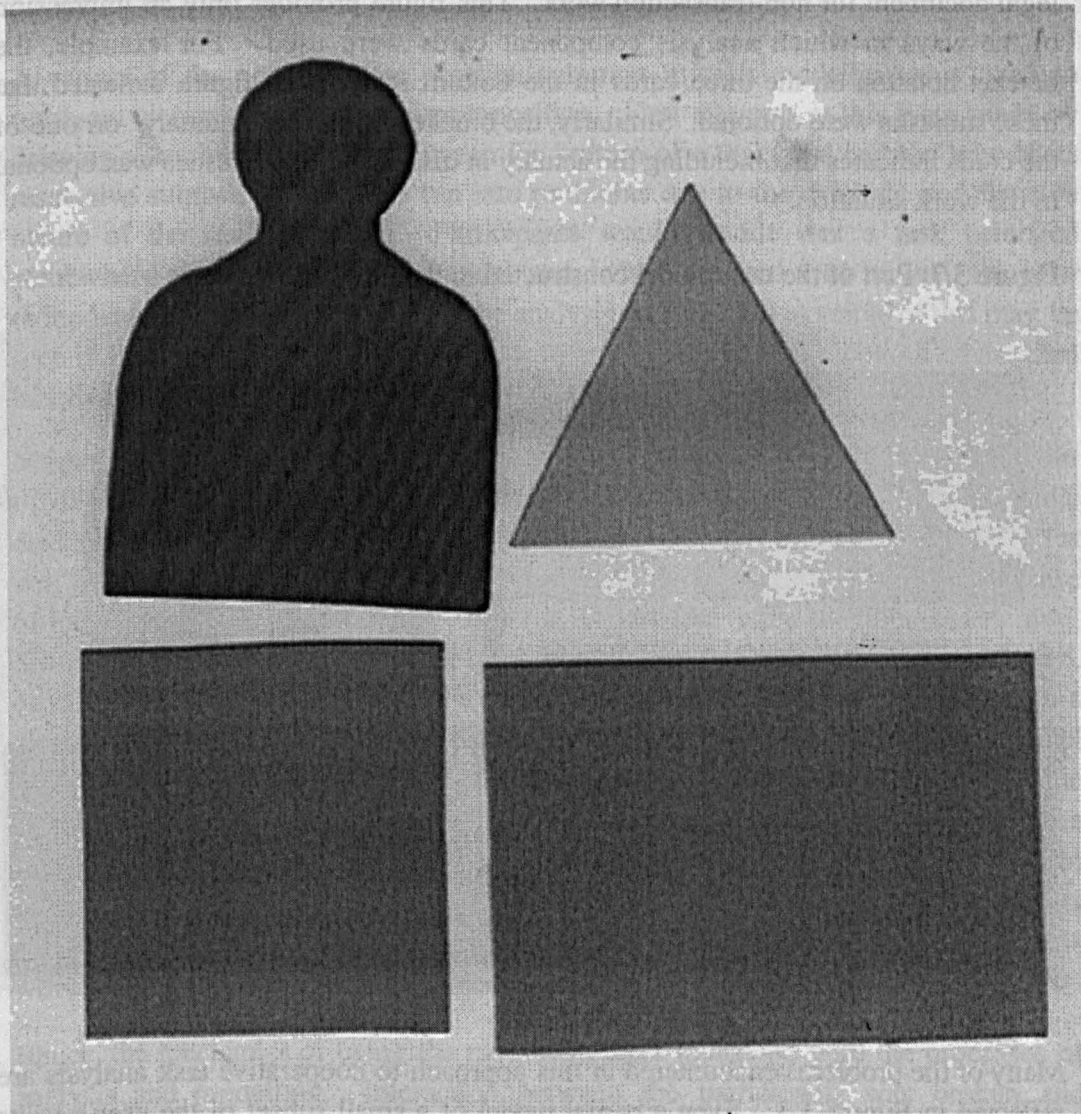
- direct manipulation of the cards by users should encourage a sense of direct collaborative participation in the analysis work;
- the use of familiar stationery should prove unthreatening to users from most workplace backgrounds;

¹ A pseudonym.

² Leaf nodes in a task decomposition.

- mobility and ease of manipulation of the cards should facilitate early attempts at construction of the model when multiple, tentative, 'quick and dirty' attempts are often required;
- the initially unstructured and informal nature of the modelling tools should facilitate the cooperative development and application of a task modelling notation which all participants understand and own;
- the modelling surface and materials should provide a shared focus for user-developer collaboration.

Figure 3.2: Analysis component cards in the pilot project

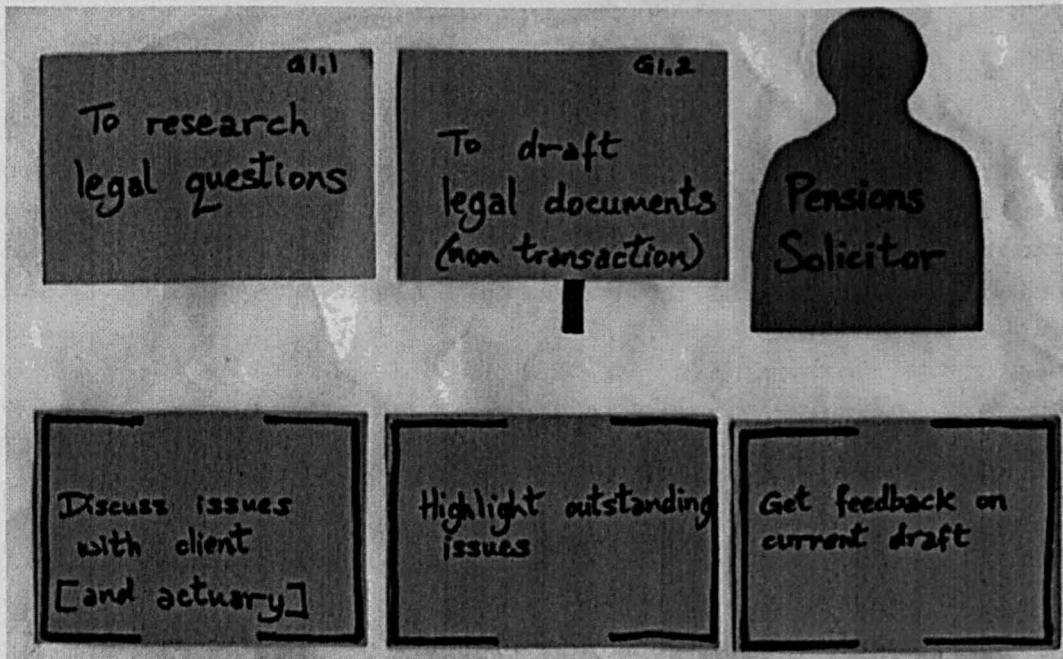


The analysis sessions proceeded through manipulation of analysis components on an 'analysis surface' to which both participants had access. The analysis surface took the form of a large (A1 size) sheet of white card around which the participants

sat. The participants cooperated in manipulating, amending and introducing analysis components to construct a model of the user's tasks. On being introduced to the model, generic analysis components were turned into specific analysis components by labelling them with markers.

Construction of a model of the user's current work began with the introduction of a generic role card. This card was labelled with the role (Pensions Solicitor) whose tasks were to be modelled. Cards representing the primary goals associated with that role were produced and placed alongside the role card. Analysis proceeded by selecting each primary goal in turn and modelling a decomposition of the tasks involved in achieving that goal. Figure 3.3 shows a small part of the task model in which part of a task structure has been modelled for the primary goal of drafting a legal document for non-transaction work. This figure provides only an impression of the ways in which analysis component cards were used. For example, the bracket notation on the three cards in the bottom row of the figure indicated that these subtasks were optional. Similarly, the bracket around 'and actuary' on one of the cards indicates that including the actuary in discussion of the issues was optional in the work situation.

Figure 3.3: Part of the task model constructed in the pilot study



Many of the problems encountered in this approach to cooperative task analysis are reflected in Figure 3.3. Even a partial model of a small subset of the user's tasks quickly covered a large area. The A1 size analysis surface proved wholly inadequate to contain a decomposition of even one primary task. Even at that size, the analysis surface itself was large, unwieldy and not readily portable or storable.

The problem of limited space in which to develop the model exacerbated other difficulties in using the analysis components. There were major difficulties in denoting relations between task model components. Placing task cards in a horizontal row was used to indicate that these tasks were considered to be at a similar level of the decomposition. A simple physical problem with this was that the necessary rows very often exceeded the width of the analysis surface by a considerable distance. A more abstract, representational issue was the need to represent relations such as sequence, selection and iteration amongst task components. As noted above, a bracket notation was devised to represent an optional task, but this did not explicitly represent the conditions under which the task should and should not be performed. In most cases, the conditions were discussed by the participants but remained represented only in their internal models of the work situation.

A variant of the node and arc notation common in task modelling was used to represent hierarchy in the task decomposition. First attempts at this were made by drawing a line with a marker between the bottom of a task card and the tops of its respective subtask cards. This ran into problems due to the dynamic and iterative nature of the analysis work. Participants would decide that a task belonged somewhere else in the evolving model and, when the card was moved, the now redundant marked line remained on the analysis surface. An example of this may be seen in Figure 3.3 where a line protrudes to no purpose from the bottom of the card labelled G1.2.

Markers were abandoned as a tool for representing the hierarchy relations, in favour of thin, coloured tape. It was hoped that the tape should support the dynamic development of the model because it could be stuck down, removed and replaced at will. In practice, use of the tape suffered from the main problem which dogged use of the analysis component cards. Both cards and tape were too time consuming, distracting and cumbersome to prepare and to use. Cards had to be selected, carefully inscribed with marker, placed on the analysis surface, as often as not in a location which was already occupied by another card. Whole sections of the model often had to be moved, card at a time, when even a minor change was made to the represented task structure. Anything more than minor changes to the labelling of a card required the rewriting of the entire card. An appropriate length of tape had to be unrolled, cut and stuck in place. Whilst the tape had the advantage over marker that it was erasable and movable, this was not particularly convenient, still less when several components of the model were moved simultaneously.

Hence, the mechanics of using the modelling tools interfered with the processes of analysis and modelling. Interaction between the participants was broken and erratic. As a fresh point was made or an old one reviewed, the developer would often stop the discussion while he busied himself manipulating cards and scissors and markers and tape. Flowing interaction and analysis of the user's work situation was common only during *post hoc* reviews of a section of the model. Even during these periods there were interruptions when an issue raised required an amendment

to the model. It is of course inevitable in analysis and modelling work that a model must be revised as the analysis progresses. The point here is that the tools for that revision failed to support smooth transitions between analysing and modelling.

A premise of the approach was that users should be directly involved in the *modelling* of their tasks. A tight binding between modelling and analysis in which the two proceed together offers advantages of continuous and very rapid verification and validation of models by the user. However, the difficulties encountered in the pilot study suggest that it is a mistake to begin the combined modelling and analysis sessions 'from cold'. The user had to pick up the skills of modelling while the developer gathered information about a work domain which was wholly new to him and while both were taking the first steps in working together. This may have placed too much load on both user and developer. A recommended approach is to precede cooperative task modelling with some information gathering and introduction to modelling ideas by the developer.

From this account, it may be seen that the hoped for benefits of the use of analysis component cards to support cooperative task analysis went largely unrealised. A sense of direct collaborative participation by the user in the development work was not imbued through direct manipulation of the cards. Despite her being asked to do so, the user remained reluctant to label cards or to place or to move them on the analysis surface. The user was willing to propose where elements of the model should be placed and how they should be labelled but almost all the physical work of generating the model was performed by the developer.

The unwillingness of the user to manipulate the elements of the model suggests that, despite the use of familiar stationery, the model remained intimidating to the user. The apparent failure of familiar materials to have the reassuring effect which they are often assumed to imbue (e.g. [Ehn and Kyng, 1991; Muller, 1992]) may be a result of the unfamiliar way in which those materials are being used. Users may be very familiar with the use of various forms of stationery *for specific purposes* in their daily lives, but they can rarely be familiar with their use as building materials for task models - or for software interface designs.

Multiple, tentative, quickly changeable representations were often required, particularly in the early stages of analysis and model construction. The above account has demonstrated that the analysis component cards were anything but mobile and easily manipulated and did not facilitate early attempts at construction of the model.

Following the pilot study, the basic requirements for an approach to task based participatory development remained the same: to ameliorate the technological intimidation of users, to facilitate their direct, active involvement in the development work, to facilitate their manipulation and use of the development artefacts. The pilot study suggested that the use of analysis component cards and stationery largely

failed to meet these requirements. As noted above, the familiarity of the materials was irrelevant to the specific context of use in the task analysis work.

Moreover, the problems drawn out by the pilot study were primarily associated with the media and tools used to support the task modelling work, rather than with the notion of user participation or the concepts and notations of task modelling. The only task modelling concept with which the user was uncomfortable was that of 'primary goal' but this was easily resolved when it was recast as 'primary task'. There were also some difficulties in making notations explicit but these were mainly due to the limitations of the media in carrying the notations.

The main problems revealed by the pilot study were in the poor support provided by the task modelling tools for flowing, dynamic cooperative modelling. It may be speculated that the analysis component cards and associated tools may be more effective in later stages of the modelling work, with more stable models. However, the pilot study did not get that far down the development track. This was at least in part due to the slow progress made with the early analysis and modelling work, which is likely to have been a feature of any prolonged development effort using these tools, along with loss of user faith and good will. The two major projects, conducted under the time pressures of real world development, abandoned the card based task modelling in favour of more quickly and easily manipulated media. This is described in the following sections.

3.2.2 CI project overview

The first major development project studied grew out of the development company's established interest in the domain of criminal intelligence and investigation work. Commercial sensitivities and confidentiality preclude a detailed account of the application which was developed and certain detailed aspects of the development work. However, the following account provides enough background to set the scene for subsequent chapters of the thesis. The company already sold one application which supported a range of activities in the domain. A major government department published a call for a further application to cover a range of other activities in the domain. Initially aimed at providing a response to this call, the project eventually refined its requirements to include the production of a more widely marketable system. The development work on which this analysis is based was aimed primarily at producing a system to support the operation of police major enquiry incident rooms. The project had between five and ten people actively involved in development activities at any one time over a three year development period from project inception to market launch of the system. The author was involved in the project as a participant-observer [Spradley, 1980] and encouraged cooperation between users and developers throughout the project.

There was quite a strong culture of marketing led commercial sensitivity in the development company. This was exacerbated due to the project's perceived importance to the future expansion of the company's applications division and due to the simultaneous efforts of more than one competitor company to produce a

similar product. The company was very unwilling to allow outsiders to discover anything about the ongoing development work.

These concerns led to a reluctance on the part of the development company in the CI project to allow direct access to users from potential customer organisations. Ironically, the concern was not what the developers might learn from the users, but what the users might learn from the developers. As a result, contact with users of current systems at whom the product was aimed was restricted to a few site visits during the course of the development work and to evaluation of later prototype versions of the developed system. Hence, participatory work throughout the earlier analysis and design activities was conducted primarily with an employee of the development organisation.

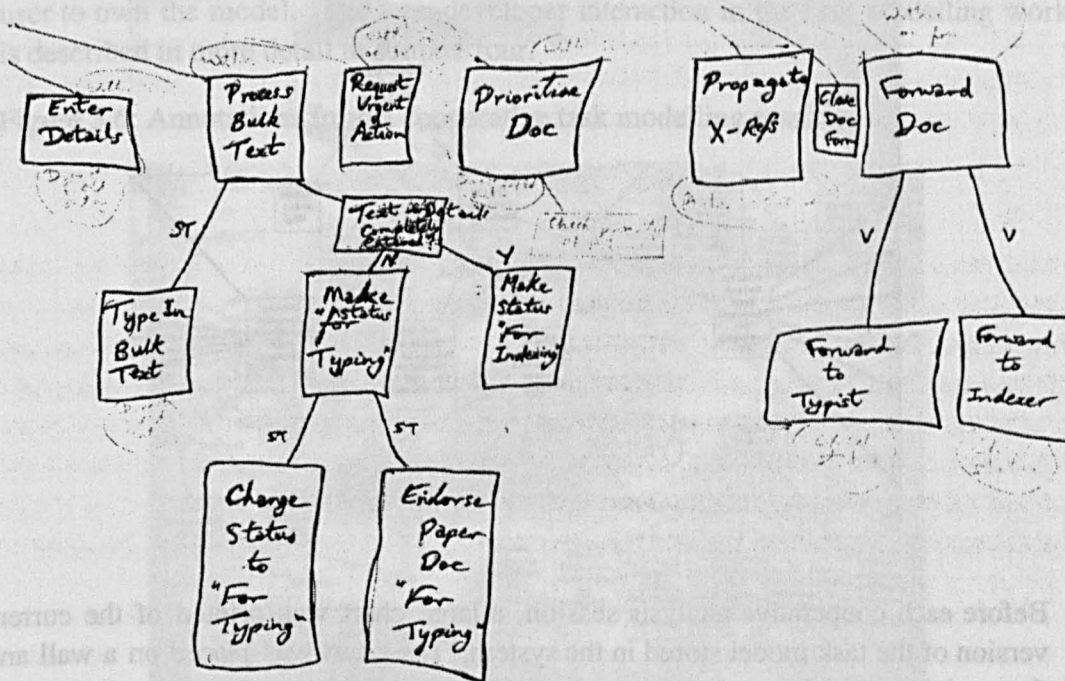
This person had decades of experience in all of the roles in the users' work situation. He had recently been hired by the development company to contribute his knowledge of the customers and their organisations to the sales department but it was quickly realised that his knowledge could also be of use in the development work. Most of the task analysis, task design and software design work was conducted with this person as a representative user. Other practising users and former users were occasionally consulted, including site visits by members of the development team. In addition, the representative user would from time to time consult other practising users if, for example, he could not provide a clear answer to a point or if there was some other confusion. In these cases, he would then report the users' views back to the other participants.

The issues around gaining access to real users or representative users and gaining the perspectives of a number of different users are widely discussed throughout the PD literature, e.g. [Bødker, 1996; Grudin, 1991b, 1993]. The heavy reliance on the perspective of a single user who was no longer a 'genuine' user may be considered a weakness of the CI project as a study in participatory development. However, this must be balanced against the advantages of his immense knowledge of the application domain, his ongoing contact with practising users and his high motivation and willingness to cooperate with the developers, both of which are often low with users drafted into participatory development work. Problems of access to real users are common in systems development practice (see chapter two). Rettig [1994, p.25] notes that when running software evaluations 'you can often get by with "surrogate users" - people who fit the same profile as your actual clients but free from whatever association that prevents you from testing with the clients themselves'. This is reflected in the CI study, giving it further real world relevance.

The first cooperative task analysis phase began with initial data gathering through semistructured discussions amongst the latter employee (hereafter referred to simply as the user), the chief developer for the project and the author. These discussions centred on the users' current work situation and tasks. Drawing on the lessons from the pilot study, this early development work eschewed the card based approach to participatory task analysis. As the developers began to share tentative

and partial understandings of elements of the users' work, pencils and paper - and erasers! - were used quickly to sketch partial models of the users' tasks. Initial sketching began during the participatory discussions and was continued by the developers afterwards.

Figure 3.4: Initial task model sketch from CI project

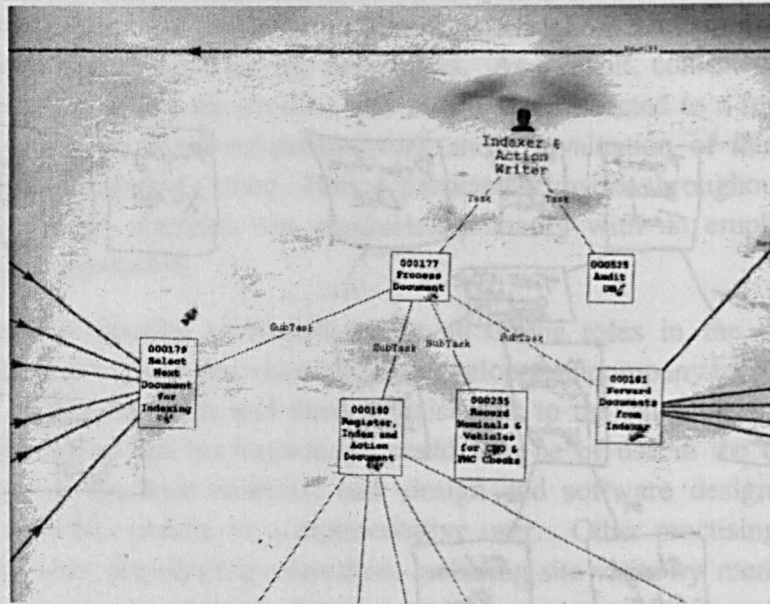


The developers - in the absence of users - turned these initial sketches into a first pass at sections of a task model (TM1) (see Figure 3.4). The exclusion of users from this initial modelling work was motivated by the developers' strong view that users need a concrete representation with which to begin development work (see also chapter two and section 3.2.1 of this chapter). The developers' construction of the initial models served to 'bootstrap' the participatory task modelling work. These early, tentative and partial models introduced the user to the concept and practice of task modelling and provided a basis on which to begin the construction of a more sophisticated, complete and integrated model of the users' current work situation.

The initial data gathering phase of the CI project provided the developers with the beginnings of an understanding of the work situation into which the system was to be introduced and some partial models of the current tasks in that work situation. In preparation for the next phase, the chief developer entered these partial models into a software system which allowed the storage, editing and graphical display of linked data. From this, a wallchart was printed showing a task model of the present understanding of the overall work situation. This chart was used during the subsequent cooperative analysis work as a basis for the construction and refinement

of a comprehensive task model of the users' work situation, including roles, tasks and workflows. Part of the chart is illustrated in Figure 3.5.

Figure 3.5: Part of a task model on a wallchart from the CI project



Before each cooperative analysis session, a large chart was printed of the current version of the task model stored in the system. The chart was placed on a wall and the participants in the modelling session worked on it. Amendments and additions were annotated on the wall chart using pencils or pens. After each modelling session, one of the developers updated the electronic version of the model to reflect the manuscript annotations. In this way, over a period of several months, a single large task model was produced of the users' current work situation.

Pencil and paper proved to be much more efficient media for task modelling than the analysis component cards. The mechanics of using the modelling tools did not interfere with the processes of analysis and modelling. Interaction between the participants was dynamic and largely uninterrupted by the modelling activities. As a fresh point was made or an old one reviewed, they could quickly amend the developing models with pencil and eraser. Figure 3.6 illustrates annotations made on a wallchart task model.

The cooperative task analysis work was viewed as a success by all those involved. The participants built up strong working relationships, the user felt actively involved in the analysis and modelling work and the developers achieved a detailed understanding of the users' current work situation. The task model provided a resource on which subsequent design work could be based. During the analysis and modelling work, the model served as a focus for the cooperative development activities. The user learned and used the concepts of task modelling while the developers learned and used the concepts of the application domain. The notation

participants. Hence, by the end of a session, the model was covered with markups. At times, this could make the model quite difficult to interpret. Also, participants generally saw a clean printed copy of the most up to date version of the model only at the beginning of the session following the session in which that version had been developed.

Having produced a comprehensive model of the users' current work situation in TM1, the development team faced a difficulty inherent in all systems development methods: crossing the design chasm from an understanding of the current situation to a design for an envisioned situation. On one occasion at this point in the project, the chief developer admired the task model and then asked, 'so what do we do now with it?'

One copy of the electronic TM1 was 'frozen' and printed as the model of the users' current work situation and a second copy was generated on which to base the transition to TM2, an envisioned task model. The freezing of TM1 does not, however, imply that it was entirely static. Additions and revisions to it continued to be made as the developers' understanding of the current work situation continued to develop - although there was progressively less rigour in keeping TM1 up to date as work on TM2 progressed. The second copy of the task model was used as a basis from which to begin cooperative task modelling of an envisioned work situation.

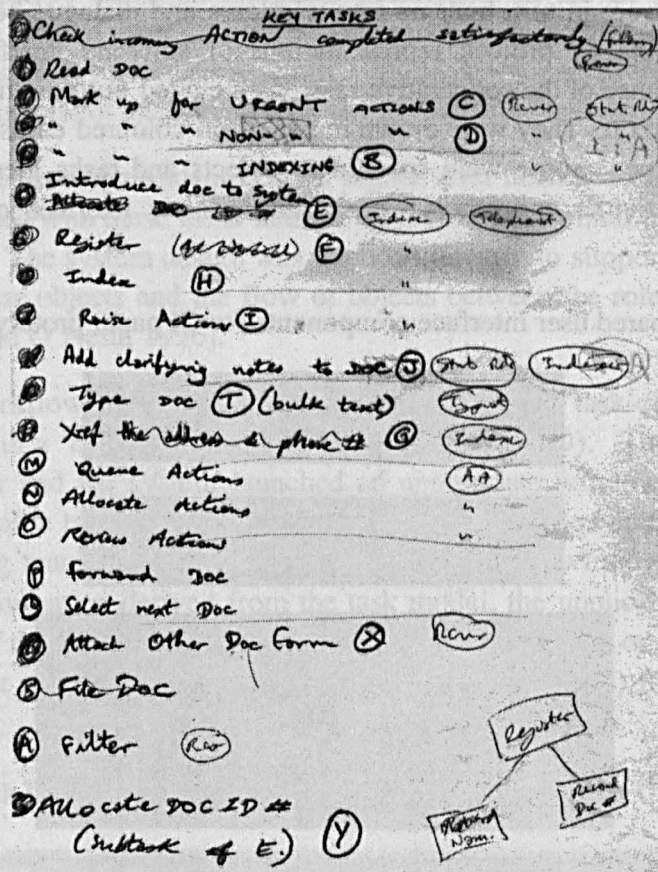
A wallchart of TM1 was produced and the participants used this to identify 'key tasks' in the users' current work situation. These were tasks which demanded support from an envisioned system design. Two main criteria were used in identifying these tasks. First, some tasks, at various levels of decomposition, were identified by the developers as necessary for goal achievement. That is, these were tasks without which the system's job could not be done. Secondly, the user identified certain tasks as central to his (mental) model of the work. Figure 3.7 illustrates an effort at identifying key tasks from TM1 in the CI project.

TM1 included a comprehensive modelling of workflows amongst roles and tasks in the work situation. Having identified key tasks, the participants noted workflows which linked those key tasks and the roles in which they were performed. This subset of TM1 then provided a skeleton of essential tasks and workflows from which to derive a first cut model of an envisioned work situation. A considerable amount of redundancy was apparent in the tasks modelled in TM1, within many roles and at many levels of task decomposition. Since one of the overriding requirements for the new system was efficiency, an important design goal in deriving the envisioned task model was to eliminate much of this redundancy. Identifying and carrying forward key tasks assisted in this process.

Development of TM2 from a first cut of key tasks and workflows was conducted with the same media, tools and participants as the construction of TM1. The developing model of an envisioned work situation was stored electronically and

printed on a large wallchart for each cooperative modelling session. Pencils and erasers were used as the main tools for updating and amending the model.

Figure 3.7: Key tasks identified from TM1 in the CI project



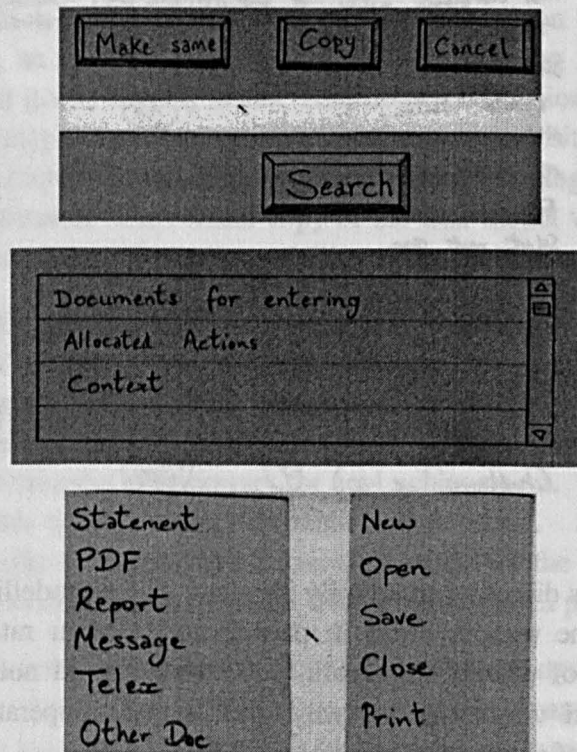
Again, the user was directly and actively involved in the modelling activity. In this phase, however, he was involved in participatory design rather than analysis - contributing ideas of what should work well, what should not appear in the new work situation and why. In this way, the second cooperative analysis phase delivered a model of envisioned user tasks (TM2).

In turn, in pursuit of the objectives of minimising layers of mediation and bridging the design chasm, envisioned software components were based on envisioned task model components. Two facets of the task model - its elements and the structure of those elements - mapped, respectively, to two elements of the software - again, elements and structure. The elements of the software interface were derived from the object classification and decomposition within the model of the envisioned work situation, while the structure of the software interface was derived from the envisioned task structure in TM2.

The software design activities also encouraged active user participation, with the cooperative scenario based construction of paper prototypes. For the cooperative

paper prototyping, the author made available coloured cards, pads of Post-It™ notes, scissors and pens as generic components and prepared user interface elements from coloured card as specific components. Elements which appeared together in several places across the task model were grouped in what was termed a panel. Panels were in turn built up into windows and dialogue boxes. For paper prototyping in the CI project, representations of software interface components prepared in advance of the cooperative sessions included buttons, menus, title bars, list boxes and so on. They were drawn in pencil on coloured cards which were cut to size and shape. Some were specific to objects and tasks identified in earlier development activities and others were blank, generic interface components (see Figure 3.8).

Figure 3.8: Prepared user interface components for CI paper prototyping work



The paper prototyping work was strongly guided by the preceding work on analysing the users' current work situation and, in particular, designing the envisioned work situation in TM2. Having worked through the previous activities together, the participants were not attempting to piece together components of a software design in a vacuum. Besides the explicit derivations of software objects from task model objects and software structure from the modelled task structure, the participants had implicit access to a shared understanding of the envisioned system developed through their earlier task design work (see chapter four). As the chief developer commented later, 'The task model was really useful because, even when

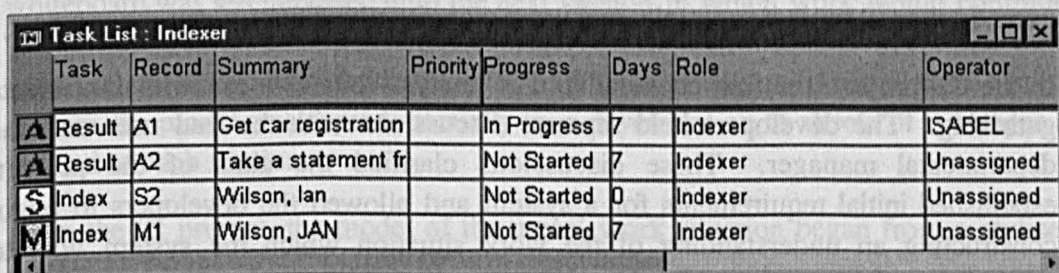
we weren't referring to it directly, it was always there at the back of my mind right through the design and implementation'.

The next phase of the development work in each project was the implementation of a software prototype based on the paper prototype. In the CI project, this was performed by a team of three developers, one of whom was chief developer and had led the cooperative development sessions. These developers were joined by other colleagues to implement later prototypes and the delivered system.

From TM2, the developers identified the higher level tasks for each user role, the objects associated with these tasks and the workflows by which the objects moved between roles. The system design was intended directly to support performance of the tasks on these objects and the flow of objects between the roles performing the tasks [Smith and O'Neill, 1996].

Hence, the workflow system in the application displayed task-object pairings, in priority order, in a 'task list' for each user role (Figure 3.9). The user selected a task-object pair and the system launched an appropriate window for the user to perform the chosen task. Upon completing the task, the user updated the status of the object being manipulated and then simply closed the window. Based on the object flow information derived from the task model, the application automatically set the status of the current task to 'complete' and initiated one or more consequent new task-object pairings which then appeared in the task list(s) for the appropriate user role(s).

Figure 3.9: Task list for a user performing the indexer role in the CI system



Task	Record	Summary	Priority	Progress	Days	Role	Operator
A Result	A1	Get car registration		In Progress	7	Indexer	ISABEL
A Result	A2	Take a statement fr		Not Started	7	Indexer	Unassigned
S Index	S2	Wilson, Ian		Not Started	0	Indexer	Unassigned
M Index	M1	Wilson, JAN		Not Started	7	Indexer	Unassigned

The implemented prototype system went through several quick iterations to eliminate obvious bugs and was then subjected to a usability evaluation. This evaluation exercise used the 'Cooperative Evaluation' method [Wright and Monk, 1991]. The evaluation work and its results are reported in chapter six. The author's active involvement with the development project ceased with the delivery to the developers of a report on the results of the usability evaluation.

Following the evaluation exercise and based on its results, the chief developer prepared a list of proposed design amendments. These were implemented and, following further iteration and refinement, the implemented system was released for beta testing at selected customer sites.

3.2.3 CS project overview

The second project studied was the development of a technical query and fault reporting system for a university computing support department. This project had between three and eight people simultaneously involved over an eight month period from inception to installation of the implemented system. The author was involved in the project as a participant-observer and again encouraged cooperation between users and developers throughout the project.

The CS project was conducted for and within the Computing Services department of Queen Mary and Westfield College. The system was intended to include inventory of computing equipment, technical query and fault report handling, management statistics and reports generation. A previous in-house attempt at this development had failed largely due to lack of experience and time constraints. For this project, one user from the department (the 'lead user') was appointed by the departmental management as responsible for the project. He hired an external developer to design and implement the system and arranged the involvement of the author to help with work analysis and HCI design. The author took a more leading role in the development work in this project than in the CI project due to the inexperience of the other developer. Participatory work throughout the project involved the lead user and a varying number of other users from throughout the department.

Commercial sensitivity in the CI project raised issues of access to and publication of data. There were fewer such problems in the CS project with its in-house non-commercial work. In the CI project, again motivated by commercial sensitivity, most of the cooperative analysis work was conducted with only one representative user. In the CS project, multiple, real users were involved throughout the project.

In the CS project, the first cooperative task analysis phase began with initial data gathering. The developers held separate discussions with the lead user and the departmental manager. These discussions clarified the aims of the project, established initial requirements for a system and allowed the developers to begin constructing an understanding of the work situation which the system was to support. The requirements for the system included support for (i) a fault report and technical query database, (ii) an online inventory of computing equipment and (iii) generation of management reports on workloads. It was agreed very early in the project that the last element could not be produced within the six weeks originally allowed for analysis and design work. The inventory was not viewed as a priority and, in the end, only the technical query and fault report facility was built in the development period reported here.

Semi-structured interviews were then conducted with fourteen users from throughout the department. From the interviews and ongoing discussions with the lead user, the developers had begun to derive a broad understanding of the users' current work situation. They now applied the cooperative task analysis approach to modelling this work situation. All employees in the department were invited to

participate in the development work. Several email invitations were distributed around the department and these were supplemented by oral and manuscript invitations to several users. The invitations to participate met with a mixed response.

Some users were very keen to participate in the development work, for a variety of reasons. Reasons included simply being curious or excited by the idea of developing a new system, wanting to ensure that their requirements were met, disliking the existing systems and wanting to ensure that the new system was different and feeling that they had to participate to represent their section of the department. Some users simply ignored repeated invitations to participate, a few flatly refused to participate and one kept promising to participate but never quite got around to it.

The initial discussions and interviews revealed five user roles on whose work the technical query and fault report system directly impinged: reception, liaison, fault line, help desk, technical consultant and network services. These roles were identified by the users and corresponded only roughly to formal job distinctions in the department, with several users performing more than one role. They were based on mixtures of physical location, departmental structures and common tasks. The cooperative task analysis work began with the developers working in turn with users who worked in each of the roles.

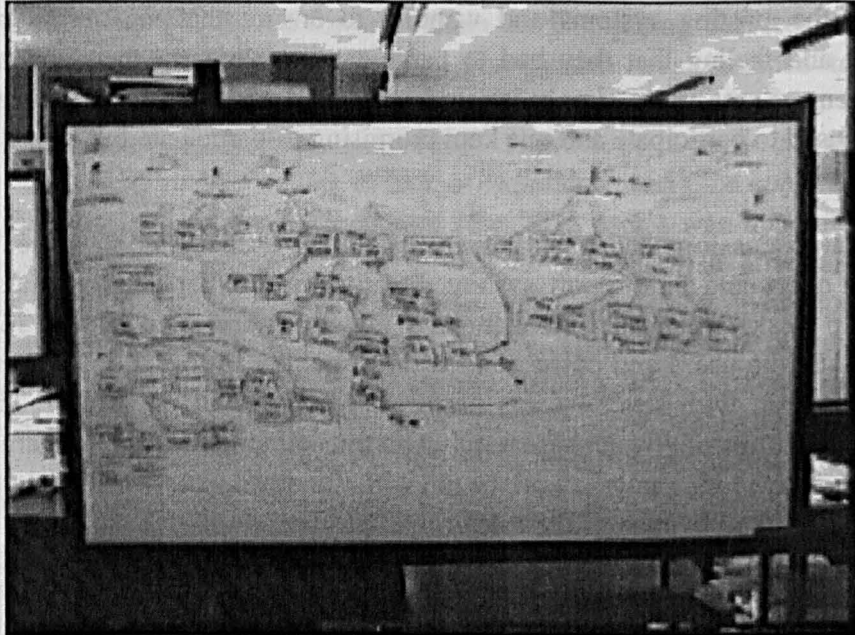
There was another change of task modelling media for the CS project, the models of the users' current and envisioned work situations being produced with a whiteboard and markers (see Figures 3.1 and 3.10). At the end of each modelling session, the whiteboard was left unerased until the next session in which work would continue. The difficulty of producing multiple copies of the model (in contrast to the electronic version in the CI project) made the model vulnerable to loss. As insurance against erasure, the model was copied manually by a developer on to sheets of A4 paper at the end of each session.

As in the CI project, the model of the users' work situation began from a stylised sketch of the physical layout of the users' working environment. The sketch included stick figure representations of the roles described above, their relative working locations (represented by their relative placement in the sketch) and flows (represented by arrowed lines) of work objects, documents and so on, amongst the people.

Once again, primary tasks associated with a role were denoted by labelled boxed nodes connected to the role by arcs. The cooperative task analysis work with each user concentrated on modelling the work performed in the one or more roles filled by that user. For example, one session was devoted to cooperative modelling of a technical consultant's tasks, while another session was devoted to cooperative modelling of a receptionist's tasks.

Having separately developed parts of an overall model of the users' work situation, it was necessary to integrate the parts into a single comprehensive model. This helped in checking that users from across the work situation agreed that workflows and interactions between roles had been modelled accurately. Such agreement was readily achieved for TM1 but not so easily for TM2.

Figure 3.10: A task model produced on a whiteboard in the CS project



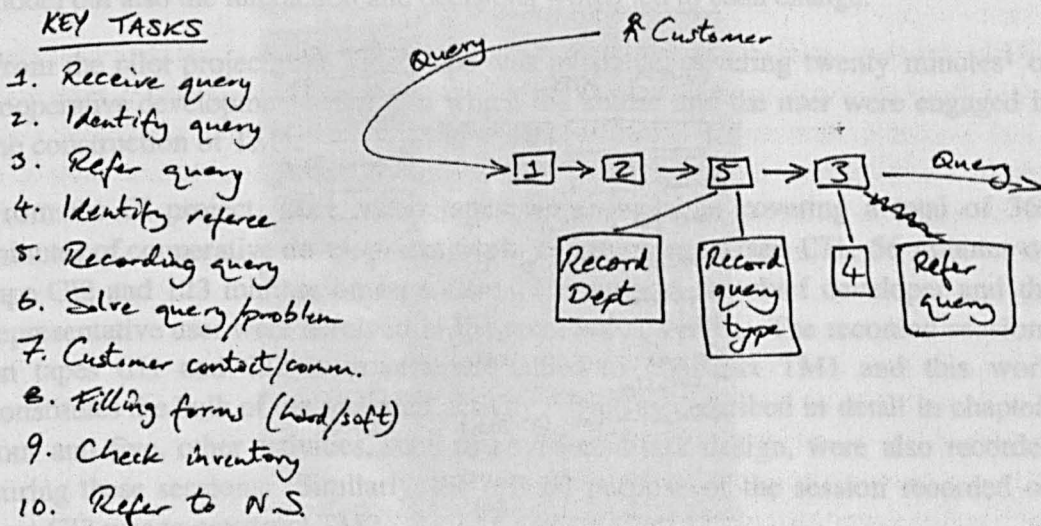
One advantage which the whiteboard based models had over the models in the CI project was ease of amendment. During the cooperative task modelling sessions, amendments and additions to the task models in the CS project were made using whiteboard markers and erasers. This was fast, reduced ambiguity by immediately erasing unwanted clutter from the modelling space and ensured that participants were constantly working with the most up to date version of the model. In contrast, the task models in the CI project were not so easily or clearly amendable. Markups were made using pen or pencil on the printed wall chart and the original printed parts of the model could not be erased by the participants. A disadvantage of the use of whiteboard and markers was that, apart from manual copies, previous versions of the model were not recoverable.

The task model in the CS project was less portable than that in the CI project even at the level of the single model since the wallchart could be rolled up and displayed in a new location more easily than a large whiteboard could be moved around. In a similar manner to the user in the CI project requesting an electronic copy of the task model, a user in the CS project also requested individual copies of the model for himself and his colleagues in order that they could work on the model independently between sessions. This necessitated taking a manual copy of the model and photocopying it. Photocopies of the pen and paper version were then distributed to

participants. Amendments made using pen and photocopy were discussed and integrated with the whiteboard representation at the beginning of the next session.

Movement from TM1 to TM2 in the CS project again involved bridging the design chasm by identifying key tasks in the users' current work situation from TM1. Having identified key tasks, the participants identified workflows which linked those key tasks and the roles in which they were performed. This subset of TM1 then provided the skeleton of essential tasks and workflows from which to derive a first cut model of an envisioned work situation (see Figure 3.11). Development of TM2 from a first cut of key tasks and workflows was conducted with the same media, tools and participants as the construction of TM1.

Figure 3.11: Key task-workflow model from the CS project

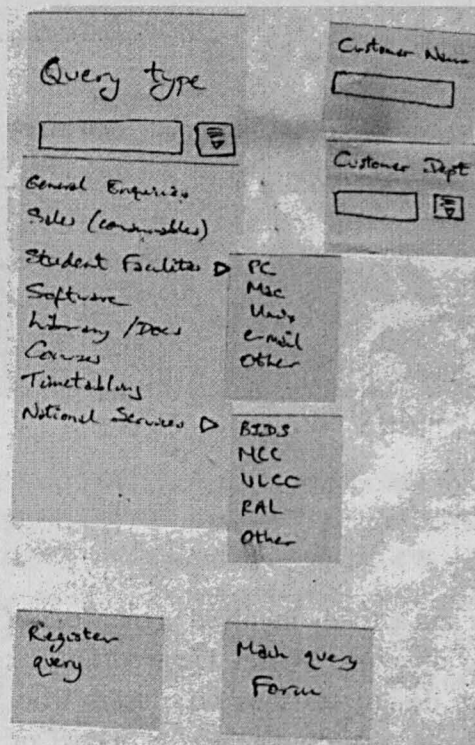


The users were directly and actively involved in the work of producing a task model of the envisioned work situation. Again, tasks performed in particular roles were modelled in cooperative design sessions with users who worked in those roles. When the constituent parts of a comprehensive model of the work situation had been produced, users were brought together in a larger group to integrate the parts into a whole.

The software design activities again encouraged active user participation, with the cooperative scenario based construction of paper prototypes. In the CS project, the paper prototyping was more informal, simply using pens and blank Post-It notes to construct interface designs on a blank sheet of A3 paper. The cooperative prototyping sessions began with completely blank Post-It notes and an A3 sheet of blank paper. During the cooperative prototyping activity, interface components were drawn freehand on the Post-It notes which were then placed on the paper which represented the computer screen (see Figure 3.12).

The paper prototyping work was strongly guided by the preceding work on analysing the users' current work situation and, in particular, designing the envisioned work situation in TM2. The elements of the software interface were derived from the object classification and decomposition within the model of the envisioned work situation, while the structure of the software interface was derived from the envisioned task structure in TM2. Besides the explicit derivations of software objects from task model objects and software structure from the modelled task structure, the participants had implicit access to a shared understanding of the envisioned system developed through their earlier task design work (see chapter four). Again, as in the CI project, the paper prototyping was for the most part adding detailed interface designs to already established task designs.

Figure 3.12: Paper prototype using Post-It™ notes in the CS project



The next phase of the development work was the implementation of a running software prototype. In the CS project, early software prototypes were implemented by the developer who had been hired at the start of the project. His contract expired and a new developer was hired for a short time. The latter worked with the lead user in completing the implementation. The first running version of the system was released to the users and six weeks later a usability evaluation was run, again using the Cooperative Evaluation method [Monk, Wright, Haber and Davenport, 1993]. Details of the evaluation and its results are given in chapter six. Following the evaluation, the author gave a copy of the evaluation results to the lead user and had no further active involvement with the project.

3.3 Data available from the projects

Sources of data available from the projects were extensive and heterogeneous. They included video records of cooperative development meetings from the projects. There were interviews with the participants in the development sessions and external artefacts produced as a result of the development activities, including task models, prototypes, the implemented software, usability evaluation reports and project databases containing requirements specifications, change requests, bug reports and so on. Other available external artefacts were personal, informal products of the development activities, such as notes and sketches. In addition, artefacts from the users' current working environments were available for analysis.

The video records of cooperative development sessions captured the progress of the task and design models through the sessions, recording not just each change to the model but also the interaction and decisions which led to each change.

From the pilot project, one video tape was available, covering twenty minutes¹ of cooperative development activity in which the author and the user were engaged in the construction of TM1.

From the CI project, three video tapes were available, covering a total of 362 minutes of cooperative development work: 183 minutes on tape CI1, 56 minutes on tape CI2 and 123 minutes on tape CI3. The author, the chief developer and the representative user were involved in the recorded activities. The recorded sessions on tapes CI1 and CI2 were officially called to construct TM1 and this work constitutes the bulk of the recorded activity. But, as described in detail in chapters four and five, other activities, such as envisioned task design, were also recorded during these sessions. Similarly, the official purpose of the session recorded on tape CI3 was to construct TM2.

From the CS project, seven video tapes were available. A 79 minute tape, CS1, recorded the author, the contracted developer and a user constructing TM1. A 30 minute tape, CS2, recorded the author, the developer and a different user constructing TM1. An 18 minute tape, CS3, recorded the author and the developer discussing requirements for the system. An 80 minute tape, CS4, recorded the author and the developer deriving a model of the workflows through the key tasks and roles in TM1. A 70 minute tape, CS5, recorded the author, the developer and a user constructing TM2. A 109 minute tape, CS6, recorded the author, the developer and five users constructing TM2 and an accompanying object decomposition model. A 72 minute tape, CS7, recorded the author, the developer and a user constructing a paper prototype of the envisioned software. In addition, a 40 minute audio tape was available of the author, the developer and a user who refused to be video recorded constructing TM1.

¹A recording of a further three hours work in the pilot study was lost due to technical problems - an early lesson in the hazards of video analysis.

The recorded development activities represented only a tiny fraction of the work that went on during the two projects. The CI project lasted around three years from inception to first application release. Most of the task modelling was conducted over a period of three months. The paper prototyping sessions were held about two months after the main task modelling work had ended. A further six months later, evaluations were conducted on a running software prototype. Finally, around eight months later, the first commercial version of the software was released.

The CS project lasted around eight months from inception to delivery of the implemented system. Most of the task modelling and paper prototyping work was conducted in a two month period. This was followed by a six month period in which software prototypes were developed and refined into the eventual system. The time periods given here for the activities in each project are of course approximate since there was considerable iteration and review of earlier activities.

From the pilot study, the partially completed TM1 was also available for analysis (see Figure 3.3). From both the CI and CS projects, a large amount of paperwork was available, including documents and artefacts from the work situations which were used during the development activities. These included copies of recommended working procedures and practices, departmental telephone and room lists, preprinted forms from current paper and software based systems, and user manuals. Further documents and artefacts were produced and made available as a result of the development activities. These included informal products, such as personal notes and sketches, and the formal artefacts of the development work, the task models, paper prototypes and the implemented systems themselves. Also from each project, there were reports from the usability evaluations of the implemented software.

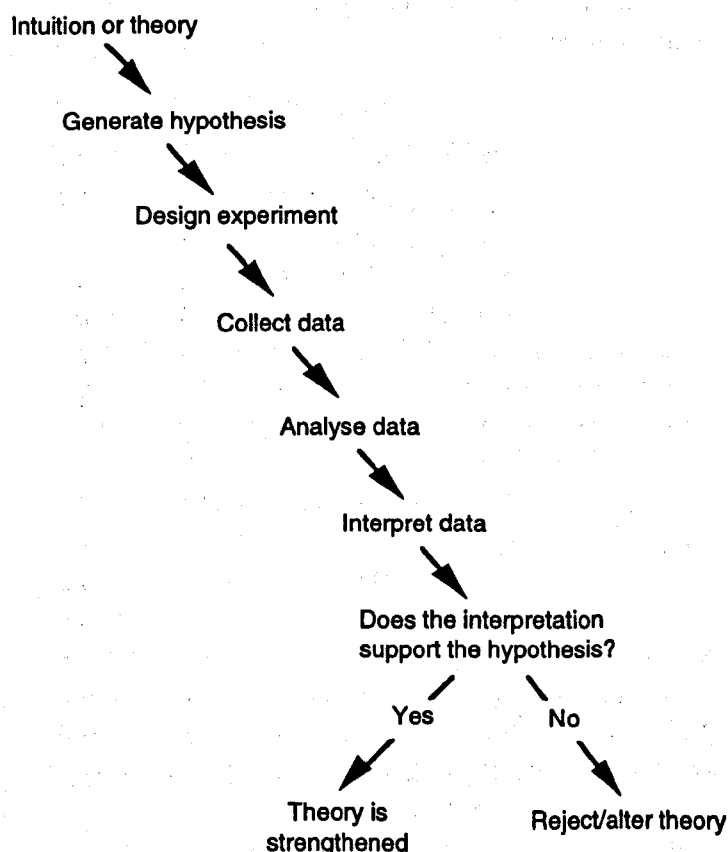
The video records and artefacts were not, however, analysed as isolated, decontextualised chunks of data. The results both of development activity (e.g. a task model) and of research activity (e.g. a video tape) were complemented by insights and results from interviews and conversations with the participants in the development sessions, contemporaneous notes of development work and, crucially, the analyst's involvement in the projects as participant-observer. The uses to which the data were put are described in the following section. Illustrations of example artefacts and transcript fragments appear throughout the thesis.

3.4 Research method

The introduction to this chapter noted that issues to be tackled included the nature of user-developer interaction in cooperative development settings, the effectiveness of user participation in contributing to the development process and the impact of user participation on software usability. In the course of this research, these issues motivated the research questions presented at the end of chapter two and were addressed through empirical studies of real world software development projects.

The strong cognitive science tradition in HCI has encouraged an experimental approach to empirical research. This approach may be characterised as in Figure 3.13. An initial intuition, based on current theory, is formalised into an experimentally testable hypothesis. A controlled experiment is run to generate data. The researcher puts an interpretation on the data or, more commonly, on the results of statistical operations performed on the data. A discussion then ensues as to whether or not the interpretation arrived at supports the original hypothesis. Assuming the experiment not to have been flawed, the underlying theory is then developed according to whether the hypothesis has been strengthened or refuted.

Figure 3.13: Standard experimental approach



Hypothesis testing experimental approaches are most appropriate to tackling research questions where there is an established theoretical base on which to found hypotheses and controlled experimental situations can be set up in which to test the hypotheses. Many of the successes of HCI research have applied an experimental approach to research questions on cognitive processing and relatively low level task planning and execution, areas in which it is feasible to establish the prerequisites for an experimental approach. However, as described in chapter two, the remit of HCI research has continued to expand and this expansion has rendered the experimental paradigm infeasible or impossible to apply to many areas of interest within the field. Even in the laboratory setting, the human activity element of the objects of study

must be kept to relatively simple, well defined tasks in order to maintain experimental control. In the organised chaos of a software development project, such control is impossible and the experimental method cannot readily be applied.

The difficulties inherent in empirical, especially experimental, studies of systems development projects are widely acknowledged. For example, Johnson and Johnson [1990] note difficulties with studying real development projects as the time constraints on both developers and researchers in projects which may take months, the lack of susceptibility to study of some development activities and the confidentiality restrictions imposed on many projects. Shneiderman [1992, p.474] states plainly that 'the social and political environment surrounding the implementation of a complex information system is not amenable to study by controlled experimentation'.

Herbsleb and Kuwana [1993] wanted to test the hypotheses embodied in software development tools for knowledge capture but resignedly noted that

it is very difficult to test these hypotheses directly, i.e. by building an appropriate tool then designing a software system and assessing the results. The expense, risk, and the difficulty of interpreting the results of complex processes in the real world make this option untenable. Laboratory studies solve some of these problems by isolating the effects of selected variables, but ... there is generally no realistic organizational or project history in a laboratory context. Preserving small quantities of knowledge for the duration of a typical experiment, i.e. an hour or two, is radically different from preserving potentially enormous quantities of knowledge for more realistic time periods of months to years.

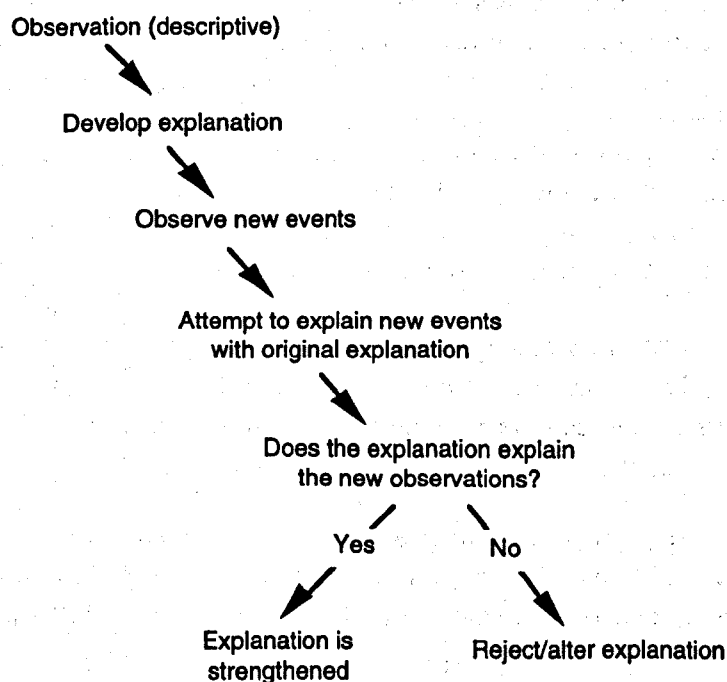
[Herbsleb and Kuwana, 1993, p.8]

Christel and Kang [1992, p.39] note that 'evaluation studies of development methodologies are plagued by the difficulty of controlling for developers' experience and level of creativity, and the unlikelihood of commercial developers employing parallel designs for the objective comparison of methodologies'.

As may be seen from these examples, most of the authors who refer to the difficulties of studying real world development work note the impossibility of running controlled experiments. However, another difficulty in applying an experimental approach to this type of research is the absence of established theory on which to base hypotheses. As the scope of HCI has expanded from its earlier areas of strength in the cognitive tradition, the development of a solid, unifying theoretical base has not kept pace. There is now a growing consensus that HCI is in a 'pre-theoretical' stage. Dixon [1987] argues that this is true of design research in general while, for example, Long and Dowell [1989] and Barnard [1991] make similar arguments for HCI in particular. More recently, Kuutti [1996] and Kaptelinin [1996] also discuss this lack in HCI.

The relatively recent expansion of HCI into the study of software development processes and activities has not yet allowed its theory generation and testing to catch up with the production of prescriptive and automated 'aids' to software development. This conclusion is supported by the results of the surveys reported in chapter two which found large numbers of prescriptive methods and tools for systems development and almost no real developer use of or interest in them. This is also borne out by Bellotti [1988] and by the work of Meister [Meister and Farr, 1967; Meister, 1987], suggesting that this is a problem of long standing and is not confined to software design. 'It is impossible to assist the designer effectively without understanding how he designs, because any aid provided must match his design processes. The repeated failures of human engineering guides to elicit designer interest and use may well have resulted from ignorance of those processes' [Meister, 1987, p.229].

Figure 3.14: Observational approach



More specifically to the objects of this research, there is a dearth of theoretical accounts of the interaction between user and developer in cooperative development settings. As noted in chapter two, the participatory design literature provides more coarse grained, pragmatic descriptions of user-developer interaction, rather than fine grain theoretical accounts. In the absence both of a solid theoretical base from which to derive and test hypotheses and of the controlled conditions necessary to perform such tests, this research demanded an alternative approach to the empirical analysis of software development projects. Such an alternative, observational approach may be characterised as in Fig 3.14.

In this approach, the analyst first observes the object of study, often complex human or animal behaviour. The researcher's first accounts are simply descriptive, noting what is happening without attempting to explain it. As the observer begins to identify patterns in the noted events, she begins to generate possible explanations for the behaviour. These explanations are the beginnings of a theoretical account of the observed phenomena. As further events are observed, they are interpreted in terms of the extant explanations. As Minneman [1991, p.157] notes, 'interpretations are considered valid to the extent that they are robust across other instances'. The nascent theory then develops according to whether or not it explains the continuing observations. Ultimately, the theory is well enough developed to be tested for its power in predicting the results either of systematic observations against a theoretically established baseline or of experimental manipulation of the phenomena of interest.

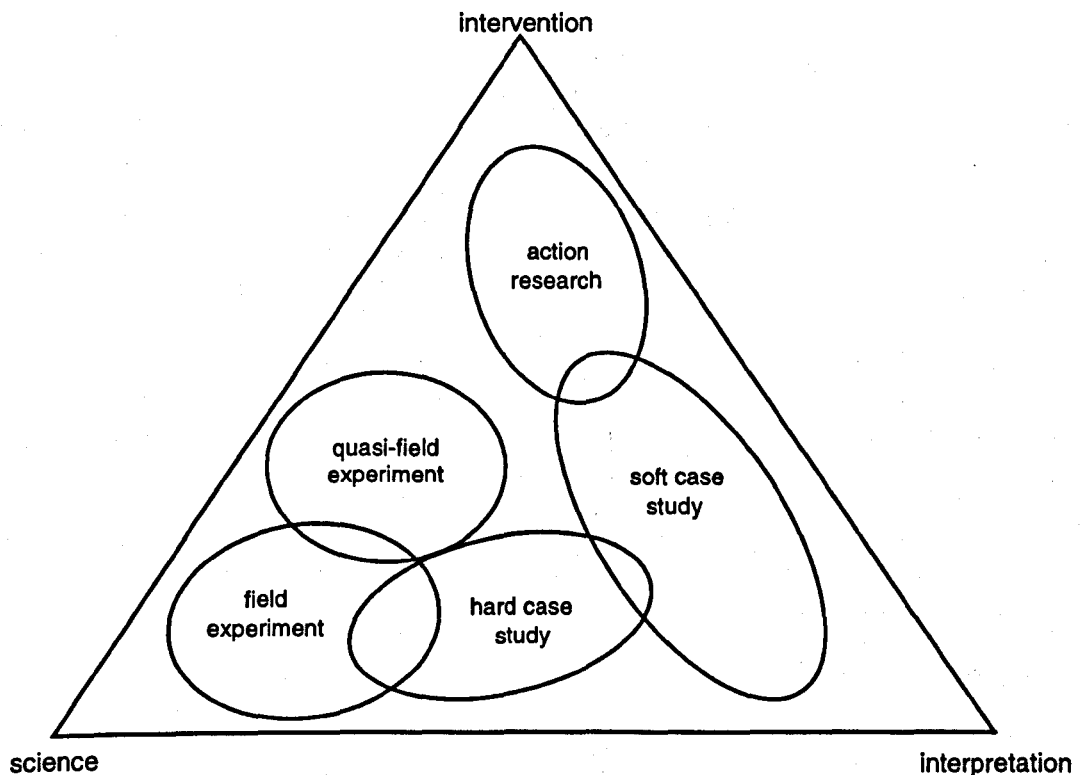
The observational and experimental approaches are in many ways complementary. The observational approach may be seen as imposing rigour on the pre-hypothesis stage of the experimental approach and the experimental approach as imposing rigour on the testing of theory suggested by the observational approach. The observational approach is appropriate when there is an absence of established theory on which to base testable hypotheses and/or a lack of opportunity to conduct such tests. 'In many cases, laboratory experimentation is inappropriate and formal modelling intractable; instead, observational data analysis is frequently the only appropriate empirical approach' [Sanderson and Fisher, 1994, p.251].

Hence, while an experimental approach can be more appropriate for theory testing, an observational approach can be more appropriate for theory generation. In order to move towards answering its quite high level research questions, this work had to describe and account for phenomena at the lower level of what actually was going on in system development meetings. At this finer level of resolution, there were no established theories of user-developer interaction in cooperative development, so an exploratory, theory generating approach was demanded. When such a theory has been established, it may then be appropriate to adopt an experimental approach to testing aspects of the theory, but that is for future work and an observational approach was adopted here.

Braa and Vidgen [1995] note that 'IS [information systems] research is situated in an uncomfortable space in which a variety of research methods are needed to reflect the relative objectivity of technical artefacts and the subjectivity of purposeful activity'. This dichotomy between the subjective and the objective runs through taxonomies of research methods such as [Galliers, 1992]. Braa and Vidgen [1995] distinguish three main forms of research method which may be applied to the study of systems development projects: case study, action research and field experiment. Braa and Vidgen [1995, p.54] note that 'the messiness of carrying out research in an "organisational laboratory" means that it is often difficult to remain faithful to the principles of a purified research method'. They identify three main concerns of

systems development research methods, science, interpretation and intervention, and propose a framework with the purified, ideal forms of these concerns as points delimiting a space of possible research methods (see Figure 3.15).

Figure 3.15: Space of systems development research methods

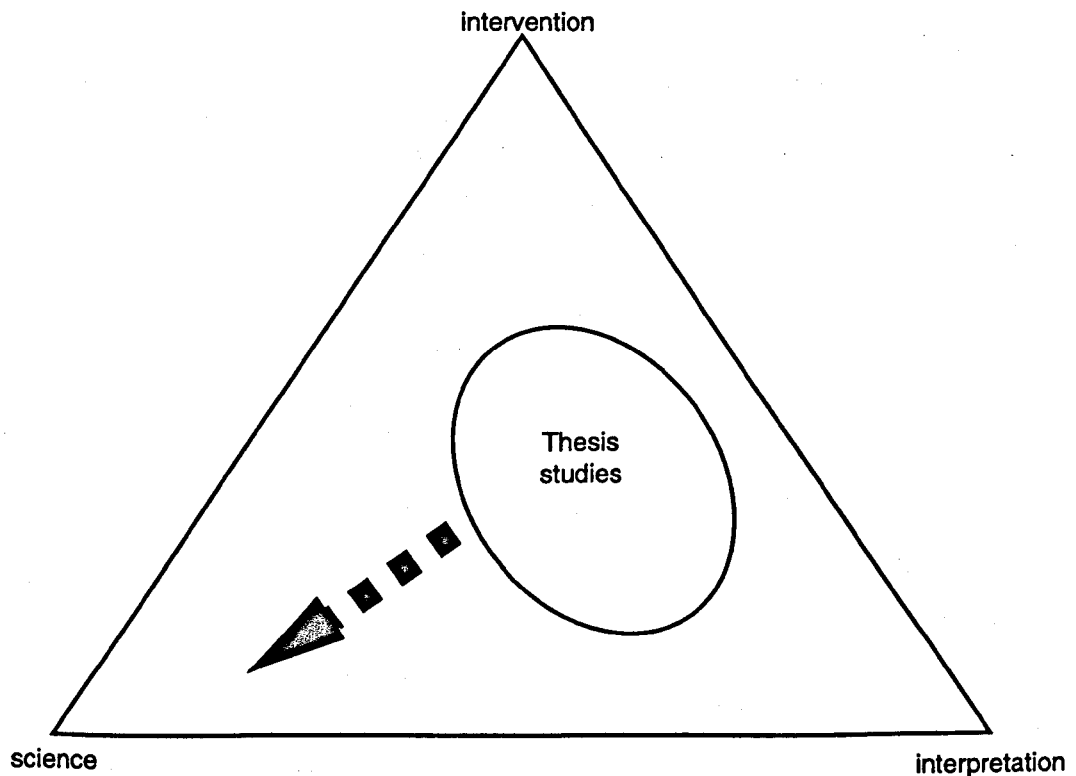


A research method towards the 'science' point should have greater explanatory and predictive power and statistical generalisability. The emphasis here is on reductionism, control, refutability and repeatability. Towards the 'interpretation' point, a method should provide greater understanding of systems development work in its organisational context. Here the emphasis is on maintaining the richness of social context and on the plausibility of arguments and reasoning. As a method moves towards the 'intervention' point, the researcher becomes increasingly less an observer and more an *unreflective practitioner* (cf. [Schön, 1987]), eventually becoming consultant rather than researcher.

Braa and Vidgen [1995] locate the notional forms of action research, field experiment and case study within the research methods space as shown in Figure 3.15. The difficulties in establishing even a notionally pure form of these approaches are reflected in their inclusion of 'hard' and 'soft' case studies and of 'field experiment' and 'quasi-field experiment'. The place of the research methods adopted in this thesis is illustrated in Figure 3.16. Initial work addressing research question one was located relatively close to the side representing the tradeoff

between intervention and interpretation. As the work developed through this and subsequent research questions (chapters four, five and six), the approach adopted moved further towards the hard 'science' point.

Figure 3.16: Locating the research methods of the thesis



3.4.1 Analysing interaction

The approaches adopted in tackling each of the four research questions were diverse but all were firmly based on a detailed analysis of the interaction between user and developer in the cooperative development setting. Video recording was adopted as a key means of capturing this interaction. Video records of the development meetings provided data for the analysis of the cooperative development activities - supplemented, as described in section 3.3, by other artefacts of the development and research processes and the author's participation in the recorded meetings.

Jordan and Henderson [1995] note several advantages of video records as data. Video recording can produce in many respects more accurate accounts of what actually occurred than participants' and/or observers' often flawed recollections. 'By approximating direct observation, video provides a shared resource to overcome gaps between what people say they do and what they, in fact, do. ... Thus we would argue that videotaping, the mechanical audiovisual fixation of an event, produces data much closer to the event itself than other kinds of representation' [Jordan and Henderson, 1995, pp.51].

A video recording provides a permanent record of the events being studied, allowing the analyst to return time and again to build and to evaluate the analysis. 'It is in the course of repeated viewing that previously invisible phenomena become apparent and increasingly deeper orders of regularity in actors' behaviors reveal themselves' [Jordan and Henderson, 1995, p.52].

In any situation with more than one participant engaged in simultaneous behaviour, it is impossible for an observer to record the behaviours in any detail using simple means such as pen and paper. Video, on the other hand, effortlessly captures a much richer stream of data.

The use of video as data is, however, not without its problems. First, despite the illusion of being a faithful and complete record, a videotape is simply a representation of certain aspects of the recorded situation. Those aspects which are captured are constrained both by what the camera was focused on and by what the medium is capable of recording. A video tape records events from a particular camera viewpoint, omitting what is out of shot. In addition, no matter how much of the immediate situation is recorded, prior events or events outside the immediate situation are not recorded regardless of how they impinge on the recorded events. On the other hand, all that the video camera picks up and makes available to the analyst may not necessarily have been picked up by the participants in the events and the analyst will inevitably choose to focus on only some of the recorded events.

In addition to the constraints on video as a representational medium, there are considerable difficulties inherent in performing the work of analysing a video record. Sanderson and Fisher [1994, p.253] note that 'with modern recording technology, observational data are much easier to collect than they are to analyze, and analysis is often prohibitively time-consuming. Even given the time, we are less and less sure what type of data we should be accessing and what form our analysis should take'. While these problems with analysing video records remain, Sanderson and Fisher do describe a generic process and techniques for addressing them (see Figure 3.17).

In providing a review of what they call 'Electronic Sequential Data Analysis' (ESDA) methods, Sanderson and Fisher [1994, p.255] define ESDA as 'any empirical undertaking seeking to analyze systems, environmental, and/or behavioural data (usually recorded) in which the sequential integrity of events has been preserved. The analysis of such data (a) represents a quest for their meaning in relation to some research or design question, (b) is guided methodologically by one or more traditions of practice, and (c) is approached (at least at the outset) in an exploratory mode'. The researcher is typically guided by the formal concepts of a particular research tradition. These include assumptions about what are relevant research questions, how such questions are operationalised, appropriate analytic techniques and what kinds of statements are acceptable as valid and well substantiated answers to the research questions.

The formal concepts underlying an analytic approach first influence the research or design questions which the researcher poses. (The questions may be 'design questions' because the analysis may be applied, for example, to video based software usability evaluation for design purposes. However, this work primarily is interested in answering research questions and subsequent references here shall be to research questions.) The formal concepts also guide the choice of what kind of raw sequences of data should be observed in order to address the research questions. 'Raw sequences are the fleeting system, environmental, and behavioural events that are observed. Logs and recordings are the aspects of these events that are captured either as computer data logs or on videotape or audiotape' [Sanderson and Fisher, 1994, p.256].

It is worth noting that the relationships between the 'formal concepts' of a research tradition and the artefacts of the research work may be more complex than is suggested by Sanderson and Fisher [1994]. Thus, whereas the model of the analysis process presented in [Sanderson and Fisher, 1994] describes the formal concepts of a research tradition influencing choices of research questions to be answered, suitable data to observe and operations to be performed on the data, there are reciprocal relations whereby research questions or constraints on the data which may be collected influence the choice of appropriate intellectual traditions and formal concepts. Similarly, the transformed products and statements which arise from an analysis may suggest an intellectual tradition which is appropriate for validating and testing them.

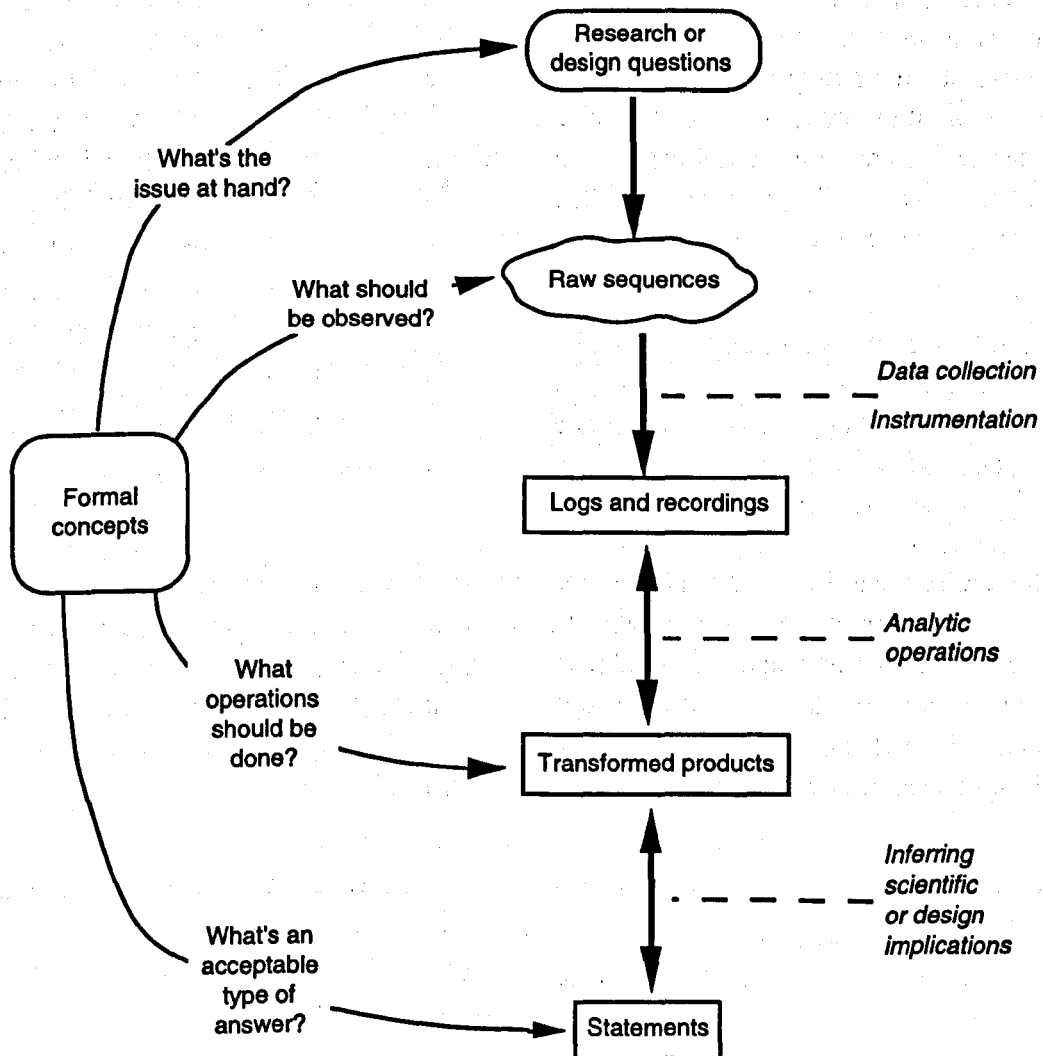
Transformed products are the products of the researcher's work on the logs and recordings. These products may include, for example, transcripts of some or all of a video recording, collection tapes of instances of a particular piece of repeated behaviour or timelines for particular activities in the logs or recordings. Again, the formal concepts have an influence: 'How the transformations are performed and what the products are like will be determined by the ESDA technique used' [Sanderson and Fisher, 1994, p.256]. In turn these transformed products are used to support statements which attempt to answer the original research question. Sanderson and Fisher [1994] stress that the analysis is absolutely not a linear process. 'ESDA involves exploration, feedback, and iteration. One is often reviewing and changing "more transformed" ESDA products in light of "less transformed" products. ... Such exploratory activities are an important part of ESDA' [p.256].

Sanderson and Fisher [1994] distinguish three research traditions which commonly underlie approaches to analysing video data: the behavioural, cognitive and social traditions (see Figure 3.18). The research tradition adopted has a strong influence on all aspects of an analysis.

In the behavioural tradition there is a formal commitment to the classical scientific method, relying strongly on sampling and measurement theory and development and standardisation of encoding schemes in order to achieve objective and replicable

results. 'Encodings are generally syntactically simple reexpressions and summarizations of raw sequences, filtered through a theory or viewpoint' [Sanderson and Fisher, 1994, p.271]. Generally, statistical techniques are then applied to the coded data. Olson, Herbsleb and Reuter [1994], for example, describe the application of two statistical techniques to observational data: log-linear modelling and lag sequential analysis. The results of statistical analysis are then interpreted to produce answers to the original research question. Statements attempting to answer the original research question are typically couched in terms of statistically significant comparisons of conditions on a numerical scale. 'If firm criteria exist to judge the desirability of [the dependent variable values] in these cases, then such statements serve design in a relatively direct way. They serve theory construction in a less direct way by providing evidence for or against theoretical predictions' [Sanderson and Fisher, 1994, p.278].

Figure 3.17: Generic ESDA process [Sanderson and Fisher, 1994]



The cognitive tradition shares with the behavioural tradition an emphasis on hypothesis testing, precision and objectivity. However, the cognitive tradition typically includes an assumption that 'an individual's verbalizations ... can serve as evidence for underlying cognitive processes' [Sanderson and Fisher, 1994, p.280]. Like the behavioural tradition, the cognitive tradition often relies on encoding a subject's activities and verbalisations. The cognitive tradition's goal of explaining underlying cognitive processes tends to emphasise capturing and encoding verbalisations (e.g. [Ericsson and Simon, 1993]) and using them to infer or to test cognitive processes. In analysing the encoded data, researchers in the cognitive tradition often try to produce a model of the underlying cognitive processes whose structure the researchers claim is reflected in the encoded activities and verbalisations. A well known example of such a model in HCI is presented by Card, Moran and Newell [1983].

The social research tradition is particularly concerned to describe, to understand and to account for group activities and behaviour. Thus, approaches within this tradition attempt to 'generate accounts of how individuals and groups use available resources to reach goals and make sense to one another' [Sanderson and Fisher, 1994, p.288]. Within the social tradition, analysis typically is qualitative rather than quantitative, relying on the comparison and interpretation of interactional events rather than on coding and statistical testing. Analysts within the social tradition do sometimes use coding, but with a different purpose and emphasis from those in the behavioural and cognitive traditions. Within the social tradition, 'codes are closer to final statements than to encoded data. When codes "encode" raw data, statistical analyses or other further operations have to be performed before the meaning of the data is revealed' [Sanderson and Fisher, 1994, p.289]. Thus, to ignore for a moment the highly iterative nature of most analysis work, in the behavioural and cognitive traditions interpretation tends to follow coding, while in the social tradition coding tends to follow interpretation.

Researchers within the social tradition acknowledge that there may well be several plausible interpretations of the data. While researchers within the other traditions might concede this point, statistical methods generally are held to promote a single, objective interpretation. Within the social tradition, 'interpretational bias is acknowledged and explored, rather than eliminated, because it is held that no neutral point of view can exist' [Sanderson and Fisher, 1994, p.289].

Figure 3.18: Three intellectual traditions in ESDA [Sanderson and Fisher, 1994]

ESDA Question	ESDA Dimension	Tradition		
		Behavioural	Cognitive	Social
What's the issue at hand?	Investigative approach	Use scientific method, often hypothetico-deductive tests, and achieve objective results	Model cognitive processes within the individual over time	Provide accounts of social phenomena; empirical, ethnographic, strong in inductive methods
	Setting	Field settings sought to ensure ecological validity; laboratory sometimes used	Systematic laboratory investigations, but also applied field settings	Field observations, emphasise value of observer's participation
	Sampling	Sampling theory and measurement theory used for subject and code selection	Ideally as for behavioural; detail needed often constrains sampling adequacy	Cover representative situations, pursue themes, artefacts or agents; seldom statistical sampling
What should be observed?	Focus of analysis	Objectively identified behavioural observables	Individuals' verbalisation and action as evidence for cognitive processes and structures	Social interactions, communicative devices; utterance, gesture, action
	Coding and description	Formal encoding using standardised terms; concern with reliability and objectivity	As for behavioural; debate whether use of context constitutes bias; induction often needed	Interpretation parallels encoding; participant involvement; use of multiple interpretations
	Means of analysis	Sequential, nonsequential statistical methods	Intensive assessment of goodness of model fit for a few subjects	Emphasis on qualitative rather than quantitative analysis; ethnographic
What's an acceptable type of answer?	Sources of rigour	Quantitative; concern with replicability and generality of results	Relation of account to extant models of cognition, computational adequacy	Qualitative; plausibility and robustness of account; finding one sound interpretation from many

Interaction analysis is an approach within the social tradition which 'has recently been used within the HCI community to study interactions among people in work environments and among people and engineered systems' [Sanderson and Fisher, 1994, p.290]. Whilst acknowledging that the field of interaction analysis is still in the process of defining itself, Jordan and Henderson [1995, p.41] state that the goal of interaction analysis is 'to identify regularities in the ways in which participants utilise the resources of the complex social and material world of actors and objects within which they operate'. They argue that verifiable observation provides the best foundation for analytic knowledge of the world, implying a commitment to inductively grounding theories in empirical evidence.

Interaction analysis is very firmly based in the use of video recording as the means of capturing data. An early stage in interaction analysis frequently is the production of a content log from the video record of the raw sequences. '[Content logs] consist of a heading that gives identifying information followed by a very rough summary listing of events as they occur on the tape. The level of detail is determined by the interests of the researcher and the available time' [Jordan and Henderson, 1995, p.43]. Content logs provide an overview of the recorded data. They may be used to aid early identification of themes and events, to relate comments to particular interaction sequences, to note sections of tape which should be transcribed fully for more detailed analysis.

As themes are identified and pursued, sequences of interaction which illuminate those themes are analysed in detail and compared. Collection tapes may be produced which bring together examples of an interesting interactional sequence from the video record. Such examples may also be transcribed in order both to facilitate detailed analysis without the need for constant use of video equipment and to provide illustrative examples within written work.

Analytic themes or 'significant features' of the interaction are identified through 'searching for specific distinguishing practices within a particular domain or for identifiable regularities in the interactions observed' [Jordan and Henderson, 1995, p.44]. The analyst inductively derives explanations for the observed interaction. Analysis then returns to the video record to test the robustness of the explanation across instances of an interaction sequence which it purports to explain (cf. Figure 3.14 above). 'At any one point when promising hypotheses have been formulated, it is incumbent on the researcher to assess which observations are indicative of general patterns, which are idiosyncratic or random perturbances, and which are due to some as yet unexplained (or unexplainable) cause. This is done by finding other instances of the event in question in the data corpus and checking whether the proposed generalization holds' [Jordan and Henderson, 1995, p.46].

Explanations, therefore, remain firmly grounded in the evidence of the video data. Ultimately, the statements which are presented to address the original research question derive from the induction and validation of these explanations. However,

this does not mean that the interaction analyst approaches a video tape with a purely inductive attitude and no initial ideas or hypothesis in mind. 'It goes without saying that the very process of looking is informed by some notions of what one is interested in looking for, notions that, in turn, are modified by what it is that one finds as one gets deeper into the analysis' [Jordan and Henderson, 1995, p.46].

Tang's [1989, 1991] work is an example of the application of interaction analysis, producing 'a descriptive study of the shared workspace activity of small groups working on conceptual design tasks' [Tang, 1989, p.iii]. Tang [1989, p.5] notes that 'the issues investigated ... were inductively discovered by examining the empirical data, rather than determined a priori for experimental verification'.

3.4.2 Approaches to the research questions

Having adopted an observational, rather than experimental, approach, an array of techniques and methods was available from the research traditions described above. It was necessary, therefore, to choose and to apply a particular method or combination of methods. Various pragmatic and theoretical criteria mitigated for or against potential techniques and approaches. The need has been noted above to draw on more than one research tradition and in analysing the cooperative development work, this research drew on both social and cognitive traditions.

From the data available, it was required, in answering the research questions, to develop an account of the processes and activities in which users and developers were jointly involved, how the participants, especially the users, contributed to those processes and activities and what was made of these contributions. Systems development, as a form of collaborative work, is primarily a social activity [Bucciarelli, 1988], suggesting a social approach to its analysis.

The intimate involvement of the analyst in the development projects demanded an approach to analysis which acknowledged and coped with such involvement. This also suggested the adoption of a social approach in which 'there is room for greater involvement of the participants themselves in the interpretation and analysis of the data' [Sanderson and Fisher, 1994, p.289].

Several factors mitigated against adopting *ab initio* a behavioural or cognitive approach to the analysis. First, defining codes and performing statistical manipulations of them demand *a priori* a theoretical model of the phenomena of interest. This work attempted in part to derive a theoretical account of user-developer interaction in task driven cooperative software development. There was, therefore, no such *a priori* theory through which to filter the raw data. Also, encodings tend overly to simplify the raw data and thereby to lose richness and contextualised meaning. Minneman [1991, p.127] argues that 'in addition to the problems in operationalizing such a counting operation, we know little of what such a thing would mean, or how important particular interactions were in the overall accomplishment of the design effort'. This is particularly pointed in the absence of a theory of the object of analysis.

The assumptions and formal concepts which influence the posing of particular research questions also influence choices of which analytic operations should be performed. Thus, a basic assumption of this work that every additional layer of interpretation and mediation introduces potential for misunderstanding mitigates against a statistical approach. Encoded data, in the behavioural and cognitive traditions, does not *per se* speak to the research question and neither yet do the results of the statistical analyses on the data. These results themselves must be interpreted. What remains after filtering through a theory, encoding and applying statistics is far removed from the interaction data recorded on videotape, still further from the recorded events. In exploratory research of social interaction for which there is little existing theory, interpreting such statistical residues must be even more tentative and potentially flawed than interpreting the data itself.

The nature and range of the research questions demanded different but related approaches to answering each. This exemplifies the growing need for work to draw on more than one research tradition as the interdisciplinary scope of HCI continues to expand. Clark [1996], for example, argues that researchers must increasingly often draw on more than one 'pure' research tradition. Sanderson and Fisher [1994, p.254] note that an 'implication for HCI research is that it is desirable and often necessary to view a research question from multiple perspectives. ... Researchers need to go beyond their own disciplinary biases or training, expanding the range of questions they are prepared to tackle and broadening the research techniques they are prepared to adopt'.

As noted above, the approaches adopted in tackling the research questions were diverse but all were firmly based on analysing the interaction between user and developer in the cooperative development settings. Thus, the analytic approach to answering research question adopted an exploratory view of user-developer interaction in the cooperative development setting and began from an interaction analysis stance within the social tradition. As this analysis moved towards a theoretical account of user-developer interaction in cooperative development, approaches to each subsequent research question could become more focused and more quantitative. The approaches adopted in tackling each research question are described and exemplified in detail in the following chapters. Here is presented a very brief summary of the data used and approach adopted for each research question.

RQ1. *What is user participation in software development: what are the processes and activities in which users and developers are involved and how do the participants, especially users, contribute to those processes and activities?*

RQ1 was addressed through an analysis of user-developer interaction in cooperative development meetings across the projects. This analysis was based primarily on the video records of development meetings, supplemented by insights gained from the analyst's presence at the meetings as a participant-observer and by other artefacts described in section 3.3 above. Minneman's [1991] framework of group design

interaction was used to focus the interaction analysis of the video records on three aspects of the cooperative development activities: artefacts, processes and relations (see chapter four).

As the analysis cycle began to produce a coherent account of the nature of user-developer interaction in the cooperative development setting, the work of Clark (e.g. [Clark, 1996]) was used to develop the emerging account of shared understandings jointly constructed by user and developer. This analysis, reported in chapter four, examined user-developer activity in producing internal and external representations, laying the groundwork for a theoretical account of user-developer cooperation in systems development as the joint construction of common ground.

RQ2. *Were the projects studied participatory, i.e. did users actively contribute to the interaction, rather than being passively present or simply a source of on-line data to be tapped by developers?*

The second research question asked whether the development projects studied actually were participatory. Again, RQ2 was addressed primarily through analysis of the video record of cooperative development meetings, extending the work on common ground from chapter four. Having offered a definition of participation in terms of actively contributing to the development discourse, chapter five built on the analysis of chapter four to extend Clark and Schaefer's [1989] notion of how contributions are made and to predict what contributions may be made in the cooperative development setting.

Having operationalised the notion of a contribution to development activity, the analysis then examined samples of the video records from three cooperative development meetings, covering a range of development activities, and assessed user participation through the types and numbers of contributions made.

RQ3. *Was user participation effective in the studied projects, i.e. were user contributions assimilated into the external and internal artefacts of the development process?*

The third research question built on the second in asking how effective user participation was in the system development projects. Having assessed user participation in terms of contributions to the development discourse, the work reported in chapter five went on to operationalise the *effectiveness* of the contributions in terms of their assimilation into the artefacts, both external and internal, of the development process. This assimilation was assessed for the contributions across the sampled video records.

RQ4. *How valuable was user participation in the studied projects, i.e. what relations were there between user contributions to development activities and features of the software product's usability?*

The fourth research question asked how valuable user contributions were to the development process. Chapter six offered a definition of the value of a contribution in terms of its influence on the implemented software.

In answering RQ4, the work reported in chapter six first involved running usability evaluations of the software developed in each project. Good and bad usability features identified in these evaluations provided a prescribed set from which to trace upstream through the development activities. The analyses of chapter five also provided a prescribed set of contributions to development activities which, in chapter six, were traced downstream to features of the software. These complementary tracing analyses provided an assessment of the influence of user contributions to development activities on the resulting software.

3.4.3 Issues in performing video based analysis

Section 3.4.1 noted the essential contribution of video recording to the research and some of the problems associated with the use of video. The difficulties are of two main types: acquiring the video record in the first place and then making sense of what has been recorded.

The first problem in acquiring a video record was in gaining access to software development projects whose participants were willing to be studied at all. Having negotiated access to the two projects described in section 3.2, the researcher had to persuade the participants to do their work in front of the camera. This was not always successful. Some participants flatly refused to be recorded under any circumstances. Others were willing to have only some meetings recorded, often for reasons of personal, political or commercial sensitivity.

The researcher had access to only one camera which seriously restricted what could be recorded. In addition, the involvement of the researcher in the recorded activities, combined in some cases with the participants' reluctance to have anyone else present, entailed the camera's being left in a fixed position and with a fixed field of focus. Given these limitations and the researcher's primary interest in the interaction of the participants, almost all of the video record was taken as 'long shots'.

Having produced a video record of at least some of the projects' development activities, the next issues raised were in analysing what had been recorded. A lack of automated video analysis tools suggested the initial transcription of the video records. It seemed at first that analysis from transcripts should be easier than from tapes. This proved to be wrong on two counts. First, it was impossible to find a fast and accurate transcriber. Two professional transcription services and an audio typist took several weeks to produce inaccurate transcripts of a few hours tape before this path was abandoned. Jordan and Henderson [1995] and Sanderson and Fisher [1994] both comment on the large amount of time required to transcribe even a small section of video tape. Transcription is, therefore, often restricted by

interaction analysts to 'those features of the interaction that emerge as significant in the course of tape analysis' [Jordan and Henderson, 1995, p.49].

In addition to the difficulties of entrusting transcription to an independent party, the process of transcription itself helps the analyst to become immersed in and familiar with the data. Minneman [1991, p.72] argues that the activities of being present and observing the development session, becoming familiar with the data through logging and organization, and transcribing the audio record all contribute to analysis. Similarly, Jordan and Henderson, [1995, p.50] argue that interaction analysis 'cannot easily be delegated to assistants because a deep understanding of the phenomenon of interest requires proceeding through successive approximations until the relevant analytic categories are identified'. This reinforces the inappropriateness of attempting to analyse a video record in isolation from the context in which it was recorded but also points up the difficulty of repeating the analysis for those not involved in all the activities.

A second problem with attempting to precede analysis with transcription is that the strength of a video record is in the richness of what it captures as data. This advantage over other media is lost when the data is transformed once again, for example by transcription. Various notations have been used for transcription. Jeffersonian notation (see [Atkinson and Heritage, 1984]) is widely held to be the most comprehensive notation for verbal transcription. Many researchers (e.g. [Bowers, Pycock and O'Brien, 1996; Suchman and Trigg, 1991]) claim to use modified versions of Jeffersonian notation, where the modifications often seem to consist of eliminating much of the detail of that notation. But there is as yet no established notation for transcription of nonverbal interaction, although there are non-verbal schemes, e.g. Laban for choreography. The analyst (and her audience) must be aware that any transcription loses aspects of the recorded data. 'There is no ideal or complete transcript according to any abstract standard. Rather, the question must be: how adequate is this transcript for purposes of the analysis to be performed' [Jordan and Henderson, 1995, p.48]. It seems that video based interaction analysis must remain video based, with the expense of transcription incurred for detailed analysis of short sequences and for exemplification in reporting the results of the analysis.

While the initial analysis in tackling research question one took a broad sweep and looked for patterns in the video records of whole meetings, later analysis became more focused on shorter sections of the video record which were transcribed. The analysis of these sections, for example in the samples used in tackling research question two, utilised the video record and transcript, often simultaneously. This movement from broad analysis of the video records to detailed analysis of shorter sections raised issues of sampling.

In the research reported here, some *de facto* sampling had of course already occurred in recording only some aspects of the development projects. It is simply not possible, nor is it generally necessary, to have a complete video record of every

moment in a software development project. Getting a video record of only parts of a project is in the nature of this type of research and of real projects with participants going away, access refused and other problems. Similar problems face the developers themselves in performing their work.

In turn, sampling of the video record is necessary in general simply to reduce the sheer volume of data available from a video record. Even a small amount of video data can demand an enormous amount of analytic effort. Sanderson and Fisher [1994] note analysis time to sequence time ratios (AT:ST) ranging from 5:1 for simple analysis to 5000:1. It is not unreasonable to estimate AT:ST ratios for the work reported in this thesis as several thousand to one.

Mason [1996, p.83] notes that 'the term sampling is very often associated solely with a logic derived from general laws of statistics and probability' and argues that in qualitative research other logics often must be applied. She suggests that at the heart of the logic of sampling is the question: what relationship do I want to establish, or do I assume exists, between the sample or selection I am making and a wider population or universe? Mason [1996] describes four examples of such relationships. The first is when the sample is considered to be *representative* of a wider population in that it displays characteristics in similar proportions and patterns to the total population. Statistical operations are used to calculate the probability that patterns observed in the sample will exist in the wider population. As with the 'hard' techniques described in section 3.4, this approach relies upon the analyst having *a priori* knowledge (or, at least, theory), in this case of the parameters of the total population.

Several problems arise with this approach to sampling in qualitative, exploratory research. First, the parameters of the total population may not be known and may not be measurable. Secondly, the statistical approach often demands very large samples and, as noted above, video based analysis is extremely costly and time-consuming and therefore cannot readily be applied to large samples. Thirdly, and perhaps most fundamentally, exploratory, qualitative research 'uses a different analytical logic and one which is not particularly well supported by the generation of a representative sample' [Mason, 1996, p.91].

The other three examples of relationships between sample and wider universe described by Mason [1996] are (i) a relationship between sample and the wider universe which is *ad hoc* or unspecifiable in any way; (ii) a relationship where the sample is intended to provide a close up, detailed view of particular features; and (iii) a relationship where the sample is designed to encapsulate a relevant range of features in relation to the wider universe.

Much of the sampling in reported applications of video analysis is driven by the last two relationships. Chin, Rosson and Carroll [1997, p.165], for example, identified 'topics and themes that were important to teachers and students, and then tried to find occurrences of these themes in the classroom videotapes ... and selected the

relevant episodes'. The criteria for identifying sequences may be quite vague: 'There were also episodes that we simply found interesting and worthy of analysis' [Chin, Rosson and Carroll, 1997, p.165]. Similarly, Bødker [1996, pp.157-8] notes that 'an analysis of four hours of videotape is a complex matter. ... We then viewed the videotape and selected interesting sequences for closer inspection'.

The sampling practice in this thesis became more refined as the analysis progressed. Following initial, exploratory analysis which maintained a broad, research focus across the recorded material (chapter four), later analysis (chapters five and six) sampled across meetings to include examples of different types of development activities: work situation analysis, requirements analysis, work design and software system design. Here, the last relationship held between sample and wider universe, with the relevant range of features being these identified types of development activity. The analysis attempted to assess the participants' contributions to these activities in the samples.

The first, *ad hoc* relationship is undesirable, severely limiting the analytic potential of a study. The latter two relationships are often associated with theoretical or purposive sampling which selects samples which enable the analyst to make comparisons and to develop theory. This strategy contrasts with statistical or probability sampling which is used to generate empirically representative samples. Theoretical or purposive sampling 'links sampling very directly into the process of generating theory and explanation inductively from or through data' [Mason, 1996, p.93].

Mason [1996] proposes several further questions which the analyst should ask of a sampling strategy. For example, what is the wider universe or population to be sampled? What is the analytical interest in this universe or population? What is to be compared? What would 'negative instances' look like?

In analysing the development groups in the studied projects, this research assumed that features of the interaction within these groups should reflect similar features in other small development groups involving users and developers in task based development and, less reliably, in cooperative development generally. In choosing relatively short samples of the video records for detailed analysis, chapter five assumed that they should reflect the wider universe of the development meetings from which they were taken.

The analytic interests included the moment to moment interactions of user and developer and the longer patterns which these interactions formed. The broadly focused interaction analysis of chapter four pursued the interest in the longer patterns while the video samples of chapter five pursued the interest in analysing the moment to moment interactions.

In the samples of chapter five, the analysis compared contributions by users and developers. Chapter four identified four types of development activity and six types

of contribution. The work of Kensing and Munk-Madsen [1993] suggested a basis for predicting the relative contributions of user and developer across these categories. The sampling reported in chapter five then supported comparisons of users' and developers' actual and expected contributions and of actual contribution to types of development activity compared to the declared purpose of the analysed meeting.

In addressing research question two, chapter five took user contributions as a measure of active participation and so a comparison of user and developer contributions with their predicted levels allowed an assessment of the level of participation in the sampled development activities. Since research question two asked if the development work was participatory, 'negative instances' here were reflected in low levels of user contributions. The focus of the second half of chapter five, in addressing research question three, was the effectiveness of contributions. This was defined as the proportion of contributions noted in the samples which were assimilated into development artefacts. Hence, there was a fixed baseline in the body of noted contributions. This also provided a fixed baseline for tracing downstream to software usability features in the chapter six analysis of the value of contributions.

3.4.4 On researcher involvement in the projects

Across the studied projects, the analyst was also involved as participant and champion of cooperative development. The implications of this involvement may be considered both in terms of the practical development work done and the research reported in the following chapters.

Bødker [1996] emphasises that when a PD champion and researcher withdraws from a project, a successful withdrawal is hardly visible, with the participant-researcher not being missed. Carroll [1996] notes that this is a far less interventionist conception of the PD champion's role than in earlier PD work.

In the CS project reported here, the author had a strong hand in involving the participants in the task based cooperative development approach. When the author withdrew from the CS project, the pace of development work slowed. This may, however, have been more due to the almost simultaneous departure of the other developer at the end of his contract. The project stalled for several weeks until another developer was appointed to complete the implementation.

More cogently, contact between those immediately involved in the development work and other users throughout the department evaporated almost completely after the author's withdrawal. For example, when the system went live there was just a brief email around the department saying that it was available. Common complaints picked up from users when the author returned to conduct usability evaluations included a lack of information on the facilities of the new system, a lack of training on its use and the absence of an identified person to whom to pass comments, requests and complaints.

In the CI project, contact with users if anything increased - through beta testing and evaluation and eventually a marketing drive - after the author's leaving the project. As throughout this project, the work was primarily driven by the demands of a commercially successful product. There is evidence, however, that the author's presence had a direct impact on development practice during the project. The chief developer referred to the author on several occasions as his 'conscience' for the latter's frequent enjoinders not to move ahead with development activities without taking on board user contributions.

The author, as champion or facilitator of cooperative development, did not determine the nature or extent of the cooperation - indeed, one person cannot possibly 'cooperate' - but did play a crucial role in setting up the conditions in which cooperation could occur (and was encouraged). One point of the research was to assess what then did occur in that setting amongst all of the participants. There is a question of objectivity in the researcher's both encouraging user participation and attempting to assess the extent of such participation. However, the assessment of the extent of participation, reported in chapter five, relied on counting the number of contributions made according to defined categories, thus reducing the risk of confirmation bias. This risk cannot, however, be wholly discounted. Future work by other researchers and with other methods should include testing the findings of this analysis. Although, as Greenwald, Pratkanis, Leippe and Baumgardner [1986] have argued, confirmation bias is a major problem in the development of theory even through controlled experimental methods.

Following a cooperative development approach, as described in section 3.1, helped to create the setting for user-developer interaction and the author needed to be present to monitor and, when necessary, to encourage the use of the cooperative development approach. A further pragmatic reason for the author's involvement was the absence of anyone else to perform the logistical tasks in gathering data on the development work. Also, since video recording could capture only a small proportion of the overall development activities, any additional access to those activities by the researcher was useful.

The author's presence as part of the development group made him familiar to the other participants. This in turn made recording and analysing their activities less intrusive. However, such intrusion can never completely be eliminated. It has long been noted that the mere knowledge amongst workers that research is going on can influence the work practices which are the subject of the research [Mayo, 1948]. Many approaches within the social and action research traditions acknowledge and actively value the researcher's involvement in the situations studied. Soft Systems Methodology (SSM) [Checkland, 1981; Checkland and Scholes, 1990], for example, insists that analysis of the situation of concern must include the analyst's own role.

The action research philosophy underlying both SSM and PD can be useful to both researcher and researched. In addition to providing the potential for immediate

useful application to the organisation cooperating with the research, the theory which develops from action research is closely associated with the practice which it seeks both to explain and to inform. Thus, theory and models can be built upon practice and quickly tested in continuing practice. Theory and practice evolve together with the practice aiding collaborators along the way. The action research philosophy contrasts with the 'hard' [Checkland, 1981] or 'rationalistic' [Bødker, Grønbaek and Kyng, 1993] software engineering approach. In the latter, theory and practice often have little in common.

The limitations (see above) of video recording also encourage the analyst's involvement in the studied situation. Nardi [1996b, p.98] criticises Jordan and Henderson's [1995] approach to interaction analysis for using participant-observation simply to identify interactions which are then analysed from the video record. Notwithstanding this criticism, Jordan and Henderson [1995] do recognise and endorse the value of analyst participation as an important complement to studying the video record. They state that information gathered from participant observation and the analysis of artefacts and documents contributes to 'the background against which video analysis is carried out, and the detailed understanding provided by the microanalysis of interaction [on the video], in turn, informs our general ethnographic understanding' [Jordan and Henderson, 1995, p.43].

Similarly, in this research, the author's close involvement in the development activities and his examination of the video records of those activities supported each other in producing the analysis which is reported in the following chapters. Chapter four begins by introducing this analysis and developing the first major strand of it.

Chapter 4

Towards a theory of user-developer cooperation

The work reported in this chapter addressed research question one: what *is* cooperative software development; what are the processes and activities in which users and developers are involved and how do participants, especially users, contribute to those processes and activities? Research question one was addressed through an analysis of user-developer interaction in development meetings across the projects. This analysis was based primarily on the video records of development meetings, supplemented by insights gained from the analyst's presence at the meetings as a participant-observer and by analysis of other artefacts, described in section 3.3 of chapter three.

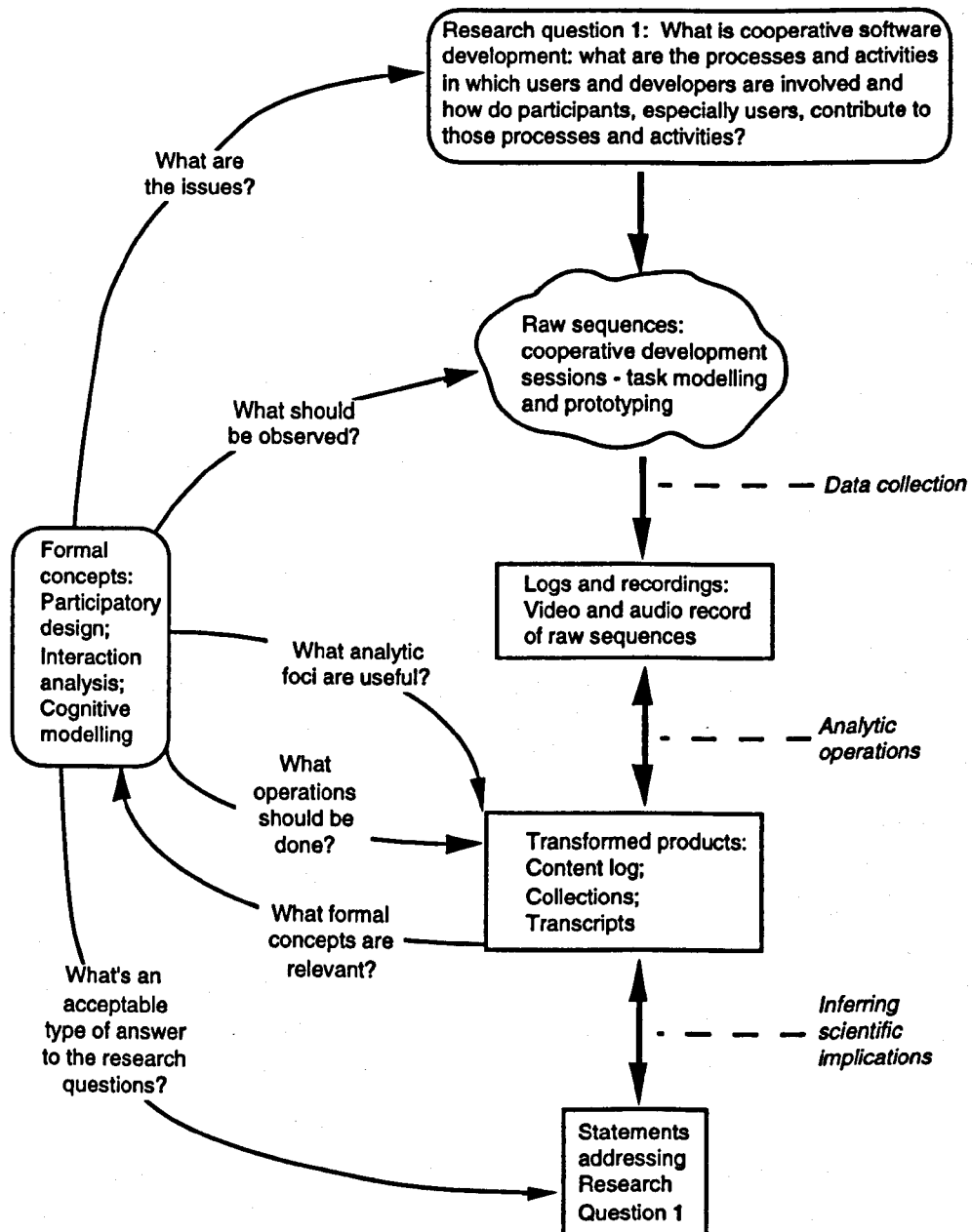
The work began with an exploratory analysis of the interaction between developer and user in cooperative development settings. Early analysis was guided by Minneman's [1991] framework of group design interactions and by the 'analytic foci' of Jordan and Henderson [1995]. Section 4.1 describes the process and results of this interaction analysis. The findings are presented in terms of the three facets of Minneman's framework: artefacts, processes and relations.

A major result of the research reported in section 4.1 was an analysis of user-developer interaction as the coconstruction of shared knowledge and understandings. The predominant pattern of interaction between user and developer was the instantiation and verification of shared understandings. Other disciplines, such as philosophy and linguistics, have a history of theoretical concern with the concept of shared understanding or 'common ground'. Hence, section 4.2 presents a summary of influential theoretical treatments of this concept.

The next phase of this research brought the theoretical concepts of mutual knowledge and common ground to bear on the continuing analysis of user-developer interaction. The results of this work, reported in section 4.3, extend both the analysis of user-developer interaction and the theoretical concept of common ground in the cooperative development setting.

Figure 4.1 illustrates an instantiation for this research of Sanderson and Fisher's [1994] generic model of the analysis process (cf. Figure 3.17 of chapter three). The research question (RQ1) was inspired by the 'formal concepts' of participatory design. As described in chapter three, the formal concepts and the research question determined the need to have real world development meetings as the raw sequences. Data collection resulted in logs and recordings consisting of video and audio records and notes of development meetings.

Figure 4.1: Extended instantiation of ESDA process for research question one



It is important to note, however, that in this figure, in addition to the logs and recordings, the data collected included artefacts of the development process which

were also available to the analytic operations. Also, note that in Figure 4.1 influences are indicated in both directions between formal concepts and the ongoing analysis (cf. Figure 3.17). For example, in an applied setting, questions may arise which are outside the scope of the initial formal concepts.

The interaction analysis began from the video recordings of development meetings. The formal concepts of interaction analysis and the developing analytic account itself influenced choices of operations and analytic foci used to produce transformed products such as transcripts. The transformed products in turn supported the generation of a theoretical account which addressed the original research question. The ongoing analysis itself suggested formal concepts which might be relevant. Specifically, from the analysis of the video record a theoretical account of the interaction was developed which sat comfortably in the cognitive tradition.

The analysis reported here used Minneman's [1991] framework of group design interactions as an initial guiding resource. Minneman argued that communicative acts are the primary means by which designers go about designing. He developed a framework (Figure 4.2) which characterised group design communications in terms of 'facets' (topical orientations of particular sequences of interaction) and 'trajectories' (communicative responsibilities incumbent upon the interactants). The facets of group design communications identified by Minneman [1991] are artefacts, processes and relations. 'The first facet of the framework captures those interactions that centrally concern or accomplish work on any artefact with which the group or individual is concerned. ... The second facet ... deals with communications about the processes that are being employed. ... The final facet addresses the manifold relations extant in the design setting. Included here are all of the permutations of individuals to groups, the relations between rules and activities, or the relations between performance and reward' [Minneman, 1991, p.114]. Trajectories, or communicative responsibilities, include (i) informing about the current state of artefacts, processes and relations, (ii) making sense of their history and (iii) framing their future development.

Figure 4.2: Minneman's [1991] framework of group design interactions

<i>Trajectories</i>	<i>Facets</i>		
	artefacts	processes	relations
current state of			
making sense of			
framing futures of			

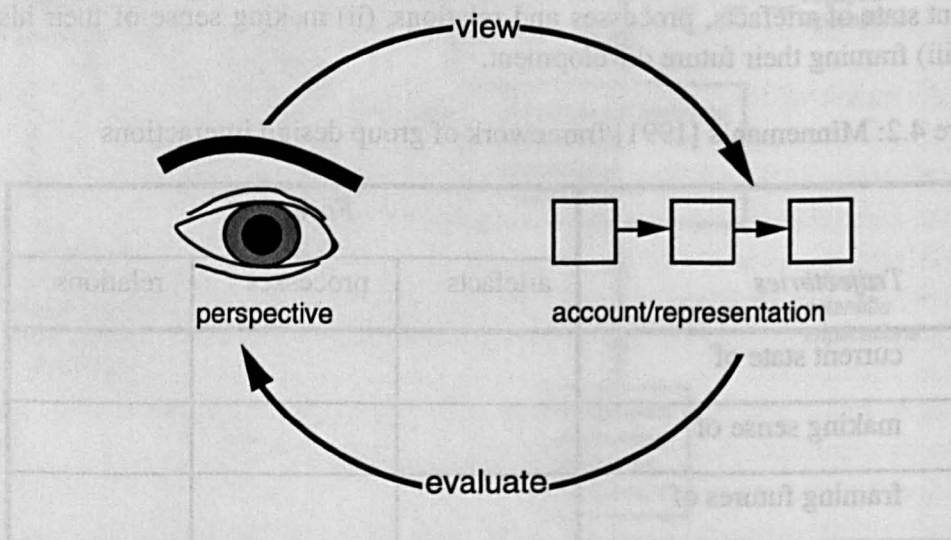
The analytical use made of the framework was in its identification of artefacts, processes and relations as the main topical orientations within group design

interactions. This research used these facets as primary analytic foci in analysing the interactions of users and developers.

Minneman [1991, p.14] described his analysis method as a cyclic process involving the production of perspectives (a particular sensitivity during analysis) and accounts (detailed explanations of the analysed activity). In any cycle, there is a current analytic perspective and the activity is examined to produce an account consistent with that perspective. The account itself is then scrutinised, checking its fit with the video data and comparing it with findings from other disciplines. This process produces issues from which a new perspective evolves and the analysis cycle is reiterated. This cyclic process is illustrated in Figure 4.3.

The representations in Figure 4.3 broadly equate to Sanderson and Fisher's [1994] 'transformed products'. As analysis moves around the cycle, the analyst's perspective develops and the representations and accounts become more comprehensive and more sophisticated. 'At the outset, the understanding of what transpires ... is very rudimentary. ... Later we may begin to see additional order manifest in the interaction' [Minneman, 1991, p.103]. Thus, as the analysis cycle in this research began to produce a coherent account of the nature of user-developer interaction in the cooperative development setting, existing theory on shared understanding and common ground was used to develop a more strongly theoretical account of shared understandings jointly constructed by user and developer. Later analysis cycles examined user-developer activity in producing internal and external representations, laying the groundwork for a theory of user-developer cooperation in systems development as the joint construction of common ground.

Figure 4.3: Schematic representation of the cyclic interaction analysis method [Minneman, 1991]



Minneman [1991] described the goal of his work as 'essentially a descriptive account' [p.4]. The work reported here attempted to go further in producing an account which was not only descriptive, but explanatory.

Analyses of purposeful human interaction may, in taking account of the social aspects of interaction, sometimes focus too strongly on these social aspects *per se* and lose sight of the fact that the interaction *has a purpose*. This despite almost always explicitly acknowledging that the purpose is important. For example, Minneman [1991, p.10] states that his research 'focuses, not on the number and quality of the final results of a design activity, but rather on the processes that the participants engage in during the design activity. This focus has the benefits of not requiring an assessment of the quality of the design result and permits a concentration on what I consider the more important initial question of how the interactions between participants are accomplished and what purpose those interactions serve'.

Whilst agreeing with Minneman's contention that an understanding of the interactions is the more important initial question, the work reported here assumes that the purpose of those interactions and, therefore, the interactions themselves, cannot adequately be understood without regard to the quality of the results. The social processes of designing are intended to achieve the overall goal of designing. The participants are not just milling around 'interacting'. They have an overall intention to produce designs and, generally, good designs.

Minneman's [1991] account of group 'design' meetings also leaves analysis implicit and analysis activities intertwined with design activities. He uses the term 'design' to cover both of what this work refers to as 'analysis' and 'designing'. Analysis is implicit in his accounts of 'designing'. This follows the very common usage of the term 'design' to cover a range of development processes, often covering analysis and design and sometimes extending to implementation. (Note the common use, for example, of 'software designer' for 'software developer' and 'participatory design' for 'participatory development'.)

Despite this common approach, it is useful, perhaps necessary, to make analysis explicit, rather than to leave it lurking as an implicit part of design. There are important differences of emphasis between analysis activities and design activities. Analysis activities are always intertwined with design activities (see, e.g., [Swartout and Balzer, 1982]), but leaving the difference implicit fails to address the difficulties for developers of defining the design problem. Minneman's [1991] lack of interest in the quality of the product of the design work he studied and his failure to distinguish between analysis and design activities may both be symptoms of his focus on the activities *qua* social interaction, with little regard to their overall purpose.

Minneman [1991] argues that 'design practice can be effectively characterized as being a negotiation process, where the nature and particulars of the design *emerge*

from the various social interactions' [p.128; emphasis in original]. Therefore, if research, in addition simply to describing and explaining what it is to design, is to suggest how the design process may be made more effective, then that research must take account of the 'nature and particulars' of the emergent design, specifically its quality. Thus, this research is interested in explaining what goes on in the design (or, rather, development) sessions and also in relating this to the quality of the resulting products.

Chapter six of this thesis deals with the quality of the software systems which were products of the studied development projects and relates an assessment of this quality to the development activities. It is worth noting that the software is not the only product. We also have the task models (and paper prototypes etc) which are the structuring and representation of the information which has been elicited by and from the participants. Thus, a measure of the quality of *these* products is how well validated and understood the information, structure and representation actually are.

4.1 Developing interaction analysis

This section describes and presents results of a first comprehensive attempt inductively to derive a descriptive, and ultimately explanatory, account of user-developer interaction in task driven cooperative development. Each of sections 4.1.1, 4.1.2 and 4.1.3 presents an important stage in the developing analysis. Each of these sections describes analytic work which was done, produces an account of a particular aspect or theme revealed in the data and presents examples of transcripts from the video corpus to illustrate that aspect or theme and the developing theoretical account. In the early part, the analytic work was more exploratory, looking for issues and themes within the data. As the analysis progressed and a theoretical account of the interaction began to cohere, the analytic work tended towards developing the account. This tendency is reflected in the reporting of the analytic work across sections 4.1.1 to 4.1.3.

Other than a strong tendency, common in interaction analysis, to use chunking of events as the main focus of initial analysis effort, there is no simple relationship amongst sequence time, analysis time and analytic foci. A particular interaction sequence may be visited many times with different questions in mind. The same piece of interaction may serve multiple functions in the domain and may fruitfully be addressed with multiple analytic foci. A particular piece of interaction may be revisited with insight gained from analysis subsequent to its last examination. Perhaps most importantly, sequences of interaction may be revisited to ground and if necessary to revise the current theoretical account developing from the analysis.

As noted in chapter three, the highly iterative nature of interaction analysis makes for difficulties in presenting the results. Therefore, the description given here of the analytic work may at times be structured in terms of analytic foci. This does not necessarily imply that the analysis at any stage undertook a linear progression through particular analytic foci. Neither does it imply that all analytic foci were

equally valuable in addressing a particular issue or that a particular analytic focus had no relevance to an issue.

With these *caveats* in mind, however, a general indication may be given of which analytic foci proved most useful in addressing particular issues or at particular stages of the analysis. As noted above, the analysis used Minneman's [1991] framework of design activities to identify important processes, artefacts and relations. Analytic foci described by Jordan and Henderson [1995] also helped to guide the analysis. These analytic foci are orientations or 'ways-into-a-tape ... in intended distinction from analytic categories or coding categories' [Jordan and Henderson, 1995, p.57]. Analytic foci which proved useful included trouble and repair in interaction, participation structures and spatial organisation of activity. Turntaking and an analytic focus on artefacts and documents proved more useful in addressing the concerns of chapter five, where participation structures and spatial organisation again proved to be useful analytic foci.

As described below, 'structure of events' was the main analytic focus in the very early analysis cycles. Initial analysis concentrated on identifying chunks of coherent interaction. Temporal organisation of the activity then became the most important analytic focus, in order to reveal repetitions of the identified chunks and to investigate the robustness of the interpretation across instances. Jordan and Henderson [1995] identify this strategy as a common first analytic step.

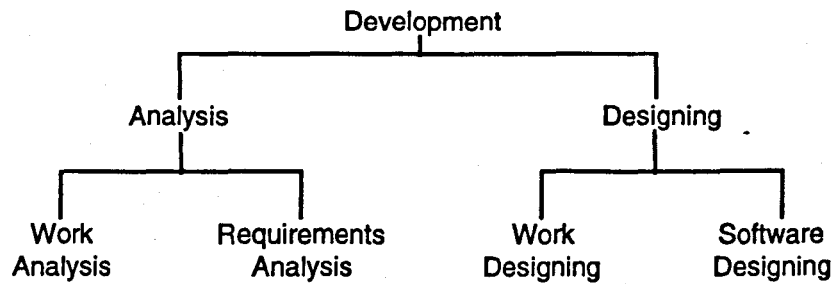
It is worth noting also that the following account of the analysis has been sanitised for the sake of coherence and readability. It omits a detailed account of the ceaseless iteration and occasional deadends which at times were pursued and are inevitable in the course of an exploratory analysis. As Minneman [1991, p.97] succinctly puts it, 'I'll be explaining this process with the benefit of 20-20 hindsight, so the reader will not see those times where days or weeks of struggle with the material produced the detailed account presented here'.

4.1.1 Identifying process in cooperative development

Taking the process facet of Minneman's framework, in first considering the question of what was going on in the development meetings, the analysis focused on identifying processes of interaction between developer and user. Initial analysis concentrated on identifying chunks of coherent interaction with recognisable boundaries. An overall structure to the interaction was first sought in the application of the systems development approach described in chapter three.

The approach to cooperative task driven development used in the projects differentiated between analysis and design activities. Within analysis, there was a distinction between analysis of the users' current work situation and analysis of the requirements for a new, computer-supported, work situation. Within design, there was a distinction between designing an envisioned work situation and designing a software system to support that work situation (see Figure 4.4).

Figure 4.4: Software development activities



Analysis of the development meetings looked first for evidence of a top-down structure of coherent interaction processes imposed by the development approach adopted. All four of the development activities in Figure 4.4 (work analysis, requirements analysis, work designing and software designing) were found to occur right across the development projects, with examples of all the activities often apparent within a single meeting. Thus, Figure 4.4 represents a decomposition, not a chronology. A pattern was apparent, however, in that earlier meetings were mainly concerned with work analysis while later meetings were mainly concerned with software system design. This pattern reflects a general and perhaps essential movement through the chronological course of software system development. The task driven method adopted in this development project may have accentuated the distinction between analysis and designing, so an interesting finding is that the different types of work did not happen in discrete stages. Rather, they were jumbled up.

The task modelling activities drew out the distinction between analysis (TM1) and design (TM2). However, they did not, for example, produce a single, unambiguous, 'complete' and static model of the current situation. During design, participants can - and did - return to and amend the model of the current situation. This intertwining of analysis and design is essential to supporting the emergence of information about the problem situation throughout the analysis and design processes.

For example, even when a meeting had been called to work on analysing the users' current work situation and this had been made very explicit, participants dropped in occasional discussions of an aspect of an envisioned software system design. One of the purposes this seemed to serve was to link the diverse parts of the overall development project, to relate the main ongoing activity to the overall purpose and to the other activities. For example, from a very early stage of a CI project meeting whose official purpose was current work situation analysis:

CI10945¹

D: If, for example, you had a system whereby you could actually generate an action which corresponds to getting the statement as a kind of way of getting you going, would that be useful?

Here, the developer both makes a design suggestion for the envisioned software and invites requirements analysis. This mixture of development activities was apparent throughout the projects. For example, at the other end of the CI project, when most activity was geared towards software system design and implementation, a user interrupted a meeting between the chief developer and the researcher specifically to bring up a point about the users' current work situation which he believed had been misinterpreted in the prototype software.

Swartout and Balzer [1982] argued that the specification and implementation of a system are inevitably intertwined in the course of systems development practice. In the projects studied here, notwithstanding a systematic progression through prescribed development activities, *all* of the identified development activities were to some extent intertwined. A clear example of this was in the aftermath of usability evaluation exercises in each project, when the developers response to poor usability features was a homogeneous mixture of task analysis, task redesign, software redesign and reevaluation.

Of course, not all activities in which participants engaged contributed, at least directly, to one or more of these types of development work. 'Deviations' from specific work activities included discussion of family holiday trips, discussions of how to maintain the room temperature at an acceptable level (during a very hot summer) and company politics. While these last interactions are not examples of one of the types of work identified above and seem to have little or nothing to contribute to the goal of developing a software system, this type of activity may well serve a useful function in bonding the participants as colleagues. This in turn may help lay a foundation for productive participation structures between people who now share a little more common ground: a theme to which this analysis shall return. However, the analysis reported here focused directly on system development specific interactions.

Having identified four broad types of work activity going on in the cooperative development meetings, the analysis focused on identifying and describing in detail the processes constituting these activities. This entailed identifying chunks of coherent interaction with identifiable boundaries. In studies of people performing activities which ranged from aligning blocks to creating tunes, Bamberger and Schön [1991] describe this process of 'bootstrapping' the initial interaction analysis cycle. On the first several passes of the recorded activity, they looked for boundaries delimiting chunks of interaction, 'without trying to be explicit about the

¹In the example transcripts, D refers to a developer and U refers to a user. Example transcripts are referenced by two letters for the project, one number for the video tape and four numbers for the tape counter.

criteria we were using or exactly what sorts of behaviour were signaling the boundaries we found. ... Once having found a chunking that seemed right, we went back and looked for the criteria we had quite spontaneously used' [p.187]. As they go on to note, the analyst must then be convinced that she has correctly identified the assumptions underlying the chunking.

A similar approach to the analysis of the development activities in the projects here identified a recurring pattern of interaction processes: an invitation to provide information, accompanied by an explicit reference to an external shared model (i.e. task model or prototype), a central interaction sequence and, finally, termination of the sequence, with another explicit reference to the external shared model. (See Figure 4.5.) For example, in an instance of current work situation analysis activity:

CS12515

D: And now presumably those lines [*points to lines on TM*] that you drew from reception presumably this links to administration in some way.

U: Yes, again what I was saying was [*gestures to TM roles*] if you like [*starts pointing to Reception TM role*] reception is the front line of all front lines. And anybody who comes in to reception, the idea is that reception know everything about what everybody else does but themselves know absolutely nothing about anything. So you come in to the receptionist and say who do I want to talk to and they point [*stops pointing to Reception TM role*] to one of these three other roles [*points to TM roles in turn*] or possibly directly to [*points to TM role*] one of these.

D: Right, so this was second line [*starts writing on TM*] you said?

U: Well, it was just that if we call reception first line [*points to Reception TM role*] which I guess you reasonably can then that makes these three [*points to TM roles*] in fact second line.

D: And that there

U: and that third line yes. [*points to role on TM*] [*D stops writing on TM*]

The boundaries here, when made explicit, were the references to the external shared model. This invariably involved the simultaneous orientation of the participants to a specific point or region of the model, the pointing to that point or region during invitation and termination and, in many cases, the manipulation of the external shared model on termination. This manipulation typically involved the updating of the external shared model to reflect the results of the central interaction sequence. Jordan and Henderson [1995, p.58] note that 'beginnings and endings are often marked by rearrangements of artifacts'.

Figure 4.5: Structure of development work activities

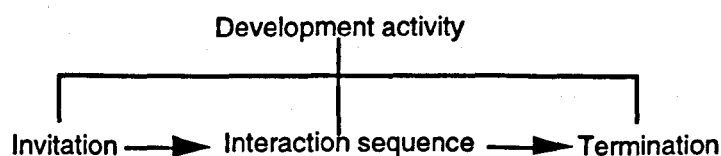


Figure 4.5 illustrates the structure of this recurring process within the development activities. This structure was first identified within current work situation analysis activities. Further analysis revealed that it also characterised the other three identified types of cooperative development activity: requirements analysis, work designing and software designing [O'Neill, 1996]. For example, in the following extract from a paper prototyping session, the developer invites the user to provide information on the most useful order for elements of the interface. An interaction sequence follows which the developer terminates by manipulating the paper prototype. The developer then invites another interaction sequence with a gesture to the prototype and the pattern continues.

CS71941

D: So which, which order then would be most useful to you? [*gestures to prototype*]

U: Uh I guess frankly to us the most useful things would be first contact details, then login then department, then type and last of all name. We don't care who they are, we want to know what categories they fall into [*smiles*]

D: Right.

U: for our purposes.

D: Right. But they're liable to give their name first, you, you

U: Oh yes. As individuals they tend to think they matter [*smiles*].

D: [*moves Post-Its on prototype*] Contact gets pretty much priority in both. [*moves Post-Its on prototype again*] Uhh. So. Is is that what you said was the ordering or is that? [*points to prototype*]

The interaction structure described often centred around the developing external shared model. As noted above, the initiating invitation was usually accompanied by an explicit reference to a part of the external shared model, attempting either to elicit from another participant information on an undeveloped part of the model or validation that the current version of a part of the model accurately represents what it is supposed to represent. The first participant's seeking elicitation or validation typically prompted a lengthy sequence of interaction with the second participant providing information about issues related to the part of the model in question. Finally, the sequence almost always was terminated by the first participant with an explicit reference to the external shared model.

Each activity might be initiated or taken up by any participant. Invitation may be, for example, a user saying, 'I have this problem ...' or 'I want this ...'. This is an example of the interaction modelled in Figure 4.5 in the context of requirements analysis activity. An analysis of contributions to each type of development activity initiated by each type of participant is taken up in detail in chapter five.

It was, however, apparent that interaction sequences in general were much more frequently initiated and terminated by a developer than by a user. This illustrates that cooperative development does not mean placing development in the hands of users. It remains the job of the software developer to develop the software and the

developer must retain ultimate responsibility for guiding the development processes. However, cooperative development does facilitate the user's taking the initiative when appropriate. For example, in the transcript below, the first developer (D1) has suggested that they trace object flows through a task model. The user immediately interjects with the most important objects to follow. They agree to do that and then the user has to tell them where to begin.

C110502

U: Well, the most important thing to follow would be a statement followed by an action.

D1: Do that then?

D2: Yeah.

U: Right.

D2: Uh ... where do we start?

U: Well, initially, before the incident room was probably set up, this guy [*points to TM role*] would probably get a statement.

D1: Yeah.

U: So you can actually start [*points to TM role*], here. OK?

D1: Yeah.

A key thread in these interactions was that participants' activities were geared towards developing shared understandings. Trouble and repair of participants' understandings appeared to drive these chunks of interaction. The initiating invitation usually was to provide information on or verification² of an understanding, while the central interaction sequence included many mutual verifications of understanding by the participants. A complex pattern of mutual verification was very often apparent within the central interaction sequence whether the initiating invitation was for elicitation of information or for verification of an understanding.

Just before the following example transcript, the discussion has wandered down a sidetrack and D1 here pulls it back, with a gesture to the task model, to tracing object flows. The ensuing interaction sequence is long and complex, using the object flow representation as a frame on which to build a shared understanding of the tasks performed in several roles. Eventually the sequence is terminated by D1 with another gesture and reference to the task model ('OK? So then we have exhausted that bit?') and acceptance by U ('Yes. Quite happy with that.').

² The term 'verification' is often used to refer to the comparison of models or artefacts, e.g. verifying that a software design matches its specification. Hence, verification is used frequently in this and the following chapters to refer to the comparison of internal and external models, e.g. checking that a user's understanding of a particular feature matches a developer's understanding of it.

CI11106

D1: Tell you what, [*starts pointing to TM*] let's go back to just following the statements. The normal way that a statement comes in is as the result of an enquiry, and [*moves finger, continues pointing to TM*] attached to a returned action. [*moves finger, continues pointing to TM*] This goes to the receiver

U: Yes.

D1: He chooses the [*points to TM*] next document and then he does [*gestures down TM*] whatever he has to do to receive it.

U: Right. [*starts pointing to TM*] It goes to the receiver. If there is any property or exhibit brought in with the action [*D also starts pointing to TM*], it would go there, [*D stops pointing*] give him the exhibit, get an exhibit reference number on his action form, and then go [*moves finger, continues pointing*] here and [*moves finger, continues pointing*] here. [*U stops pointing*]

D1: So are you saying that actually the returned action and the exhibit goes through [*points to TM*] that path?

U: Yes.

D1: What we've got [*points to TM*] is an exhibit going there, separate ways from the action, and you're saying that the two are kept together.

U: [*points to TM*] This officer will take his bundle of stuff and go [*starts pointing to TM*] here, because if you look on the action form it asks for an exhibit reference number, and this is the guy that can give him the exhibit reference number. And he should also give him a statement of recovering that exhibit. [*stops pointing to TM*]

D1: Right. [*starts pointing to TM*] So what we've got here is that the [*stops pointing to TM*]

U: Now, [*starts pointing to TM*] exhibits are a bit of a dodgy area. We have started hitting some snags already. Some of the exhibits will be stored in the incident room, so they merely go there, they're referenced by him in his exhibits book.

D1: Mmm.

U: OK? Put a number on it and then come back here and go through this system. [*stops pointing to TM*] Now, some of them will be documents that can be happily kept in the incident room. But if they are forensic items, they will stay [*starts pointing to TM*] there and he will look after the forensics. The easiest way to put it is that he will look after the forensics stuff, but any other stuff like document exhibits will go through this channel. It must go there first for numbering, and he should, in practice he *should* give him a statement of recovery, [*D1 starts writing on TM*] [*U stops pointing to TM*] because again if. Now you won't find this in MIRSAP you see. It's the praxis thing that's important.

D1: A what thing?

U: It's the practical application that's important. You might not find it in your MIRSAP or whatever you call it. If and I'll tell you why. [*D1 stops writing on TM*] [*U starts pointing to TM*] This guy is going to end up with thousands of exhibits, if not tens of thousands, depending on the job. Yeah? [*U stops pointing to TM*] At the end of the enquiry, somebody has to find out and get all these statements to find continuity. Even if its only two hundred exhibits out of these thousands, he's got to get two hundred statements, possibly. It is also difficult for the officer because he has got to backtrack all through the database and his pocket-books to say, well, I found the cigarette packet

at nine o'clock at so and so location, and I found the matches at so and so location ten minutes later, y'know. Now, it is all done and dusted [*gestures to TM*] at this stage. It is all right for the guy that has done it, but there's no chasing. It saves an awful lot of time and resources.

D1: If there's not an exhibit, [*starts pointing to TM*] it goes through that route. If there is an exhibit, then the exhibit and the returned action ...

U: Yeah.

D1: goes [*stops pointing and starts writing on TM*] through that route ...

U: Yeah.

D1: with an exhibit number ...

U: Yeah, the exhibits book number.

D1: is generated there, [*stops writing and starts pointing to TM*] a recovery statement made.

U: The exhibit book number [*starts pointing to TM*] here, *his* exhibit book number, which is a manual book, ...

D1: Yeah.

U: could be different to anything that is registered over here. [*stops pointing to TM*]

D1: Yeah, OK.

U: OK.

D1: OK. So that number [*points to TM*] needs to be incorporated and recorded into the system.

D2: If the exhibit is a document, is it the document that gets passed on or a copy of it?

U: Right, if it was for finger-printing or any forensic data then it would have to stay here, [*points to TM*] and if [*starts gesturing to TM*] you can take a copy of it without contaminating it or anything else, it can carry on and flow through. [*stops gesturing to TM*]

D2: But the original would never be sent through?

U: The original document would go through, yes. A statement can be an exhibit. A list of vehicles in a street could be an exhibit, but that does not need to stay with this guy. [*points to TM*]

D1: But when it [*starts pointing to TM*] comes to here, you've got this choice of either going to the exhibits officer or not, and you say that even a statement can be an exhibit, how [*stops pointing to TM*] do you know whether it is going to be an exhibit or not?

U: You don't. You don't know. You don't know how anything's going to be used in an incident. You set off, on most occasions, with a body and a scene. All right? And basically you saturate both to get intelligence about the body, because that's all you've got. You gather intelligence about the two things that you have got: you have got a scene and a body. Now what you take out of that... the deceased's antecedent history might not be relevant at all, but it might be very relevant when you associate who has done it. But if you've gathered a load of antecedent history of associates and it turns out to be a complete stranger that has done it, then all that stuff you took was irrelevant, no use, but you don't know that at the time. If you knew at the time then the

investigation would stop here [*points to TM*] because you would go straight to the person that has done it.

D1: Right.

U: Yeah.

D1: [*starts gesturing to TM*] But, I mean, not everything that the enquiry officer brings in will go to the exhibits officer.

U: No, that's right.

D1: So.

U: This [*U starts pointing to TM*] [*D stops gesturing to TM*] guy, the enquiry officer won't bring in that many exhibits because this chap, there could be two of these guys. [*U stops pointing to TM*] There could be scenes of crimes guys. There will be a number of officers, maybe two or three officers who will bring in the bulk of exhibits.

D1: Yeah.

U: There might be a search team, and they will find exhibits or likely exhibits. The exhibits officer is going to come across exhibits or likely exhibits. And if they use different scenes of crimes officers to exhibits officers, they are going to find exhibits or likely exhibits. It is becoming more and more important that a qualified scenes of crimes officer is the evidence gatherer, the exhibits gatherer at the scene, because the protection of that stuff is highly important.

D1: So [*starts pointing to TM*] let's just finish off this then. It goes through this route.

U: Right.

D1: The action and all the documents and exhibits go here.

U: Yeah.

D1: The officer will fill out the statement of recovery ...

U: Yeah.

D1: an exhibit number will be written on the various things

U: Yeah.

D1: Right. And then the whole bundle comes back through this route ...

U: Yeah.

D1: as if it came in ordinarily... [*stops pointing to TM*]

U: ...including the newly written statement...

D1: ...including this statement of recovery. [*points to TM*] OK?

U: Yeah.

D1: OK? So [*starts gesturing to TM*] then ... we have exhausted that bit. [*stops gesturing to TM*]

U: Yes. Quite happy with that.

4.1.2 Exploring artefacts and relations in cooperative development

In tandem with the development, described above, of a preliminary account of the processes observed in cooperative development activities, the analysis explored

these activities through the other two facets of Minneman's [1991] framework: artefacts and relations. Again, the framework was used as a resource for providing analytic foci and not, as Minneman [1991, p.113] cautions, as a framework for classification of discrete and countable acts.

This research included under the term 'artefact', in addition to visible, tangible products, such as task models, notes or software, such invisible but coherent products of development activities as, for example, a developer's internal model of the user's work situation or a user's internal model of the software system. For the purposes of this analysis, the term artefact was not, however, extended to cover such incorporeal products of the development activities as, for example, user commitment to the software or shared allegiance amongst the development team.

Whilst the external, shared artefacts, especially the delivered software package, are conventionally seen as the goal and measure of success of development work, the internal, private artefacts may themselves be important products of the development activities. For example, four users across the two projects stated independently and unprompted that for them an outstanding result of their involvement in the development work was an unprecedented understanding of their own work and that of their colleagues.

The development method adopted drove the construction and use of external artefacts in the forms of task models and prototypes. The participants' use of these external models entailed their construction of further artefacts which were their respective internal models, that is their respective understandings of the external models and of the worlds represented by the external models. The nature of these internal and external artefacts and the relations amongst them motivated this phase of the analysis.

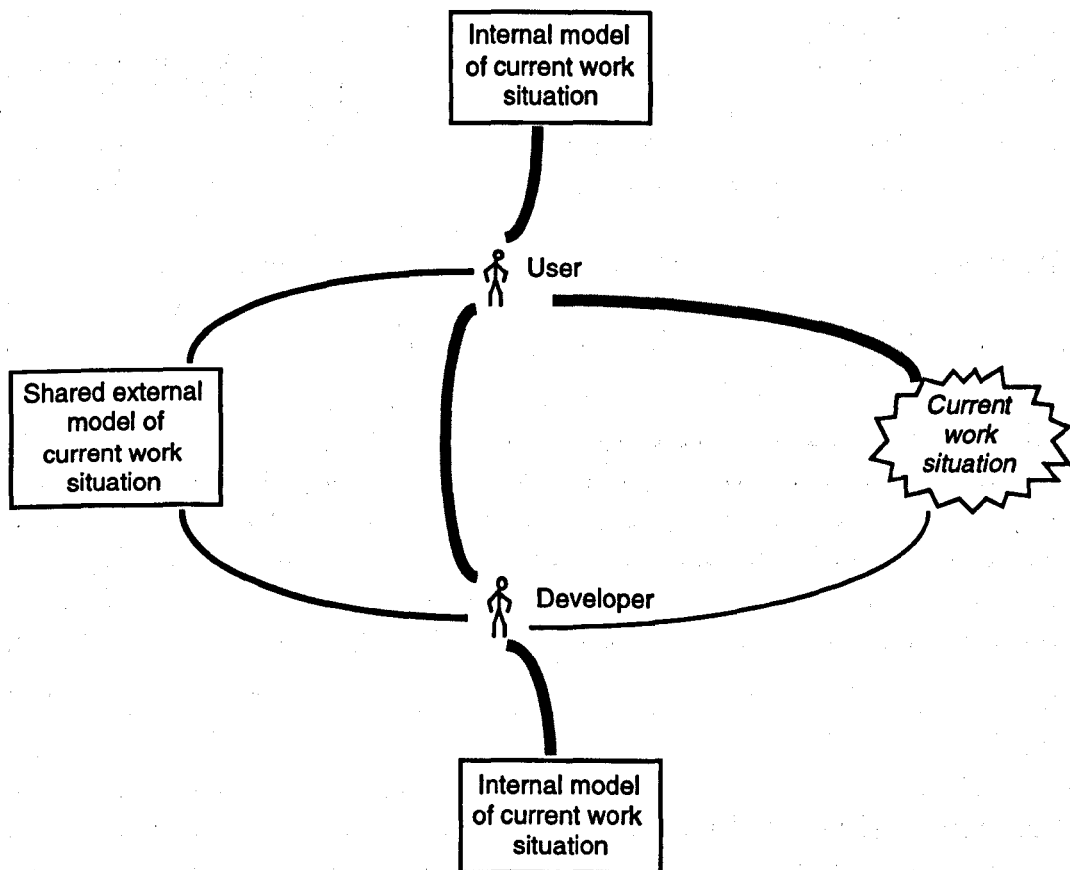
Relations of particular interest were those amongst the user, the developer, the object of the current development activity and the various models developed through that activity. Figure 4.6 illustrates a first analytic conception, for the cooperative development activity of analysing the user's current work situation, of the relations amongst the participants, the work situation which was in this case the object of the development activity and the models of that work situation produced by the participants. Similar illustrations are presented below of the relations for other development activities.

Relations are depicted in Figure 4.6 by the lines between the various entities. The user, for example, has direct access to the work situation which is to be supported by the software. This work situation includes the user's tasks, colleagues' tasks of which the user has knowledge, the various roles in the situation with which the user is familiar and the working environment in which the users' tasks are situated. The user has access to her own internal understanding or model of that work situation and to the external, shared model of that work situation. The user also has access to the developer, through their interaction.

The thickness of the connecting arcs in Figure 4.6 (and in Figures 4.8 to 4.10 below) is intended to convey the relative strengths of the relationships between the connected entities. In work analysis activity, the relations associated with the developer mirror those of the user. The notable exception, indicated by the lighter line in Figure 4.6 between developer and work situation, is the developer's relative lack of familiarity with the target work situation. This is inevitable when the developer is not also a user, as is usually the case.

The idealised goal is for all the participants to come to have understandings of the work situation which, *for the purposes of the current development activity*, are comprehensive, accurate and consistent both internally and with each other. This is not to suggest that participants constantly strive to homogenise their understandings. As Minneman [1991] has described, understandings are often left contradictory and incomplete. But conscientious participants will attempt at least to acknowledge where they have not negotiated an agreed understanding.

Figure 4.6: Simple models and relations in work analysis activity



In the following example, the participants simply stated their respective positions (on the meaning of 'registration') and moved on. In the final line of the example transcript, D2 simply ignores that the user has demonstrated a different understanding and invites a new interaction sequence on another role in the work

situation. Interestingly, the interaction sequence on registration was not anchored by bounding references to an external shared model. The failure to establish a shared definition of registration or, at least, to acknowledge different definitions, lead to breakdowns at several later points and eventually a shared definition had to be established.

C114416

D1: When you say registered, is this the same registering ...?

U: Registering it is normally giving it a number.

D1: There is also giving it the nominal phone and address. In indexes, that is usually part of the registering, but is that ...

U: But with a message it is slightly different. If you go right into the pre... If you type it then it purely gives that message a number. I am not sure what it actually does with the ...

D2: We have taken, as a working definition, we have distinguished between the allocator and the number and the complete registration. So we have defined registration to be that its existence is based upon the system, you have allocated it a number, plus it is cross-referenced to the standard thing it should be cross-referenced to. That is what we deem to be registration.

U: Registration is whereby a document is registered to an individual and given a unique number. Registered to the author and given a unique number. That is why the message is a bit different. If I run the procedure right: if the manuscript document goes here... this is what is confusing me, you could have variants of themes on how the thing works here..

D2: Can we distinguish between the index and action writer and the person...

Another important aspect of Figure 4.6 is the absence of direct relations between a participant and the internal model of her coparticipant. In attempting to derive a shared understanding or model of the work situation, the participants cannot directly access their coparticipants' current understanding and must, therefore, rely on the interactional relation between the participants to provide information on their respective understandings. In particular, whereas the user constructs an understanding, or internal model, of the work situation primarily from her direct experience of the latter, the developer typically has much less such experience and, consequently, relies more on information gleaned through interaction with the user.

In the absence of participants' direct access to other participants' internal models, the shared, external model comes into its own as a representation of a current understanding of the work situation to which all the participants have direct access. The external, shared model makes a particular understanding of the work situation copresent with the participants. For example, one may point to a visible, and therefore readily identifiable and communicable, part of the external, shared model and seek information on that topic or express one's current understanding of it.

Consider the following example.

CI10554

D: [*starts pointing to role on TM*] What we've got is that the inquiry officer delivers a returned action. The returned action includes some documents.

U: Statement, PDF.

D: [*using both hands points to two roles on TM*] This goes to the receiver and he does what he has to do with it.

U: Yeah.

D: And you find that you get a um a completed action with documents. Right? So you've transformed from a returned action into a completed action. And that'll be marked up with urgent actions um and so on. It then goes to this guy [*moves finger, points to two roles in TM*], the indexer action writer [*stops pointing with both hands*] and you'll see that [*starts pointing to TM*] he's got a resulted action. And then they get separated in terms of you get a registered statement manuscript and a report manuscript. [*stops pointing to TM*]

U: [*starts pointing to TM*] So we're saying here then that the action, the complete action and documents, that includes statements?

D: Yes.

U: Right. [*stops pointing to TM*]

D: [*gestures to TM*] We haven't separated them yet, [*gestures to TM*] the bundle's all at one end. But what you are saying is that when you start the incident up, some things will come in without an associated action.

U: Some. [*points to TM*] There will be some statements or documents taken before that probably is running.

Here, the developer initially believes that a description just given by the user of a particular set of tasks in the work situation does not accord with how the external shared model currently represents those tasks. So, the developer recounts his understanding of what the task model currently represents. The user in turn recounts his understanding of what the model represents, seeking verification from the developer of the user's understanding of the model and clarification of an unclear part of the representation ('... *that includes statements?*'). The developer provides this and then seeks verification from the user of the developer's understanding of the user's recent description of the work in question.

In addition to aiding the assimilation and verification of understanding, the external, shared model also serves to store and to structure information made public through the interaction. Thus, the external, shared model simultaneously supports and is a product of the interaction (Figure 4.7).

Figure 4.7: External shared model as support for and product of interaction

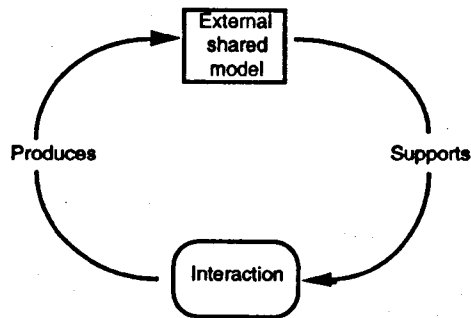
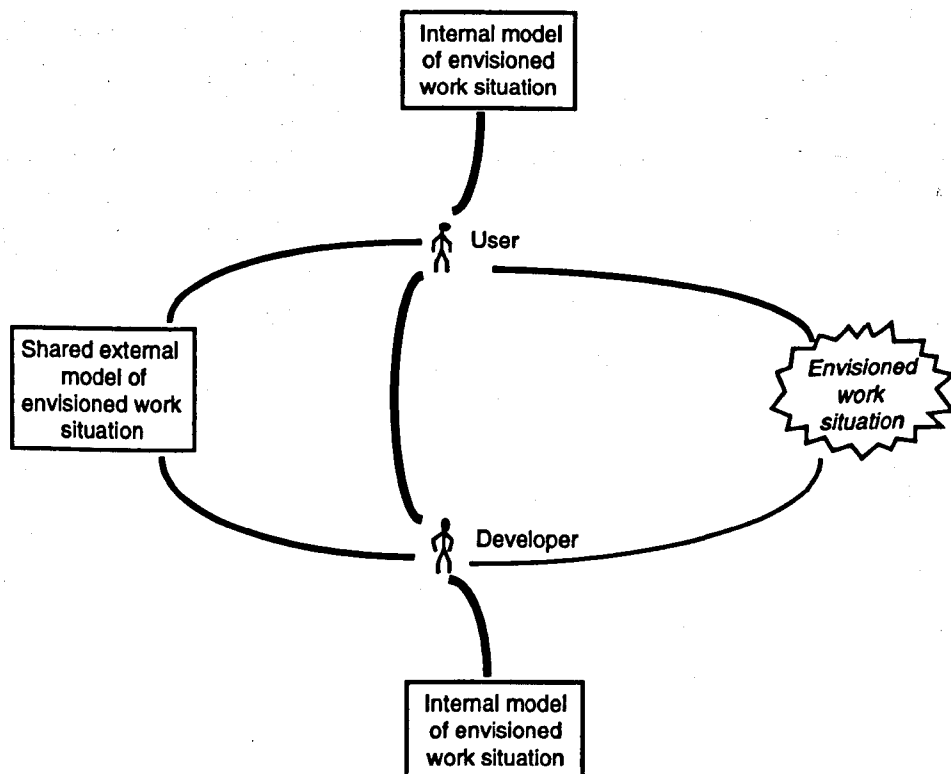


Figure 4.6 illustrated the relations which pertain in the development activity of current work situation analysis. Moreover, an almost identical set of relations pertains in the development activity of designing an envisioned work situation. (See Figure 4.8 below.) Within the development activity of work analysis, the models were of the users' current working situation (for which the external shared model is TM1). Within requirements analysis activity, they were models of the requirements which must be met by a system to support the work situation (for which the external shared model is a requirements specification - formal or informal). Within work design activity, they were models of the envisioned work situation (for which the external shared model is TM2). Finally, within software design activity, they were models of the software (for which the external shared model is a paper or software prototype).

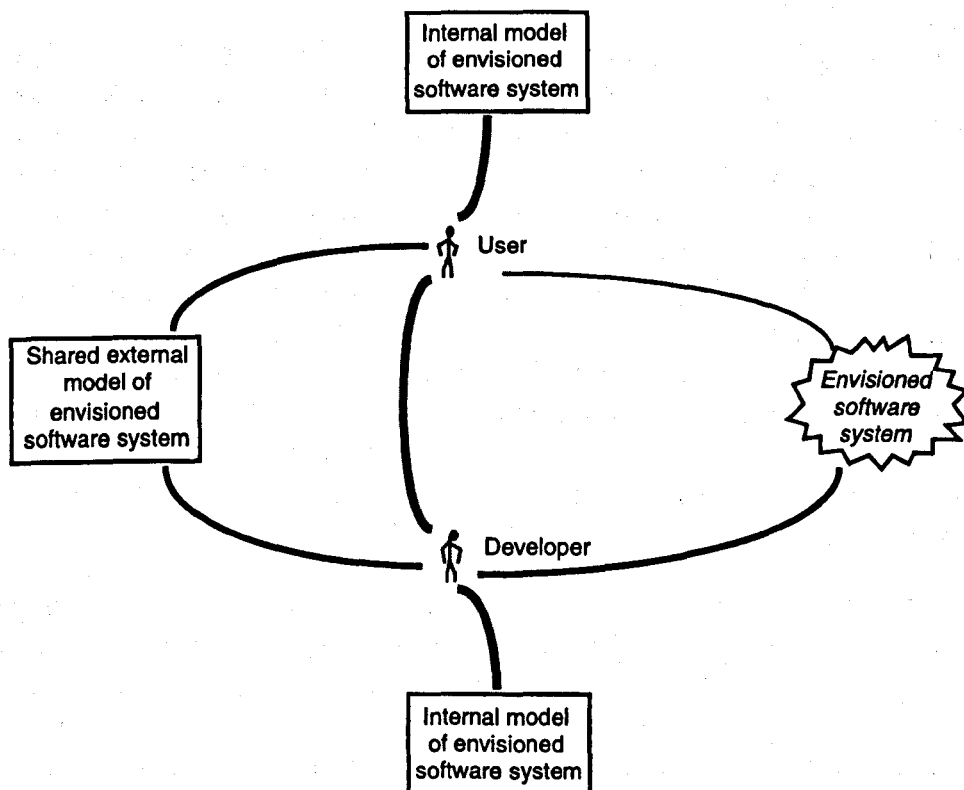
Figure 4.8: Simple models and relations in work design activity



Whereas the shared external model in work analysis activity (Figure 4.6) is a task model of the current work situation which is the target of the development project, the external shared model in work design activity (Figure 4.8) is a task model of an envisioned work situation. The most notable difference in relations with an envisioned task model is a weakening of the user's relation to the work situation being modelled. Developer and user are modelling an abstraction, a work situation which does not yet exist and of which, therefore, neither participant has intimate knowledge or experience. They are guided in this difficult activity by their current understandings of both the current work situation and the requirements to improve it. The user's intimate knowledge of the current work situation is likely to be of help provided that the envisioned work situation is not radically altered from the current work situation or that she understands and can rationalise the changes.

A similar set of relations also pertains in the development activity of software designing. Here, in addition to the internal models, the external shared model in the observed software design activities took the form of a prototype representation of an envisioned software system (see Figure 4.9). Various media may be used for prototype construction including paper and software. Both these media were used in each of the studied projects. Each of the participants develops an understanding or internal model of the envisioned software system and once again the idealised goal is for these models to be comprehensive, accurate and consistent both internally and with each other.

Figure 4.9: Simple models and relations in software design activity



It is a fine point here that the 'envisioned software system' of Figure 4.9 is an abstraction, in a similar manner to the 'envisioned work situation' abstraction of Figure 4.8. That is, the envisioned software system does not currently exist, although it is intended that at some time it shall be brought into concrete existence. The already concrete prototype to which the participants have direct access is *a model of the envisioned software system*. It is not the envisioned system. If 'evolutionary prototyping' is pursued, a software prototype may yet become the concrete basis for the developed software package. However, a strict proponent of 'throwaway prototyping' should ensure that this does not happen.

A notable difference between software design activity and other development activities is the relative strengthening of the developer's relation with the object of the development activity (in this case, the envisioned software) due to the developer's greater experience and knowledge of software systems, while the user's relation with the envisioned software system is likely to be relatively weak, due both to lack of experience and knowledge and frequently to a consequent lack of confidence. This is apparent in the following extract from a paper prototyping session.

CS70433

D: So we've got a pretty good idea of what fields we want on the screen. It's just a question of how we're going to lay it out and what's going to be there.

U: I'm not claiming to have any knowledge to be relied on about how best to lay things out. You could deliberately design a form to be as worse laid out as it possibly could be and I wouldn't know the difference.

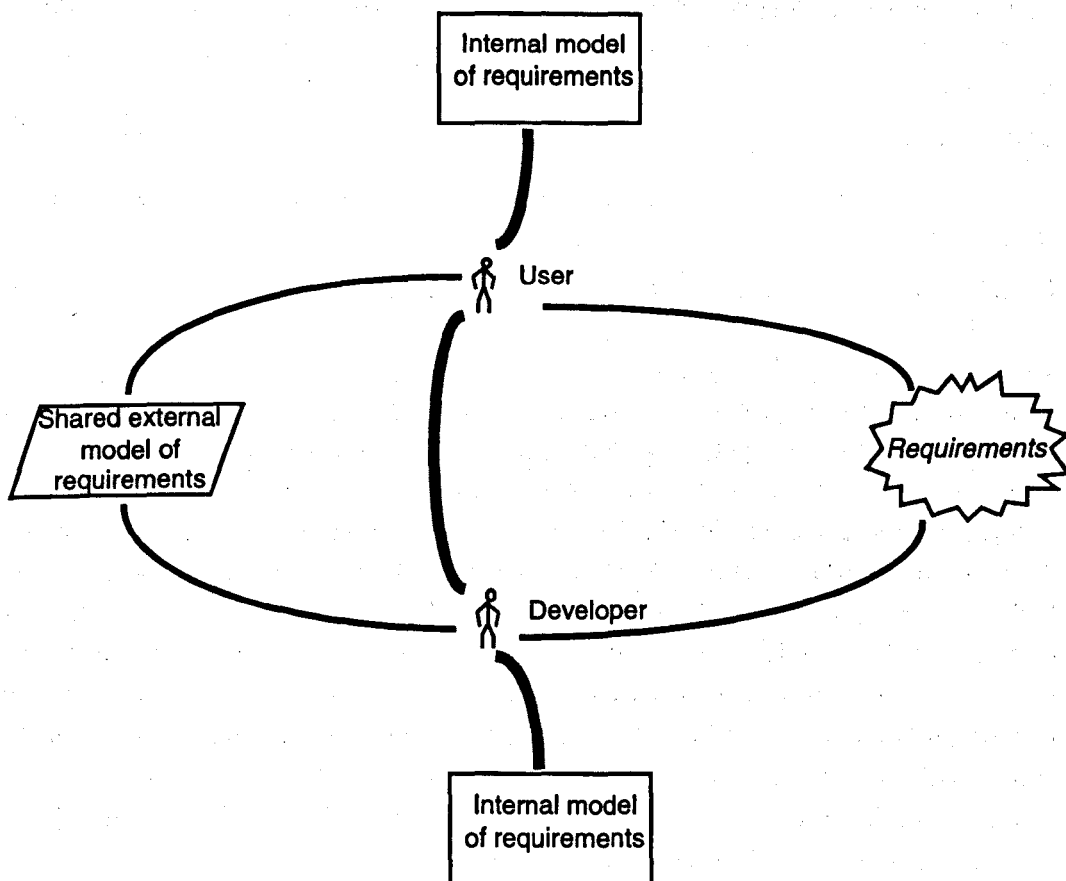
Several researchers have examined the nature and role of 'user models' and 'developer models' of software (e.g. [Dagwell and Weber, 1983; Stagers and Norcio, 1993; Whitefield, 1990]). In many of these analyses, the developer's model of the system tends to be a model of an envisioned software design in an envisioned work situation, whereas a user's model of the system tends to be a model of an extant software system in an extant work situation. Also, the user's models, particularly of the software system, are use-oriented while the developer's models are technological or construction-oriented. This reflects conventional development habits of the developer's attempting to derive an understanding of the work situation and requirements and to design a system with little or no cooperation with users. The user meanwhile is left to attempt to derive an understanding of a new software system in an altered work situation, both of which she has had little or no previous knowledge.

A set of relations similar to those in software design activity also held when the participants were engaged in the activity of analysing and specifying requirements for a software system to support the users' work situation. However, in the two projects studied, users and developers did not work together on a concrete, external model of requirements. In the CS project, a natural language list of requirements

was drawn up by the developers early in the project, on the basis of interviews with the user. There was some initial discussion with users on validating the list and it was subsequently used simply as a checklist by developers. In the CI project, a text based requirements specification was produced but its circulation and use were confined to developers.

In both projects, users and developers did cooperate in discussing and defining requirements but this work was performed not explicitly in relation to an external shared model of the requirements but in relation to an external shared model of the current work situation (TM1), of the envisioned work situation (TM2) or of the software system (prototypes). TM1 and the work performed in its development served as sources of requirements, while the external shared model of the envisioned work situation and of the software system served as embodiments and reminders of particular requirements.

Figure 4.10: Simple models and relations in requirements analysis activity



Thus, participation structures in requirements analysis activity were once again organised around a task model or prototype which both served as a resource to support the ongoing activity and was developed and refined through that activity. Participants often explicitly noted requirements on the task models.

In the following extract, an interaction sequence on an aspect of the current work situation is grounded in a task model of that current work situation and terminates with the establishment of a particular requirement for the software system.

C111419

D1: If there's not an exhibit, [*starts pointing to TM*] it goes through that route. If there is an exhibit, then the exhibit and the returned action ...

U: Yeah.

D1: goes [*stops pointing and starts writing on TM*] through that route ...

U: Yeah.

D1: with an exhibit number ...

U: Yeah, the exhibits book number.

D1: is generated there, [*stops writing and starts pointing to TM*] a recovery statement made.

U: The exhibit book number [*starts pointing to TM*] here, his exhibit book number, which is a manual book, ...

D1: Yeah.

U: could be different to anything that is registered over here. [*stops pointing to TM*]

D1: Yeah, OK.

U: OK.

D1: OK. So that number [*points to TM*] needs to be incorporated and recorded into the system.

As noted above, the thickness of the connecting arcs in Figures 4.6 and 4.8 to 4.10 conveys an indication of the relative strengths of the relations between the various participants and models. In the studied projects, the weakest relations were often between the developer and the current work situation, the developer and the envisioned work situation, and the user and the envisioned software. The developer-current work situation relation tended to be weak because the developers had not actually worked in that situation and depended on information from observations, manuals and interaction with those who did work there. This weakness tends to propagate to weakness in the developer-envisioned work situation relation, unless the envisioned work situation is radically different from the current work situation. The user-envisioned software relation tended to be weak, as noted above, due to lack of detailed experience and knowledge of software systems and a consequent lack of confidence.

Some difficulties in development projects may be characterised as a result of relatively weak relations, others as a result of a mismatch between different participants' relations with the same object or model. The weaker relations suggest possible reasons for usability problems with the resulting software. The developer knows the software well but may not have a sound grasp of the work for which the software is to be used. The user has a keen grasp of her work but may have a poor

understanding of the software with which she is asked to perform that work. Mismatches between the relative strengths of relations may also contribute to troubles in development activities when mismatches remain unacknowledged and participants have consequent difficulties in establishing shared understandings.

Strengthening the weaker relations may provide more effective development. To do this, it is necessary to allow the user more effectively to share the developer's knowledge and *vice versa*. The weak link between the developer and the work situation might be strengthened by the developer's spending time working as a user in that work situation. In addition to the well-known difficulties associated with this approach, such as the long time required to make it useful, this approach does not address the even weaker (i.e. non-existent) relations between the participants' respective internal models. While such an 'ethnographic' approach may usefully strengthen the relation between developer and work situation, contributing to the developer's internal model of that situation, there is no certainty that the developer's resulting model is consistent with that of the user. Cooperative analysis (and modelling) is still needed to strengthen the other relations and, in particular, to compare user's and developer's models of the work situation.

Cooperative development also goes some way towards increasing the strength of various relations by drawing the user into the development process, demystifying software and its development and, thereby strengthening, for example, the user-envisioned software relation. Strengthening relations and revealing and ameliorating mismatches between relations depends fundamentally upon communication and cooperative development attempts to aid such communication across the cultural boundaries which separate user and developer.

So, we now begin to have an account, derived inductively from the projects studied, of the processes, models and relations with which participants are engaged across their cooperative development activities. This account involves the inference of constructs including mental models from the observed behaviour of the participants. Some interaction analysts have taken an almost behaviourist line on refusing to treat participants' internal, mental constructs. For example, Tang [1989, p.3] argues that 'it is the *expression* of the idea that is publicly accessible, while the mental state of an *idea* remains unavailable for observation'.

Similarly, Minneman [1991, p.102] states that the goal of analysis 'is to produce an account of the communicative activity of the participants without having to resort to speculations about intentions, cognitive states, and other constructs to which we, as analysts, have no access', arguing that 'this appeal to sensory (primarily visual and auditory) evidence is hardly a harshly restrictive analytical stance - the other participants in the room have little access to the cognitive activity of the other participants either. This analytical focus has a practical bent; it is aimed at illuminating how it is that participants in a situation collectively function and, as such, it makes use only of those elements of the situation that are available to the other participants'.

However, it should be emphasised that inferences, rather than speculations, can - and, indeed, must - be made from the observed behaviour in order to develop explanatory, rather than purely descriptive, accounts of that behaviour. Minneman describes one of his two main contributions (the other being methodological) as 'a detailed *description* of the activities that constitute group engineering design practice' [1991, p.96]. The work reported here, on the other hand, set out to produce an explanatory account of cooperative software development practice.

Also, to argue the merits of restricting the analyst to observable phenomena on the grounds that participants in the interaction are so restricted is to miss the point that these same participants continually infer meaning and intention in their coparticipants from the latter's externally observable behaviour and their own knowledge of context. Indeed, as this work shall demonstrate, and has been demonstrated by others (e.g. [Clark, 1996; Lewis, 1969; Sperber and Wilson, 1987]), participants in interaction spend a great deal of time striving to understand, by inference and otherwise, the internal, cognitive states of their coparticipants.

The involvement of the author as a participant in the development situations facilitated the analytic inference process. While not assuming that the other participants viewed and understood the situation just as he did, the author's experience as a participant allowed him to bring to the analysis a comprehensive first hand insight into the shared understanding which developed through the project meetings.

Minneman [1991, p.15] concedes a similar point in acknowledging 'the need to judiciously admit [*sic*] the content knowledge background that typical participants in such a situation would possess. ... As an analyst, I permitted myself to view and interpret the tapes as I might have as a participant in the room, bringing all of my facilities as a designer to bear on making sense of the emerging situation'. Indeed, it is difficult to see how one might have interpretation without inference.

In general, the more extensive a participant's contextual knowledge, the more accurately she can infer her coparticipants' meanings and intentions from their actions and speech [Clark, 1992]. Thus, in utilising knowledge from personal involvement in the interactions, the analyst is drawing on the same resources as the participants. As Minneman [1991, p.18] notes, 'design communications serve as the primary means by which the social work of designing is accomplished; by virtue of being available to other participants in the design effort, these communications are also available for design research'.

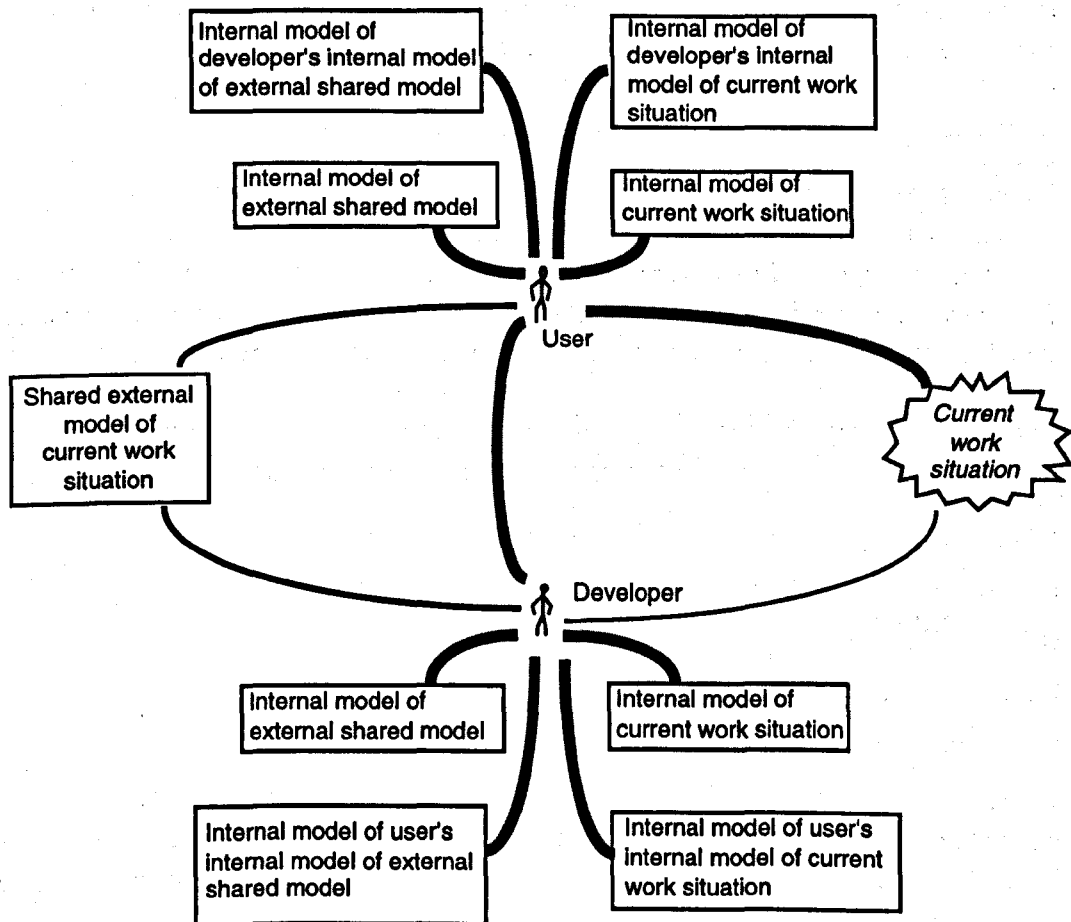
It is, moreover, a primary tenet of interaction analysis that interpretations and hypotheses should be firmly grounded in the recorded data. Thus, although 'ungrounded speculation about what individuals on the tape might be thinking or intending is discouraged ... it is not the case, then, that intentions, motivations, understandings and other internal states cannot be talked about in interaction

analysis. Rather they can be talked about only by reference to evidence on the tape' [Jordan and Henderson, 1995, pp.45-6].

So, following the iterative analytic cycle between perspective and account (Figure 4.3), this work once again compared the developing theoretical account of participants' internal models presented above with the interactions observed in the software development projects. Having brought the relatively simple account of development activities presented above back to ground it in and compare it with the data, it became apparent that a rather more complex account of models and relations was required to explain the observed interactions.

This further analysis produced an expanded account of the models and relations in cooperative development activities. Figure 4.11 below illustrates this expanded account for the cooperative development activity of analysing the user's current work situation. Again as for the simpler account above of models and relations, very similar illustrations may be presented for the development activities of requirements analysis, work designing and software designing. These illustrations are omitted here for the sake of brevity.

Figure 4.11: Models and relations in current work analysis activity



The obvious difference between Figures 4.6 and 4.11 is the inclusion in Figure 4.11 of three more internal models for the developer and user respectively. This more complex account derives from further analysis of participants' activities with regard to how participants might be using the postulated models and relations and what efforts, if any, the participants might be making to overcome the limitations of weak relations.

Example transcripts supporting this analysis are presented at several points below. Moreover, evidence for this interpretation may be apparent in reviewing example transcripts given previously. Hence, it may be illuminating for the reader to get a flavour of the interaction analysis method by returning to review previous transcripts with this more developed perspective.

In Figure 4.11, the participant no longer has just an internal model of the object of the current development activity: in this case, the users' current work situation. In negotiating towards a shared understanding of that development activity object, each participant attempts to determine the content of her coparticipant's model of the work situation. The developer primarily is interested to acquire information about the work situation and the user's perception of it. The user primarily is interested to check that the developer's understanding of the work situation is accurate; or, rather, that it is consistent with the user's understanding of the work situation. Both user and developer, therefore, are interested in comparing their own respective models of the work situation with that of their respective coparticipants. Informally, each participant is wondering, 'Are we talking about the same thing?'

However, since a participant does not have access to her coparticipant's internal model, she must infer its content. In making that inference, she constructs her own internal, private model of her coparticipant's understanding of the work situation. The evidence which is available to her (and, incidentally, to the interaction analyst) on which that inference may be based is the verbal and nonverbal communications which her coparticipant presents through their interaction. Often this evidence must be elicited by direct questioning. For example, in the extract below the developers explicitly seek an understanding of the user's model of the work situation.

C116556

D1: So are you saying that he actually marks up all documents for all actions?

U: Yes. At his level, yes.

D2: What about the statement reader?

U: He gets it later.

Thus, each participant has both an internal model of the current work situation at which the development activity is aimed and an internal model of her coparticipant's internal model of that current work situation. In determining consistency of the participants' current understandings, it is these models, to which a participant *has* direct access, which may directly be compared.

In principle, to be certain of shared understanding, to be sure that they are 'talking about the same thing', the participants need not just their second order models of their coparticipants' models, but third order models of the second order models and, ultimately, an infinite regress of such models. In practice, this is neither attainable nor necessary. This issue is taken up in detail in sections 4.2 and 4.3.

In addition to an understanding of the object of the development activity, each participant has an understanding of what is represented in the external, shared model. In work situation analysis, as discussed previously, this external, shared model takes the form of a task model. Although each participant has direct access to this model, individually and in collaboration, this is no guarantee that the participants share a common understanding. A participant does not have direct access to her coparticipant's understanding - or internal model - of the external, shared model and must infer *its* content. In making that inference, she constructs her own internal, private model of her coparticipant's understanding of the external, shared model. Once again, the evidence which is available to her (and to the interaction analyst) on which that inference may be based is what her coparticipant communicates to her through their interaction.

In the following example, the developer has introduced the user for the first time to a task model developed with other users. The developer is keen to check the user's understanding of the task model. The user has understood it well, as later becomes apparent, but uses terminology which confuses the developer. The user uses the term 'current model' to mean 'the current way of doing things in the work situation', with no connection to the use of models in the system development work. When he says that there are problems with the current model, he means that there are aspects of the current work situation which he does not like. The developer, however, takes 'current model' to refer to the current version of the task model with which they are working. He believes that the user believes that the task model does not model 'exactly right what happens' in the work situation. Hence, confusion has arisen between, on one hand, the developer's model of the user's model of the external shared model and, on the other hand, the developer's model of the user's model of the users' current work situation. The developer asks for clarification, expecting the user to reply in terms of what's wrong with the task model's representation of the work situation. The user instead replies in terms of a work situation practice - customers calling him directly instead of going through reception - about which he has previously expressed displeasure.

CS22357

U: This is the current model that you're modelling at the moment and and which is fine. There's obviously some problems with the current model as well.

D: Mm.

U: Although when I say it's not exactly right what happens, but we want to improve the current model as well to to take it another step above so

D: When you say that's not exactly what happens, what do you mean?

U: Well, they, there are, there are certainly the practices which people actually follow which are not quite strictly true, like people ring for example straight in. Those sort of examples.

So, each participant constructs both a private, internal model of what is represented in the external shared model and a private internal model of her coparticipant's model of the external shared model. Once again, in determining consistency of current understandings, a participant may compare these internal models. However, establishing a *shared* understanding of an external shared model - or, at least, noting where understandings differ - is made easier by the direct, collaborative access which participants have to this model. In attempting to infer her coparticipant's understanding of the external, shared model, a participant may refer to, enquire about and manipulate the model itself in the presence of the coparticipant. This process may also be aided by the history of collaborative development of the external, shared model. This model is a creature of the interaction. Its content, structure and notations all to some degree have been negotiated and created through the participants' interaction.

Hence, the external shared model is a resource directly accessible to the participants in a way in which the object of the development activity (in this case, the target work situation) is not. The difficulties in establishing a shared understanding of the target work situation have been discussed above. Approaches such as Contextual Inquiry [Holtzblatt and Jones, 1993] suggest that, in analysing the current work situation, the developer should interview the user in the user's normal work environment. Whilst such an approach may make copresent the participants and the object of the development activity, this object does not afford the assistance in developing a shared understanding of it which is afforded by an external, shared model.

In work analysis activity, for example, deriving a shared understanding of the task model is easier than deriving a shared understanding of the work situation. Even if, for example, the developer may refer to and enquire about the user's work *in situ*, it is unlikely that the developer will introduce any significant changes to that work, at least in analysing the current work situation. Also, the developer typically shall have had nothing to do with the creation of the current work situation. Indeed, the user may be in a similar position, having inherited a long established working environment and practices.

In the cooperative development activity of work situation designing, both user and developer have a hand in creating the envisioned work situation which is the object of the development activity and may, of course, introduce changes to that work situation. Once again, the goal is for all the participants to come to have understandings of the object of the current development activity (in this case, the envisioned work situation) which, for the purposes of that development activity, are comprehensive, accurate and consistent. For this development activity, in wondering, 'Are we talking about the same thing?', the developer is interested to

confirm that the user is agreeing to a particular designed work situation. The user, in pondering the same question, is interested to confirm that the developer intends to deliver a particular designed work situation. Once more, user and developer are interested in comparing their own respective models of the object of the development activity with those of their respective coparticipants.

However, as noted above, the envisioned work situation is an abstraction which does not yet exist. What the user and developer can actually view and manipulate and alter is the task model, the external shared model of the envisioned work situation. So, once again, the external shared model is a resource directly accessible to the participants in a way in which the object of the development activity is not. Thus, establishing a shared understanding of the external shared model, or, at least, noting where understandings differ, is again easier than establishing a shared understanding of the envisioned work situation *per se*. Participants tend to rely heavily on references to the external shared model when their purpose is to establish a shared understanding of an aspect of the envisioned work situation. A more detailed analysis of the process of this type of interaction is taken up in section 4.1.3 below.

Similarly, in the cooperative development activity of software designing, both user and developer have a hand in creating the envisioned software system which is the object of the development activity and they may alter its design as they work. Once again, the participants want to have understandings of the object of the current development activity (in this case, the envisioned software) which, for the purposes of the current development activity, are comprehensive, accurate and consistent. For this development activity, in wondering, 'Are we talking about the same thing?', the developer is interested to confirm that the user is agreeing to a particular designed software system. The user, in pondering the same question, is interested to confirm that the developer intends to deliver a particular designed software system. Yet again, user and developer are interested in comparing their own respective models of the object of the development activity with those of their respective coparticipants.

Here, the envisioned software system is an abstraction which does not yet exist. What the user and developer can actually view and manipulate and concretely alter is the prototype, the external shared model of the envisioned software system. Once again, the external shared model is a resource directly accessible to the participants in a way in which the object of the development activity is not. Thus, establishing a shared understanding of the external shared model, or, at least, noting where understandings differ, is again easier than establishing a shared understanding of the abstract software system *per se*. Again, participants tend to rely heavily on references to the external shared model when their purpose is to establish a shared understanding of an aspect of the envisioned software system.

In the following example transcript, the user has difficulty in constructing a part of the paper prototype because he is unsure of the relationship between the spatial

proportions of the paper prototype and those of the envisioned software system (see Figure 4.12). There is a conflict between the user's internal model of the external shared model and the user's internal model of the envisioned software system. An analysis of the processes involved in resolving such conflicts between models is taken up in section 4.1.3 below.

CS71112

D: If we take this to be the form [*gesturing to sheet of paper*], is there any particular ordering, left, right, up, down, you'd like to see these elements.

U: I don't know. Perhaps customer details at the top.

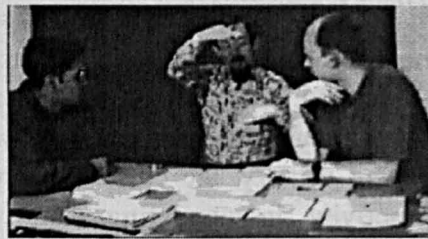
D: Would you like to place it wherever you'd like it?

[*User places Post-It note to represent Customer Details field.*]

D: Top centre?

U: Well, I don't really know whether it's going to be tall and narrow or broad and thin.

Figure 4.12: User's difficulty with paper prototype dimensions



This type of difficulty for the user is not uncommon in working with paper prototypes [Rudd, Stern and Isensee, 1996]. However, within prototyping work with software prototypes, so-called high-fidelity prototyping, the prototype itself often is seen as the envisioned software system [Rettig, 1994; Rudd, Stern and Isensee, 1996]. The interaction between user and evaluator around the prototype is seen as being an interaction about the envisioned software system. The analysis in this chapter illuminates the distinctions between the prototype as a concrete model of the envisioned software system and the system as a still unrealised abstraction. Also revealed are the subtle distinctions between participants' interaction around developing understandings of, on the one hand, the prototype and, on the other hand, the envisioned software system of which the prototype is a model. One advantage of low fidelity paper prototyping over high fidelity prototyping is that those working with the prototype can more easily distinguish between their models of the envisioned system and their models of the prototype representation of the system. Ehn and Kyng [1991, p.192] note that 'one of the reasons for the effectiveness of cardboard mock-ups is that nobody confuses them with the product, the future computer system'.

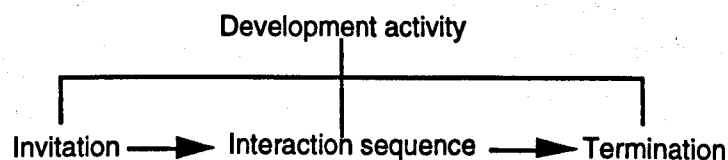
In the activity of requirements analysis, both user and developer have a hand in determining the requirements which are the object of the development activity. Again, they may alter these requirements as they work. Once again, the participants want to have understandings of the object of the current development activity (in this case, the requirements) which, for the purposes of the current development activity, are comprehensive, accurate and consistent. For this development activity, in wondering, 'Are we talking about the same thing?', the developer is interested to confirm her understanding of the user's requirements and that the user adequately understands any technical constraints. The user, in pondering the same question, is interested to confirm that the developer accurately understands the user's requirements. Yet again, user and developer are interested in comparing their own respective models of the object of the development activity with those of their respective coparticipants.

For the development activity of requirements analysis in the studied projects, there was no explicit external shared model, that is, there was no concrete requirements specification which the participants negotiated. Interestingly, the user and developer once again resorted to external shared models which were directly accessible as a resource to help establish shared understandings of the abstract object of the development activity, the requirements. As described above, participants relied heavily on references to the external shared model, task model or prototype, when their purpose ostensibly was to establish a shared understanding of an aspect of the requirements. It seems that using an external shared model, even one which is not intended directly to support the current development activity, is easier than establishing a shared understanding of an abstraction.

4.1.3 Process revisited

The more complex and detailed account of models and relations presented in the latter half of section 4.1.2 demanded a more complex and detailed account of cooperative development process than that presented in section 4.1.1. Specifically, it was necessary to further the analysis of the interaction sequences which formed the central part of the invitation-sequence-termination pattern of development activities (reproduced in Figure 4.13) to take account of the models and relations identified in section 4.1.2.

Figure 4.13: Simple structure of development work activities



Section 4.1.1 described how the processes in Figure 4.14 very often utilised the external shared model as a central resource. Thus, the invitation to an interaction sequence was usually accompanied by an explicit reference to a specific aspect of

the external shared model and the sequence almost always was terminated with an explicit reference to the external shared model. Section 4.1.1 also noted that the central interaction sequence often involved multiple references to the external shared model, without analysing in detail the processes involved in this interaction sequence. Section 4.1.2 presented an account of user-developer interaction in software development activities in terms of models and relations. This section, then, draws together the analyses of the previous two sections by looking in detail at how the processes identified in 4.1.1 form part of the overall interaction with the models and relations of 4.1.2.

Returning to the video record with this analytic perspective revealed complex processes of model construction and use. As noted above, the external, shared model frequently was used to support the invitation to an interaction sequence. An invitation from one participant (call her *A*) might be to provide information where it was missing or validation of extant information represented in the model. The central interaction sequence would then begin with the coparticipant (call her *B*) ostensibly accepting the invitation by contributing a piece of information.

However, the interaction sequence often possessed a complex, recursive structure. Take the seemingly straightforward example, from the development activity of current work situation analysis, of *A* pointing to a hitherto unresolved node of a task model and asking, 'What happens here?'. The information presented by *B* in response serves two immediate functions. It says something about the target work situation and it says something about the external, shared model's representation of that target work situation.

A may first consider two sorts of questions of the form: 'did *B* really say that?' and 'did *B* really mean that?' (see [Clark and Schaefer, 1987a] and section 5.1.1 of chapter five). Consideration of these questions, and any pursuant interaction, establishes that *A* has heard and understood *B*'s meaning; that *A* has not, for example, simply misheard or misconstrued *B*'s words.

A already has versions of the various internal models discussed in section 4.1.2. Thus, *A*, having received *B*'s contribution, compares the information which it conveys with what is represented in *A*'s internal models. Initially *A* takes the sense of *B*'s contribution (having clarified any simple mishearing, for example) as *prima facie* representative of *B*'s internal models: as discussed in section 4.1.2, *A* infers *B*'s internal models from information gleaned through their interaction. Thus, *A* may consider whether or not the information just received is consistent with *A*'s internal model of *B*'s internal model of the external shared model. *A* may also consider whether or not the information just received is consistent with *A*'s internal model of *B*'s internal model of the target work situation.

Consideration of these two questions, and any pursuant interaction, is once again aimed at establishing for *A* whether or not *B* is 'making sense'. However, in distinction to *A*'s previous questions, *A* has received and understood the meaning

of *B*'s contribution and is now attempting to establish whether or not that meaning is consistent with what *A* believes to be *B*'s current understandings of the task model and work situation under analysis.

Having established that *B*'s contribution makes sense within the terms considered so far, *A*'s next questions may be to consider whether or not the information just received is consistent with *A*'s internal model of the external shared model, the task model, and to consider whether or not the information just received is consistent with *A*'s internal model of the target work situation.

In the case of each question considered, confirmation of consistency with the relevant model may allow the interaction sequence quickly to flow to a smooth conclusion. Thus, for example, a shortest path through invitation, interaction sequence and termination is illustrated in the following exchange. The developer invites the user to provide information. The user provides it. The developer signals assimilation of it into his internal models (with 'Right') and also publicly assimilates it into the shared external model.

CS21431

D: And do they [*gestures to TM role*] give you the details of of what the fault is?

U: They would they would actually they would actually give me the hard copy if I were there in their room. If not, they'll get some details from the from reception. They'll phone back.

D: Right. [*writes on TM*]

However, it may be the case that the information just received clashes with an aspect of one or more of the recipient's current internal models. That is, a participant may have received from a coparticipant a contribution, heard and understood as the contributor intended, which is inconsistent with one or more of the participant's internal models of

- the object of the current development activity;
- the external shared model of that object;
- the coparticipant's internal model of the object of the current development activity; and
- the coparticipant's internal model of the external shared model of that object.

In the case of such a clash, the new contribution cannot immediately be assimilated into the recipient's understanding. A process of negotiation ensues in which the participants attempt to clarify their respective understandings and possible reasons for the mismatch. Participants will often attempt also to reconcile the mismatch and thereby to establish a shared understanding. However, as described above, participants may not negotiate to a decision point, but may merely establish their

respective positions more or less clearly and without agreement on a shared understanding.

For example, in the previous transcript of participants discussing the meaning of 'registration', agreement was not reached. This discussion resurfaced several times later in the meeting as the meaning remained unagreed. In the following extract, the continuing confusion is promoted because the developer has assimilated the user's definition of 'registration' into his internal model of the user's internal model of the work situation, although he has not managed to reconcile this model with his own internal model of the work situation. The developer, therefore, continues in the interaction accepting that his coparticipant and he have different understandings of this aspect of the work situation. Unfortunately, he has not made this explicit and the user continues to try to establish a single shared definition of the term.

CI19751

Developer: The confusion arises when you use cross-reference as a kind of verb, as the link that is constructed between the two and that cross-reference which is used to represent all kinds.

User: Register means

Developer: The interesting question there is that how would an index refer to a link. For example how would you refer to

User: If I am registering a statement to an author I would call that registration.

Developer: How would you refer to the link between that message and and that author?

User: Registration. It's done automatically. At registration all I have done is say that statement was made by Eamonn and I have registered that statement to him and given it a unique number. It is registration. There is no other name.

Developer: So if you look at the statement index of the document

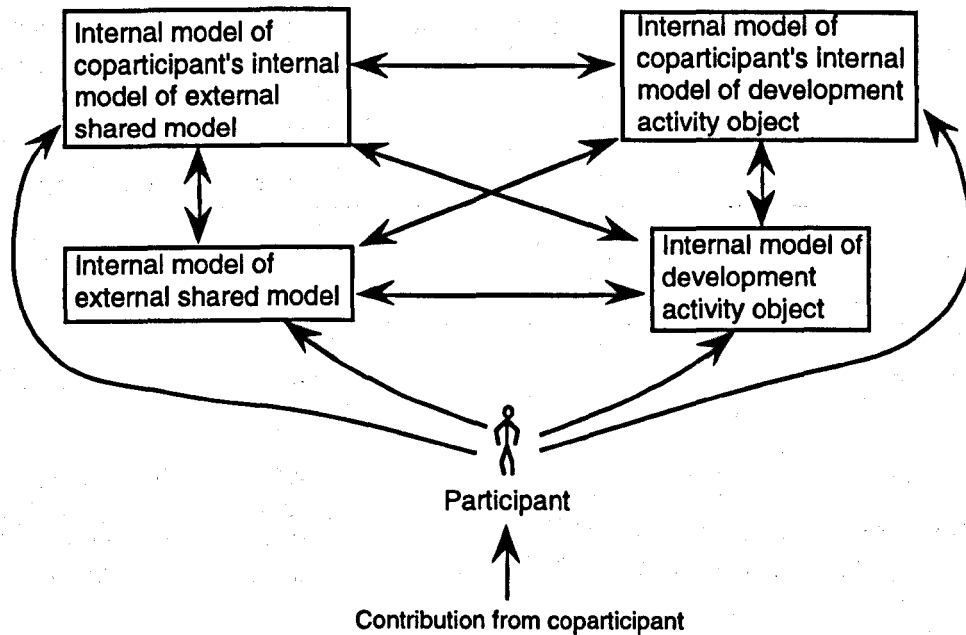
User: It shows statement one, subject, Eamonn or it will say nominal one, Eamonn, subject, statement, whatever. Just for that subject which indicates registration.

Developer: That registration stroke subject link is actually listed

User: You will not give in will you? Registration.

A contribution may mismatch *a priori* with more than one internal model. In addition, a mismatch between a contribution and one internal model may reverberate around other internal models. Figure 4.14 illustrates that a single contribution may initially be checked by its recipient against any of her internal models and reconciling a mismatch by assimilating the contribution into an internal model may have repercussions in creating further mismatches between the internal model, altered by its assimilation of the contribution, and other internal models.

Figure 4.14: Model mismatch monitoring



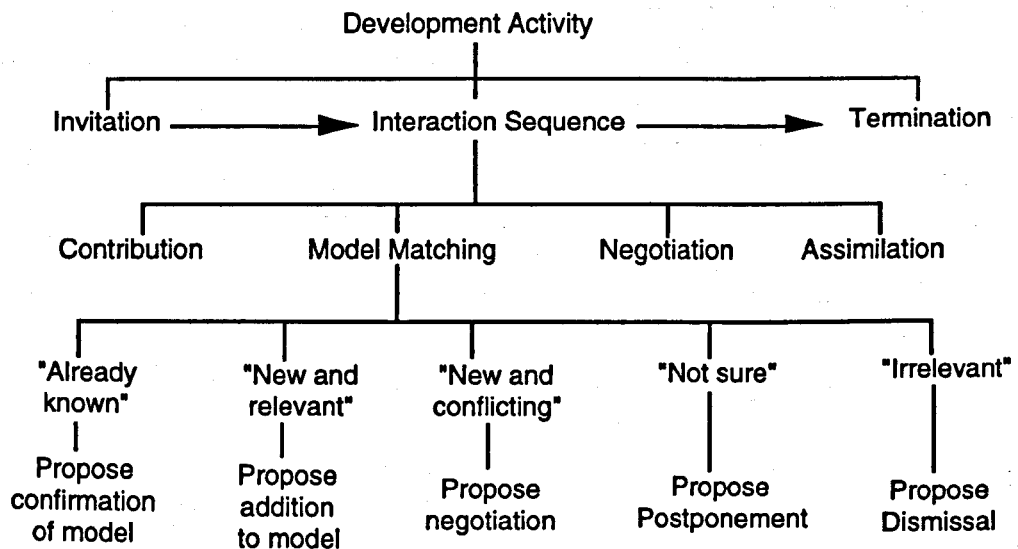
For example, consider that participant *A* believes proposition *p* to hold

- in the object of the development activity (1)
- in the external shared model (2)
- in participant *B*'s model of the object of the development activity (3)
- in participant *B*'s model of the external shared model (4).

Participant *B* then makes a contribution that proposition *q* holds in the external shared model, rather than *p*. For *A*, there is a mismatch between this contribution and her belief (4). Reconciling this mismatch creates a mismatch between (4) and (2) and between (4) and (3). Reconciling these mismatches creates a mismatch between (3) and (1), between (2) and (1) and between (4) and (1).

Thus, a single contribution may institute a complex, recursive web of mismatch monitoring, negotiation amongst the participants and assimilation and consolidation of understandings. The notion of 'consistency' between contribution and model takes several forms. Figure 4.15 illustrates the common types of mismatch which occurred in the interactions between user and developer, and typical responses to those mismatches which were proposed.

Figure 4.15: Interaction processes within development activities



A contribution may be matched against a particular internal model and be found to match the existing content of the model. That is, the informational content of the contribution may coincide with the recipient's current understanding embodied in that internal model. In this case, the recipient may decide that the contribution is 'already known' and simply propose confirmation of her internal model. This proposal may efficiently be made merely by the recipient's moving on to another topic. Or, more explicitly, the recipient may give a simple backchannel response such as 'OK'. Still more explicitly, the recipient may verbally propose confirmation with a phrase such as 'I've got that' or 'I know that'. If the contributor accepts the proposal of confirmation, again with or without an explicit verbalisation, that interaction sequence may be terminated. Again, the contributor's acceptance may be tacit and unspoken or may be a brief backchannel. Occasionally, the contributor may refuse to accept the recipient's proposal of acceptance and, in this case, an explicit verbalisation from the contributor generally seeks negotiation. This verbalisation often takes the form of a recursive invitation to begin an interaction sequence.

For example, in the following exchange the developer D wants to make a contribution about what happens to 'a pended thing' *after* it has been pended. The user U mistakes D to be stating what happens to it in the act of pending it - with which U is already familiar. So, D first begins to make the contribution (1). U interrupts, proposing confirmation of what he took to be D's contribution and verbalising what he assumed as the rest of the interrupted contribution (2). But D refuses to accept U's proposal and makes a further contribution to clarify the situation (3). U proposes confirmation of this contribution (4) and they each agree to this (5) and (6).

CI20035

- (1) D: If it's a pended thing, right, then it's a simple thing of
- (2) U: Of putting a bring up date on it.
- (3) D: Uhhh. Well. It's already got a. When you pend it, you set a bring up date on it.
- (4) U: That's right.
- (5) D: Right.
- (6) U: Yeah.

A contribution may be matched against a particular internal model and be found to provide information additional to the existing content of the model. In this case, the recipient may decide that the contribution is 'new and relevant' and propose the addition of its informational content to her internal model. Assimilation of the contribution into the model may be preceded by negotiation amongst the participants to clarify the contribution and current understandings. Also, as noted above, this assimilation may create mismatches with other internal models, while the original contribution may itself have mismatched with other internal models. Again, negotiation is likely to be called for, producing invitations to fresh interaction sequences within the original one. In the following example, developer and user are working on a selection point in a task model. The developer initiates an interaction sequence on which features of the problem result in which task branch being selected. The user immediately initiates a 'nested' interaction sequence clarifying one of the options. This nested sequence is taken up and terminated (with 'Ahh. Right') before the original interaction sequence continues after this extract.

CS14406

- D: So we have got three branches and you do one of three things depending on, what do you think? Ahmmm. So the three things are, if it is solvable you solve it.
- U: Yes, there is a sort of rider to that. It is perfectly possible that when you solve a problem there may be consequences from that that mean that you still have to refer them to one of the other people to deal with something that arises from the solution as it were.
- D: Right. Something that you can't answer or something that the user may find as an additional?
- U: Well the most frequent one that I principally have in mind is that you find a solution and the solution is they need to be registered to use another computer so you pass them on to liaison. For example.
- D: Ahh. Right.

A contribution may be matched against a particular internal model and be found to provide information which conflicts with the existing content of the model. In this case, the recipient may decide that the contribution is 'new and conflicting' and propose negotiation with the contributor to explore the conflict and establish updated understandings. Once again, this entails recursive invitations to interaction sequences and the negotiation of potential conflicts with other internal models of all the participants. In the following extract, user and developer are discussing the

design of the envisioned software's user interface. At the start of the extract, the user makes a contribution which conflicts with the developer's internal models of the software design and the previously established understandings of it. It immediately becomes clear after only one turn each in this exchange that user and developer have conflicting internal models of the design and of each other's models of the design. The ensuing interaction in the extract does little to establish a shared understanding and it took considerably more interaction before agreement was reached.

CS70719

U: Well, history is sort of part of the problem answer, isn't it?

D: Yeah. Uh. Right. Now. Yeah, well, the way we dealt with it the other day, the history was separate. Is that something that you'd rather

U: No, sorry, I mean, you actually drew up on the whiteboard, I thought that, um, certainly for the history of who had been dealing with things, um that would be like a sort of sequence of pages as it were of which what you would actually normally see on the screen in front of you would just be the most recent of the pages which would be the person that was currently dealing with the problem.

D: Yeah.

U: And I thought we said that when the network support people were describing what they wanted, a history of how the answer was arrived at, that that would be a similar thing. That each time you had a session of working on the problem, you could fill in some details of what you did during that session which would form a page of the screen of dealing with the problem. Then the answer would simply be the most recent of those pages.

D: Right.

U: And presumably the details of what the original question was would be the first of those pages. Or perhaps you'd need that separate. I'm not sure.

D: Right. But we did agree that the history was separate from the main form, did we?

U: Well, separate in the sense that it would be a subform in the main form.

D: Right.

U: Yeah.

D: So the main form itself is, sort of, these three elements? [*Pointing to prototype*]

U: Aah. Yes, yes.

D: Right.

U: I'm not clear that these three elements don't include the history element.

D: [*laughing*] Right. I thought we'd established the other day that the history was separate. No?

U: OK. Depends what you mean by separate.

D: Separately presented. If we're looking at a form on which you want in the first instance to have presented the main details of the query you're working on cos that form's to represent that query

U: Yes.

D: I thought that we'd agreed then that you could access that

It was noted above that participants may fail to reach agreement on a shared understanding and may continue with different understandings of a particular issue. As in the case considered above, this is often after a period of negotiation has produced no agreement and the participants choose to move on. However, it may also be the case that an initial contribution is matched against a particular internal model and the recipient cannot decide how to treat the contribution or does not wish to negotiate at that time. The recipient may, therefore, be 'not sure' and may propose postponement of negotiation on that contribution. The contributor may not accept postponement and may propose immediate negotiation, with the by now familiar recursive invitation to an interaction sequence. If the proposal for postponement is accepted, an explicit, public marker is often created to flag that the topic is to be negotiated at some future point. For example:

C110910

D: [*writes on TM*] I've put a little note here [*points to TM*] about early statements because it's something that persists.

The participants may utilise corresponding internal 'flags' on their internal models - e.g. a conflict between A's model of a work situation and A's model of B's model of the work situation may be flagged (internally and/or externally) as, say, 'known conflict'. In the following extract, D is writing on the task model a list of issues to which the development work should return later, i.e. a list of external flags. In the last line, D makes a remark *sotto voce* which may be evidence of his also creating an internal flag on defining 'resource and cost management'.

C123306

D: What about resource management?

U: Yes.

D: In general

U: Yeah. Definitely.

D: Resource and cost management.

U: Resources. Yeah, that would come down to him.

D: We'll have to define what that means some time.

Since the interaction is purposeful, if a mismatch occurs which does not impact on the current purpose, the recipient may dismiss the contribution as irrelevant. Once again, acceptance by the contributor of the recipient's proposal may lead immediately to termination of the current interaction sequence, whereas non acceptance may lead to further negotiation before termination. In the following example, the developers are interested in what facilities the software should provide to support supervision of people performing a particular role, indexer, in the envisioned work situation. They initially approach this question through

postulating a supervisor role. The user dismisses this as irrelevant because there is (and is intended to be) no such role. Eventually, D1 rephrases the question and gets a response.

C131131

D1: If for example the indexing supervisor

U: Right there is no specific role for an indexing supervisor so forget it.

D2: Officially, there's not. But you've just said that there is

U: Oh, we're getting semantics now. I can dream up all sorts of jobs if you want the tea trolley and all sorts you know but I just don't see what you're getting at.

D1: Well, I mean, a simple question then. If you were supervising the indexers, right?

U: Yes.

D1: Uh what kind of things would you want from the system over and above what you you

U: Auditing facilities.

Figure 4.15 provides an illustration of potential interactions, not a flowchart of an algorithm. There are many potential branches and paths through a particular interaction sequence. Overall, however, a recursive structure may be observed, with the progression through invitation, sequence and termination potentially encompassing many complex recursions through other invitation, sequence, termination patterns. Once again, the enthusiastic reader is invited to review previous examples of transcripts with the current analytic perspective.

The cycle of interaction analysis to this point had identified the joint construction and verification of shared models or mutual understandings as the characteristic goal of cooperative development activities and had begun to propose an explanation of how the participants achieved this goal. The analysis next moved on to a comparison of the by now coherent analytic perspective with existing theory from related disciplines.

4.2 The construction of shared understandings

When users and developers meet they generally come from very different professional backgrounds. Developers come from the community of software developers while users come from the community of whatever work domain is the target of the systems development effort. Thus, there is a 'community gap' between user and developer. This entails a gap in such attributes as knowledge, beliefs, assumptions, skills and interests. Part of the work of cooperative development must be in bridging this gap, in allowing user and developer to share at least some of their knowledge, assumptions and beliefs with respect to the user's work and the software system which is to support it.

Much theoretical work has been done on the construction and use of shared knowledge, beliefs and assumptions in fields such as philosophy, linguistics, psychology and sociology. Much of this work, however, has focused on the

problem of language comprehension and use, especially in the areas of semantics and pragmatics. For several examples of theories of language use as dependent on the construction of shared understandings or 'mutual knowledge', see [Smith, 1982]. Stalnaker [1978] coined the term *common ground* to include not just shared or mutual knowledge, but also shared beliefs and assumptions. Clark (e.g. [1996]) adopts this term, emphasising that common ground 'explicitly covers mutual knowledge, mutual beliefs, mutual assumptions and other mutual attitudes' [Clark, 1992, p.6].

Clark and colleagues (see Clark [1992]) present an extensive body of work on the construction of common ground in language use. However, Clark (e.g. [1996, p.29]) identifies language use, and its reliance on common ground, as just one example of the more general notion of a *joint activity*. Another example of joint activity favoured by Clark (see, e.g., [Clark, 1996; Clark and Carlson, 1982]) is that of two or more people performing a piece of music. Clark and Carlson [1982] describe a scenario of two violinists, Perlman and Zuckerman, playing a duet.

The joint act of initiating the first note is complicated enough. Imagine that Perlman has been practising his gesture to start the first note, and now he wants the gesture to be taken for real. When he gestures this time, he must believe that Zuckerman will take the gesture for real (and not just practice), since Zuckerman won't otherwise play the first note, and their joint act will fail. However, he recognises that Zuckerman will also not play if Zuckerman believes that Perlman believes himself to be still practising, since in that case *Perlman* won't play. Perlman must believe that Zuckerman believes that he, Perlman, is taking the gesture for real. This, however, is still not enough. What if Zuckerman believes that he, Perlman, believes Zuckerman still thinks he is practising? Zuckerman won't play in that case either, since he wouldn't expect Perlman to play. That is, Perlman must believe that Zuckerman believes that he, Perlman, believes that Zuckerman believes that this time he is gesturing for real.

[Clark and Carlson, 1982, p.3]

Clark and Carlson argue that, in principle, Perlman should continue this line of reasoning *ad infinitum*. Lewis [1969] has shown that this form of infinite regress is essential to any joint activity involving the development of what he calls common knowledge. Similarly, Schiffer [1972] proposed a definition of mutual knowledge as:

A and B mutually know that p =_{def}

(1) A knows that p .

(1') B knows that p .

(2) A knows that B knows that p .

(2') B knows that A knows that p .

(3) A knows that B knows that A knows that p .

(3') B knows that A knows that B knows that p .

etc ad infinitum.

Clark and Marshall [1981] used an examination of definite reference in conversation to explore the concept of mutual knowledge. For a definite reference to be felicitous, that is a definite reference for which the speaker has 'good reason to believe that [the listener] won't get the wrong referent or have to ask for clarification' [Clark and Marshall 1981, p.11], the speaker must be certain that once she has made her reference the speaker and listener enjoy mutual knowledge of the identity of the referent.

From the above definition of mutual knowledge, this certainty entails the checking of an infinite number of conditions. Thus, we have what Clark and Marshall [1981] call the mutual knowledge paradox: in everyday interaction people make felicitous definite references without difficulty and in a short, finite time, whereas to do so appears to require an infinite checking process. This paradox rests on four assumptions:

1. one attempts to make definite references which are felicitous, i.e. for which the addressee won't get the wrong referent or have to ask for clarification;
2. to be sure of making a felicitous reference one must be assured of an infinity of conditions;
3. checking each condition takes finite processing time and capacity;
4. one ordinarily makes definite references in a finite, short time.

Clark and Marshall [1981] note that the third assumption 'states a processing assumption that is common to almost every psychological model for such a process - that an infinite number of mental operations cannot be carried out in a finite amount of time [and that the fourth assumption] states the obvious empirical observation that when people refer to things, they don't take much time in doing it' [1981, p.15]. Thus, in attempting to resolve the paradox, Clark and Marshall [1981] argue that 'the inevitable conclusion is that one or both of [the first two] assumptions must be weakened and the infinite process replaced by finite heuristics' [1981, p.27].

Clark and Marshall [1981] describe two families of heuristics for assessing mutual knowledge: truncation heuristics which weaken assumption one and copresence heuristics which resolve the problems entailed by assumption two.

Using truncation heuristics, one checks only a limited number of conditions. One may adopt a progressive checking procedure, checking condition one and the next few higher order conditions. One may adopt a selective checking procedure,

checking only a single, higher order condition. These two strategies produce the same results for strict mutual *knowledge* but not necessarily for mutual beliefs or mutual assumptions.

Clark and Marshall [1981, p.29] argue that 'neither of these procedures guarantees a felicitous definite reference because both lead to something less than full mutual knowledge', although they concede that truncation heuristics can guarantee felicitous definite reference given certain conditions. Other objections to truncation heuristics are that (i) it is implausible that the sometimes extremely complicated condition checking ordinarily is carried out; and (ii) the evidence required to check the conditions suggests a more basic family of heuristics.

In order to check a condition *I know that she knows that ...* one requires evidence and ancillary assumptions from which one may infer the condition. Clark and Marshall [1981] argue that the best evidence to support such an inference is the copresence of the speaker, the addressee and the object of the proposition. Moreover, such evidence supports the inference of all the other conditions to infinity. Therefore, rather than checking an infinite or truncated list of conditions, one need only check the reliability of the evidence of copresence. Lewis [1969] presents an induction schema by which one may infer what he calls common knowledge from a set *A* of evidence and ancillary assumptions.

Let us say that it is common knowledge in a population *P* that ___ if and only if some state of affairs *A* holds such that:

- (1) Everyone in *P* has reason to believe that *A* holds.
- (2) *A* indicates to everyone in *P* that everyone in *P* has reason to believe that *A* holds.
- (3) *A* indicates to everyone in *P* that ___.

We can call any such state of affairs *A* a basis for common knowledge in *P* that ___. *A* provides the members of *P* with part of what they need to form expectations of arbitrarily high order, regarding sequences of members of *P*, that ___. The part it gives them is the part peculiar to the content ___. The rest of what they need is what they need to form any higher-order expectations in the way we are considering: mutual ascription of some common inductive standards and background information, rationality, mutual ascription of rationality and so on.

[Lewis, 1969, pp.56-7]

Clark and Marshall [1981] adopt this induction schema and elaborate on the nature of the evidence and the assumptions which comprise *A*, the basis or grounds (relabelled *G* by Clark and Marshall) for the mutual knowledge that ___ (relabelled *p* by Clark and Marshall). 'The point of this schema is that [the participants] don't have to confirm any of the infinity of conditions in mutual knowledge at all. They need only be confident that they have a proper basis *G*, grounds that satisfy all three

requirements of the induction schema. With these grounds, [the participants] tacitly realize, so to speak, that they could confirm the infinity of conditions as far down the list as they wanted to go. Because they could do so in principle, they need not do so in fact' [Clark and Marshall, 1981, p.34].

Evidence may be of various types. Physical copresence of person *A* and person *B* and the object of proposition *p* is strong evidence for *A* of mutual knowledge with *B* that *p*. This type of evidence is usually accompanied by three types of assumption made by *A*. First, that *B* is looking at *A* and at the object of *p* simultaneously with *A*'s looking at the object and at *B*. Secondly, that *B* is attending to both *A* and the object of *p*. Thirdly, that *B* is rational and, therefore, drawing the same kind of conclusions as *A*.

A second type of evidence is linguistic copresence. *A* makes an object linguistically copresent with *A* and *B* when she mentions it in speech. References may then be made to the object. In this case, to achieve mutual knowledge with *B* of a proposition about the referenced object, *A* must assume, in addition to the assumptions of simultaneity, attention and rationality, that *B* will understand *A*'s reference to be to an object 'that she is sincerely positing in some world' [Clark and Marshall, 1981, p.41].

A third type of evidence is community membership or, rather, mutual knowledge of community membership. Clark and Marshall's [1981] account of this type of evidence is less than clear. First, they do not make explicit how the participants acquire this evidence, that is, how they initially come mutually to know that they belong to the same community. Secondly, Clark and Marshall, having just suggested that one's belief in mutual knowledge of community membership provides the evidential part of *G*, then [1981, p.38] list it as one of the two main *assumptions* of *G*. In any case, the other main assumption required to support community membership as grounds for mutual knowledge of a proposition is the assumption of universality of knowledge: that all, or almost all, members of the community in question could be expected to know the proposition.

Clark and Marshall [1981, p.41] propose that 'very often mutual knowledge is established by a combination of physical or linguistic copresence and mutual knowledge based on community membership'. Heuristics may be used by the participants to determine copresence or community membership and so to provide the evidence for *G*. 'People assess mutual knowledge by use of the copresence heuristics. They search memory for evidence that they, their listeners and the object they are referring to have been 'openly present together' physically, linguistically or indirectly. Or they search memory for evidence that the object is universally known within a community they and their listeners mutually know they belong to. With such evidence they can infer mutual knowledge directly by means of an induction schema. There is no need to assess an infinite number of conditions' [Clark and Marshall 1981, p.58].

Clark [1996] refers to Lewis' [1969] representation of common knowledge as a 'shared basis representation of common ground', denoted as 'CG-shared' [Clark, 1996, p.94]. The shared basis is the evidence and assumptions which comprise G (relabelled again as b in [Clark, 1996]). Clark argues that, having established CG-shared, participants are in a position to derive 'a second representation that eliminates any mention of the shared basis'. This second representation is in fact Harman's [1977] reflexive representation of mutual knowledge:

p is common ground for members of C if and only if:

(i) the members of C have information that p and that i .

Again, Clark [1996] uses the more inclusive 'common ground' rather than 'mutual knowledge', denoting this representation as 'CG-reflexive'. Clark [1996, p.95] suggests that Schiffer's [1972] definition of mutual knowledge, denoted as 'CG-iterated' and also defined without mention of shared basis b , may be derived from CG-reflexive. Clark [1996, pp.95-6] argues that 'because it requires an infinitely large mental capacity ... CG-iterated is inconceivable as a mental representation', allows CG-reflexive as a possible representation, but proposes that CG-shared is 'the basic representation' of common ground.

CG-shared, Clark's [1996] 'basic representation' of common ground, depends in each instance on the establishment of b , the shared basis for each individual piece of common ground. 'When it comes to coordinating on a joint action, people cannot rely on just any information they have about each other. They must establish just the right piece of common ground, and that depends on them finding a shared basis for that piece' [Clark, 1996, p.99]. The problem, then, in constructing common ground is in finding the evidence and auxiliary assumptions to establish such shared bases.

Clark [1996] proposes two main types of evidence for b in constructing common ground: evidence of membership of cultural communities and evidence of joint perceptual experiences and joint actions. These correspond respectively to community membership and copresence in [Clark and Marshall, 1981]. Clark [1996] argues that a basis b comprised of evidence of membership of cultural communities and associated assumptions (such as universality of particular knowledge within the community) leads to 'communal common ground', while a basis b comprised of evidence of joint perceptual experiences and joint actions and associated assumptions (such as rationality) leads to 'personal common ground'.

Personal common ground is specific common ground established amongst the members of the community who have shared the joint experience. However, their assumptions for b of rationality and shared inductive standards depend ultimately on their previously established communal common ground. As Clark [1996, p.113] notes, 'perceptual events are never dealt with in the raw. They are always perceived *qua d*, where d is a description that depends on communal common ground'. Communal common ground typically is very wide, not only established amongst

the people who have formed *b* together, but assumed by them to be shared with other, absent members of the same cultural community.

4.3 Cooperative development and Common Ground

A comparison of the extant theoretical work on common ground with the empirical work of the preceding sections suggested a gap in the literature on common ground. Analysis of the cooperative development work suggested that participants attempted to establish two distinct types of common ground in their joint activities: common ground about the objects of the development activities, call it CG(O), and common ground about the immediate and present development situation, call it CG(P). An example of CG(O) is the participants' shared belief that the user will perform task *t* in an envisioned work situation, while an example of CG(P) is the participants' agreement to represent task *t* with a particular notation in the development activity of designing an envisioned work situation.

CG(O) is the major subject of the empirical analysis presented above: user and developer attempting to achieve shared understandings of the current work situation, the requirements, an envisioned work situation and a software design with little real shared experience of these. (For an analysis more focused on CG(P) in terms of the representations used in cooperative development work, see [O'Neill, Johnson and Johnson, 1997; O'Neill, Johnson and Johnson, under review].) However, CG(P), in the terminology introduced here, is the only form of common ground addressed in the literature. For example, Clark [1996] argues that joint activities result in the accumulation by the participants of common ground *about that joint activity*. 'If joint activities are cumulative, what accumulates? I will argue that it is the common ground of the participants about that activity - the knowledge, beliefs and suppositions they believe they share about the activity' Clark [1996, p.38]. Again, 'We can say that the common ground of the participants *about the conversation* accumulates in the course of that conversation. ... Accumulation of common ground occurs in all joint activities' [Clark, 1996, p.39; emphasis added].

Clark [1996] proposes that common ground has three components

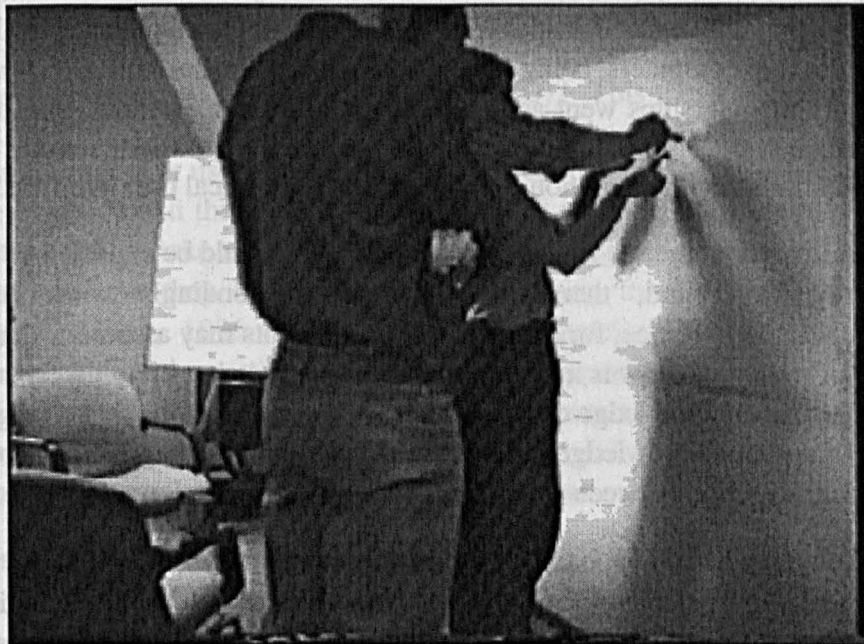
- initial common ground: the background facts, assumptions and beliefs of the participants at the start of the joint activity;
- current state of the joint activity: what the participants presupposed to be the state of the activity at the moment;
- public events so far: events presupposed by the participants to have led up to the current state.

Clark [1996] notes that in joint activities such as playing chess the current state of the activity is represented in an external concrete form: in the case of chess, the board with its pieces. Furthermore, according to Clark [1996], the physical environment of many joint activities provides a concrete external representation of

the current state of the activity. For example, in the joint activity of buying an item in a shop, the physical layout of counter, cash register, items, server, queue of customers and so on encapsulates a lot of information about the current state of the transaction. This scene is interpreted by the participants 'according to a highly developed understanding of how things work in such transactions' [Clark, 1996, p.46]. This understanding itself is a part of the initial common ground for the activity.

Clark [1996, p.47] argues that 'external representations are especially valuable as a medium for the actions themselves. ... Many joint activities would seem impossible without such representations'. Thus, the activity of playing chess is conducted in part by moving pieces around the board; the activity of buying an item in a shop is conducted in part by manipulating the item, money, a bag and so on; and cooperative software development activity is conducted in part by manipulating the task models, prototypes and so on. The latter external representations were extremely valuable throughout the cooperative development activities as a medium for the participants' actions (see Figure 4.16).

Figure 4.16: User and developer use a task model as a medium for analysis



In the cooperative development activity the external shared model - task model or prototype - may be seen as representing the current state of the activity. Thus, the external shared model represents one of the three components of the participants' common ground - where that common ground is established about their joint activity. This, however, is only part of the story. In the cooperative development activities, the situation is made rather more complex because the external shared model is simultaneously both a representation of the current state of the

development activity - supporting CG(P) - and a representation of the *object* of the development activity - supporting CG(O).

So, in the terms of this work, [Clark, 1996] suggests that CG-shared is used by the participants in cooperative development to establish CG(P). In order to achieve this, the participants rely upon the strength of the evidence and assumptions which comprise the shared basis *b* for CG-shared. Through their joint activities, the participants in cooperative development work build a strong basis for *personal* common ground. In establishing common ground about an aspect of their joint development activity, e.g. task modelling, the participants had the very best evidence for *b*: the physical copresence of each participant and the object of their discourse. For example, in agreeing a change to a task model or prototype, the participants were physically copresent with the task model or prototype. And in agreeing a new form of notation for an aspect of a task model or prototype, the participants could make the putative notation linguistically copresent in proposing it and then immediately physically copresent in using it.

The other source of evidence for *b*, as a basis for *communal* common ground, is shared community membership. Community memberships which the participants, developers and users, shared in relation to their joint activities included, for example, being native English speakers, being more or less familiar with the use of modern desktop computers, being quite well educated and so on. Along with these community memberships went assumptions such as, for example, that they were each familiar with using written notes in English, reading basic hierarchy diagrams and were familiar with the notion of a window in a graphical user interface.

However, the communities which user and developer could be expected to share are necessarily broad and, therefore, provide correspondingly weak bases for communal CG(P). Hence, for example, the participants may assume a shared basic concept of what a window is in a graphical user interface but are unlikely to assume a shared technical knowledge of what elements a window should have. Again, they may assume shared knowledge of how to read a basic tree diagram but not of how to represent selection between subtasks in a diagrammatic task model.

Thus, communal CG(P) based on shared community membership must be complemented by personal CG(P) based on the participants' joint activities in the development work. Given this combination of broad community memberships and intense joint activities, CG-shared may readily be used to construct common ground about their joint activities, that is CG(P). For example, as noted above, an agreement amongst the participants to use a particular notation is a part of their CG(P). Clark [1996] argues that 'an explicit agreement is nothing more than a shared basis *b* for a mutual belief, and it is that shared basis that enables you and me to coordinate in performing a joint action'. The development of agreed notations, conventions, shared vocabularies and so on within the development situation are all part of the evolving CG(P) between user and developer.

Take the problem of representing elements of an envisioned software design. The participants may want to establish as part of their common ground that, for example, green Post-It notes in a paper prototype represent data entry fields in a software interface. That is, in order to work effectively with the prototype, they need to establish the mutual belief that this is so.

Referring again to the problem of how to begin a violin duet, Clark and Carlson [1982] propose that Perlman may say to Zuckerman, 'On my next gesture, let us start'. Zuckerman replies, 'Right' and the participants have explicitly agreed that the next gesture shall be for real. They may then use this agreement as sufficient grounds for the induction schema and, hence, achieve mutual belief that the next gesture is for real.

In the same way, participants in the system development activities may explicitly agree that green Post-It notes in a paper prototype represent data entry fields in a software interface. When they come to work with the prototype, their explicit agreement may serve as the basis for their mutual belief that a green Post-It placed on the paper prototype represents a field. This mutual belief forms part of the CG(P) which allows them to work with the prototype as a representation of the software.

But were the participants in the cooperative development activities also using the induction schema of CG-shared to establish CG(O)? This thesis suggests that they may not have been. CG-shared relies crucially upon the shared basis *b* of evidence and assumptions. For user and developer in the cooperative development activities, both the evidence and the assumptions for CG(O) were typically very weak while 'in practice, people take a proposition to be common ground in a community only when they believe they have a proper shared basis for the proposition in that community' [Clark, 1996, p.96]. Successful development activity, however, continued to oblige the participants to establish common ground about its objects. Hence, in the absence of usable copresence and community membership heuristics, user and developer may have been forced to fall back on a CG-iterated representation and its associated truncation heuristics.

During the projects, developer and user attempted to construct personal common ground about the objects of the development activities, i.e. personal CG(O), through the shared personal experiences of their work together. However, this was undermined in so far as the work was not with the development activity objects *per se*. First, the evidence which they accumulated through their collaboration in the development work was simply evidence of their shared personal experience *in the development situation* (contributing to personal CG(P)) and was not evidence of shared personal experience of the development activity's object. Secondly, the assumptions which accompany and underlie interpretation of the evidence of shared experiences ultimately depended on a communal common ground which did not exist.

For example, in the projects studied, developer and user did not work together in the users' work domain. Hence, they lacked the evidence of joint perceptual experiences and joint actions from which to construct personal CG(O) about the users' work situation. In constructing a task model of that work situation, the participants enjoyed physical copresence with the representing world but not with the represented world [Palmer, 1978]. Thus, they had a solid basis for constructing personal CG(P), as described above, but not for personal CG(O).

The absence of shared professional community membership greatly undermined any shared basis for construction of communal common ground about the objects of the development activities, that is, for communal CG(O). For example, the user could be expected to have a great deal of knowledge about the current work situation. But the developer, as a nonmember of that work community, lacked the basis for establishing such knowledge as communal common ground with the user. User and developer may establish communal common ground on the shared basis of their membership of wider communities, such as native English speakers. However, this type of communal common ground cannot directly contribute to a shared understanding of the development activity objects, that is, again, to CG(O).

A great deal of the participants' efforts went into making the objects of the various development activities linguistically copresent (see section 4.1.2 above). The complexity inherent in what they were trying to make copresent mitigated against these efforts. Furthermore, making the current work situation, requirements, envisioned work situation or envisioned software linguistically copresent depended upon a communal CG(O) which itself was weak. The participants could not assume universality of knowledge or rationality, that is, that user and developer would interpret each other's words in the same way and draw the same conclusions.

Clark and Carlson [1982] suggest that when the grounds for the induction schema are weakened, people continue to use it. In this view, as the evidence and accompanying assumptions become weaker, the strength of the mutual belief becomes weaker, but people continue to use the induction schema to derive that increasingly weak mutual belief. Schober and Clark [1989, p.195] recognise the rarity of full mutual knowledge when they talk about understanding well enough for current purposes: '[An addressee] B's criterion for understanding is the belief that he and [a speaker] A mutually believe he has understood her well enough for current purposes. He can work until he has understood as well as he wants'. Participants in discourse generally do not aim for a guarantee of full mutual knowledge. Reliability of mutual belief, enough for current purposes, is ordinarily sufficient. In the cooperative development activities, the need remained for the participants to construct CG(O) on which they *could* rely for their particular purposes.

The examples used by, for example, Clark and Marshall [1981], and Lewis [1969], to demonstrate the need for the infinite regression delivered by the induction schema

are extremely complex. Sperber and Wilson [1982] argue that faced with such complexity in real life, someone having difficulty grasping another's meaning would either ask for clarification or *would* misunderstand. Sperber and Wilson [1982, p.69] suggest that the only cases where a genuine effort is made to establish mutual knowledge is in legal work, 'where the risk involved in misunderstanding is so great that the [cognitive] cost of reducing it is acceptable'. But software development too is a human activity in which the costs of getting it wrong have been demonstrated to be high. We may assume that the participants in the development work are making genuine efforts to come to a shared understanding of the objects of their activities. And, as noted above, the knowledge which the participants in systems development work are trying to share is also very complex.

Indeed, what we see in the user-developer interaction described in section 4.1 is the participants continually asking for clarification of their CG(O). As noted in section 4.1.3, the repeated pattern of interaction between user and developer typically begins with an invitation to provide information to form a new piece of CG(O) or clarification of an existing piece, while the central interaction sequence consisted of many clarifications of understanding by the participants. A pattern of repeated and recursive clarification was very often apparent within the central interaction sequence whether the sequence had been initiated for elicitation of fresh information or for clarification of an existing understanding.

It is possible that in seeking this clarification, the induction schema of CG-shared having failed to produce mutual belief which *is* strong enough for current purposes, the participants may fall back on CG-iterated and its associated truncation heuristics. The frequent initiation of an interaction sequence with, 'So you're saying *p*', as in the following extract, is an almost literal statement of the condition 'I believe that you believe *p*'. The illocutionary force of this statement is for the recipient to confirm, refute or expand upon *p*, thereby providing the necessary check of this second order condition.

C125400

U: Right. So you're saying if you did a surname search and you found nothing, you should then do a description, say throw a wide open search on males that fit this description.

D: Uhh. Something slightly more than that. What I'm saying is that the, you might have a kind of built in search strategy for searching searching, right?, such that it tries the different criteria in a in an order, right, until it gets uh a reasonable level of hits.

It is interesting to note that the same frequent seeking verifications of understanding was not apparent for elements of CG(P). Typically, the participants agreed a notational device, for example, and got on with using it, with very few subsequent checks that they understood the same thing by the notation. As argued above, the basis *b* for the CG-shared induction schema is generally very strong for CG(P).

One of Clark and Marshall's [1981] objections to the use of truncation heuristics is the argument that it is implausible that the complicated condition checking entailed

by truncating to high levels ordinarily is carried out. However, in the cooperative development situations studied, complex condition checking at high levels was rarely, if ever, apparent. Clark and Marshall [1981] equated Schiffer's [1972] definition of 'mutual knowledge' with 'share_∞ knowledge'. Hence, we have the following definitions:

A and B share₁ knowledge that $p =_{\text{def}}$

(1) A knows that p .

(1') B knows that p .

and

A and B share₂ knowledge that $p =_{\text{def}}$

(1) A knows that p .

(1') B knows that p .

(2) A knows that B knows that p .

(2') B knows that A knows that p .

and so on up to share_∞ knowledge.

Section 4.1.3 of this chapter presented the results of the analysis of the cooperative development work in terms of a two layer account, illustrated in Figure 4.14. The two layer account of Figure 4.14 is equivalent to share₂ knowledge or belief and it was exchanges at this level which accounted for much of the participants' interaction in the cooperative development situation. Take the following example from a paper prototyping session.

CS70947

U: I mean, what I was assuming we were talking about was that you'd have a form, right, that would have various little boxes on it and if you chose to click on one of those boxes it would like flip over to the next in the sequence of things that could appear in that box. As you step through the set of pages that could show up in that box on the kind of front level form, that would be the history.

D: Right.

Here, the object of the development activity is the envisioned software system. The external shared model is a paper prototype of the software system. Consider from the interaction the proposition, let us call it p , that in the implemented software system clicking on a particular box will allow the user to browse the history field. U and D have reached a shared understanding of this proposal for the software system, good enough for their current purposes, when the following conditions hold:

- (1) U believes that p (U's internal model of the software system);
- (1') D believes that p (D's internal model of the software system);
- (2) U believes that D believes that p (U's internal model of D's internal model of the software system);
- (2') D believes that U believes that p (D's internal model of U's internal model of the software system).

This is equivalent to: U and D share₂ knowledge that p . The interaction analysis suggests that much of the interaction in the situations studied occurred in terms of this two-layer account. For example, a mismatch between U's internal model of the software system and U's internal model of D's internal model of the software system is equivalent to a failure of U's check for share₂ knowledge due to a conflict between

U believes that p

and

U believes that D believes that q (where $p \neq q$).

Therefore, U should not ordinarily have to apply the truncation heuristics at any higher level than condition two. If CG-iterated is used to represent the participants' understandings, truncation heuristics could ordinarily be applied at the level of share₂ knowledge where the condition checking is not excessively complex.

This does not provide the certainty of full mutual knowledge (share_∞ knowledge) but can provide the participants with enough assurance of shared understanding to perform their joint activity. Evidence of the absence of share_∞ knowledge is revealed in instances of postponement and returns to earlier unresolved issues.

It is possible, of course, for a participant to require a three layer account if obliged, for example, during the interaction to consider the condition 'I believe that she believes that I believe that p '. From the development situations studied, however, this appears to be a very rare occurrence. In any case, the difficulties imposed by checking complex conditions are likely to encourage strategies for reducing the level of checking required. A potential strategy is to appeal once more to second order condition checking with a version of, 'So you're saying that ...'.

The likely level of truncation required may depend on the number of simultaneously active participants. The addition of a third participant obliges A to consider an extra condition or internal model in the two layer account:

A believes that C believes that p .

(B and C are obliged to consider corresponding conditions.)

Moreover, the addition of C also increases the likelihood of one of the participants having recourse to a three layer account since the interaction may give rise for any participant to any one of the conditions:

'I believe that she believes that I believe that *p*';

'I believe that he believes that I believe that *p*';

'I believe that she believes that he believes that *p*';

'I believe that he believes that she believes that *p*'.

Again, participants are likely to employ strategies for reducing the level of checking required when dealing with multiple simultaneous viewpoints. Nevertheless, if participants are using CG-iterated and truncation heuristics, one to one interaction should result in fewer difficulties in arriving at shared understandings than interaction actively involving more than two participants. There is some evidence of this from the projects studied.

The strategy adopted in both projects for combining multiple viewpoints was for developers to work with one user, constructing both internal and external models of the object of the development activity incorporating that user's viewpoint. Only when common ground with one user had been established were there efforts at integrating the viewpoints of other users. In the CI project, with its heavy reliance on one user, this generally meant incorporating information from other users into the external model developed with the primary user. In the CS project, this meant developing partial task models or prototypes with individual users before collating them and negotiating any conflicts or overlaps with a group of users. This allowed rapid and relatively harmonious development of external shared models. In contrast, when (in the CS project) one large meeting was held with two developers and eight users attempting to construct a task model, little progress was made and some participants felt quite uncomfortable.

Clark and Marshall [1981, p.30] suggest some evidence for the use of truncation heuristics. They conducted trials in which 'subjects appeared to use procedures very much like the truncation heuristics, especially the augmented truncation heuristics. As the scenarios became more complex, they tended to have more difficulty as this analysis would predict. So these heuristics are possible'. Lewis, who first proposed the induction schema, seems to support the conclusion that checking of high level complex conditions should not ordinarily be required when he notes that 'we rarely do have expectations of higher order than, say, fourth. For another thing, any ordinary situation that could justify a high-order expectation would also justify low-order expectations directly, without recourse to nested replications' [Lewis, 1969, p.32]. Also, Clark [1996, p.100] notes, if rather dismissively, that several researchers have argued that people do not ordinarily need to go beyond the first few orders of conditions [Bach and Harnish, 1979; Harder and Kock, 1976].

Lewis [1969, pp.56-7] argues that 'the degrees of rationality we are required to have, to have reason to ascribe, etc, obviously increase quickly [with each higher order condition]. That is why expectations of only the first few orders are actually formed. The generating process stops when the ancillary premises give out. ... A basis for common knowledge generates higher-order expectations with the aid of pre-existing higher order expectations of rationality. ... What cuts off the generation of higher-order expectations is the limited amount of rationality indicated by any basis - not any difficulty in generating higher order expectations of as much rationality as is indicated by a basis'. In other words, weak evidence and assumptions allow only lower order condition checking.

Hence, it is possible that CG-iterated and its associated heuristics may be used by the participants in representing shared understandings of the objects of cooperative development activities. However, it is true that the condition checking required for such a representation may in some complex cases become implausibly complicated. Clark and Marshall's [1981] second objection to truncation heuristics is to question why one should choose to use truncation heuristics when the evidence available to this method could also serve the more certain and potentially less complex heuristics of CG-shared. The answer, as presented above, is that the condition checking described above is done precisely when the evidence (and accompanying assumptions) is *not* strong enough to provide - *via* the induction schema of CG-shared - mutual belief which is strong enough for the participants' present purpose.

Clark and Marshall's [1981] third objection to truncation heuristics is that they cannot lead to full mutual knowledge while CG-shared can. However, this objection loses its force on two grounds. First, Clark and Marshall [1981] acknowledge that people rarely, if ever, achieve full mutual knowledge. Secondly, they acknowledge that while CG-shared often cannot lead to full mutual knowledge CG-iterated at times can.

As noted above, Schober and Clark [1989, p.195] recognise the rarity of full mutual knowledge. Clark [1996] also implicitly recognises this in his adoption of the term 'common ground' in preference to 'mutual knowledge'. Sperber and Wilson [1982, p.69; emphasis in original] criticise the very idea of mutual knowledge as being unattainable: 'it is not just that we do not need to be sure: in fact, we *could not* be sure, since mutual knowledge itself cannot be established with absolute certainty'.

The analysis reported here of cooperative development activities suggests that users and developers ordinarily do not come to full mutual knowledge. Their understandings of the object of their current development activity are rarely, if ever, an exact match. Minneman's [1991] observations on the maintenance of ambiguity in design activities and on negotiating better understandings of other participants' positions without necessarily reaching agreement also suggest that full mutual knowledge is not ordinarily reached by participants in development activities.

Clark and Marshall [1981, pp.29-30] have shown that truncation heuristics can guarantee full mutual knowledge given certain conditions. They propose conditions under which truncation heuristics may be augmented by particular assumptions to guarantee felicitous definite reference. Moreover, short of full mutual knowledge, truncation heuristics can provide a given level of confidence in mutual knowledge. One may, for example, assert share₄ knowledge by checking up to condition (4): 'I know that she knows that I know that she knows that *p*'. Thus, 'on actuarial grounds, if condition (4) holds, it should be highly likely that conditions (5) through infinity hold, too. So, although errors can occur, they should occur rarely and only in complicated situations' [Clark and Marshall, 1981, p.28].

CG-shared, on the other hand, does not guarantee share_∞ knowledge simply from the available evidence. It also depends on assumptions which may or may not be valid, such as the universality of knowledge of a proposition within a given community. 'People hold mutual beliefs with greater or lesser conviction. How strongly they hold a mutual belief depends on the evidence and assumptions it is based on.' [Clark 1992, p.5]. The weaker the evidence and assumptions, the further the participants fall short of mutual knowledge.

Participants cannot use the infinite regress of 'I know that you know that ...'. But they don't need to. Usually all a participant needs to know, in order to get on with their (joint) activity, is 'I believe *p*' and 'I believe that you believe *p*'. Very rarely does one need to go even to the third level ('I believe that you believe that I believe *p*'). If a participant has reason to doubt that her coparticipant believes *p*, the simplest solution is to ignore the doubt and hope the ambiguity will work itself out in the course of the joint activity. Minneman [1991] argues that 'participants in group engineering design practice ... demonstrably use ambiguity as a resource for accomplishing their work. ... Often the participants commit to the ambiguity, sometimes agreeing to disagree, other times depending on some unspecified future circumstance to help in finally resolving the issue, should it arise again'. If the participants must resolve the ambiguity for current purposes, the next simplest solution is to ask, to seek verification of their coparticipant's understanding. Participants' lack of omniscience (or infinite processing capacity) prevents them from directly verifying their coparticipants' understandings, but their communication channels, verbal and non-verbal, allow them to do just that.

4.4 Conclusion

This chapter set out to answer research question one: what is cooperative development? It examined the development activities in which users and developers jointly engaged across the studied projects and has produced at least a partial answer to this question. The analysis presented above covers just a fraction of the vast area which is cooperative software development. Development of shared understandings - of the users' work domain, of requirements, of an envisioned work situation and of an envisioned software system - is one necessary part of the

work, joint and individual, that must be performed in cooperative systems development.

This chapter has drawn on several disciplines in examining the nature of the interaction between user and developer. There are many disciplines and theoretical perspectives other than those used here which might further contribute to an understanding of user-developer interaction. However, this chapter has provided the beginnings of a theoretical account of cooperative development work on which such further work may draw.

Whilst experimental evidence is not yet available to test this emerging theory of user-developer interaction, making progress towards being able to prescribe development practices demands a sound theoretical base and this work lays at least the foundations for that base. A cognitive theory includes details of how mental models are constructed, represented and processed. Thus, what this work presents is not yet a fully developed cognitive theory. The account presented here says nothing about how users' and developers' internal models are constructed or processed in cognitive terms. It does make a case for the participants' needs for such models and suggests how they may be represented.

CG(O) may be represented cognitively by the participants in cooperative development activities as CG-iterated, as CG-shared or, indeed, as a combination of both. Despite the intriguing suggestions provided by the interaction analysis, the latter cannot provide conclusive evidence to determine whether participants are using CG-shared or CG-iterated as a cognitive representation of all or part of CG(O). The analysis has provided evidence that the participants had considerably more difficulty in constructing CG(O) than in constructing CG(P). To answer the theoretical question of which cognitive representation is used, we need an experimental design in which we can oblige the participants to perform tasks which we predict can only be performed with one or the other representation. The design and running of such an experiment is a massive undertaking, beyond the scope of the present work.

However, regardless of the precise cognitive representation used by the participants for their common ground, the key insight provided for software development practice is the relative difficulty of establishing strong CG(O) compared with CG(P) in the development situations studied.

A cooperative development approach may alleviate some of the difficulties of constructing the common ground between user and developer which is a prerequisite for successful interactive system development. However, it does not entirely eliminate them. As Kyng [1991, p.72] notes, 'In any real development project, there are always limits to mutual learning. The developers do not become skilled practitioners in the application area and the users do not become technical experts. One of the challenges of cooperative design is to support creative collaboration, despite the fundamental differences among the participants'.

User and developer working together on devising notations and constructing the model will tend to strengthen their assumptions of mutual rationality. This in turn makes copresence heuristics more reliable. The user-developer collaboration also tends to close the 'community gap', thus making community membership heuristics more reliable. Working together allows them to build models of their respective background communities and of their shared working community. The development team, including users and developers, will construct its own cultural community on the basis of their personal common ground. 'A cultural community is really a set of people with a shared expertise that other communities lack' [Clark, 1996, p.102]. This is another quality of cooperative development not offered by conventional development.

On the other hand, developing the communal and personal communal ground in a cooperative development situation may contribute to users' losing touch with their own user community and becoming part of a different, software development team community (see chapter two and [Bødker *et al*, 1987]). This chapter suggests that the development approach adopted in the studied projects provided strong grounds for building CG(P), particularly personal CG(P) through the cooperative work in defining and refining the processes and artefacts together. Unfortunately, it provided quite poor grounds for building personal CG(O) since users gained little or no real experience of the developers' working world and *vice versa*. Communal common ground was, inevitably, not strong between developers and users and this too made for relatively weaker CG(O).

This analysis has two main implications for improving software development practice. One implication for sound practice is to build CG(P) for the external shared models (such as task models and prototypes) based on wide world communal common ground - established through shared membership of wide communities of which the participants are part, for example, English speakers or perhaps London dwellers - and on personal common ground - established through the user-developer collaboration - rather than on professional common ground (which usually does not exist between user and developer). That is, as a slogan for development practice, 'Don't use technical representations from either professional community which the other side doesn't understand!'.

The wide world common ground allows the participants to bring to the development work a strongly established resource of conventions, notations and understandings which may readily be applied to their joint construction of a representation of a work situation or, eventually, of a software system. The extension and specialisation of the participants' common ground through their work together builds on the communal common ground which they brought to the situation. This personal common ground allows them to share representations and understandings effectively and efficiently. There is, however, a danger that personal CG(P) can be too efficient, allowing relatively simple symbols in an external representation to carry meaning for the participants which is not then available to others, for example,

implementors who have not shared in the construction of the personal CG(P). This issue is taken up in chapter six.

A second implication for sound development practice is that user and developer should work together directly on development activity objects in so far as possible. Where the object of the development activity is the envisioned software system, this implies the users' being given active hands-on experience with progressively more highly developed prototypes. This is the model of user participation which we see in many PD approaches (see chapter two).

Where the object of the development activity is the users' work situation, this implies developers' doing users' normal work in the users' work situation, rather than merely discussing the users' work in the development situation. 'Developer participation' in this way should allow user and developer to accrue evidence of shared, personal experience and copresence and of shared professional community membership, all foundations for common ground. Note that this is not a plea for ethnography. The developer should be in the work situation to experience being a user, not to study users' or their work.

The practices implied here should address the twin problems of weak and mismatched relations. As noted in section 4.1.2, the weakest relations were often between the developer and the current work situation, the developer and the envisioned work situation, and the user and the envisioned software. Developer participation should strengthen the first two of these relations. User participation should strengthen the last. The next chapter takes up the analysis with an examination of how successful and effective user participation was in the projects studied here.

Chapter 5

Effective participation: contributing to discourse and artefacts

Having addressed research question one in chapter four, the thesis moves on in this chapter to tackle research questions two (were the projects studied *participatory?*) and three (was user participation in the studied projects *effective?*). This chapter also complements the 'macro-analysis' of chapter four with a more finely focused 'micro-analysis' of samples of interaction from across the cooperative development activities.

The work reported in the chapter develops and applies a framework for assessing user participation in software development in terms of the extent and the effectiveness of user contributions to cooperative software development activities. Section 5.1 begins from a definition of participation in terms of actively contributing and establishes a definition of what it is for participants to make such contributions. Section 5.1.1 establishes a definition of what it is to contribute to the cooperative development process. Section 5.1.2 reviews common means of making contributions to discourse.

Having thus addressed the mechanics of how users may contribute to, and thereby participate in, the activities of cooperative development, section 5.2 examines whether the studied projects were participatory. Section 5.2.1 begins by considering what users may be expected to contribute, what they may bring to the cooperative development process. This establishes a baseline for assessing what users were observed actually to contribute in the cooperative development sessions studied. Section 5.2.2 reviews how this assessment was conducted and subsection 5.2.3 presents the results of this analysis of observed user contributions. Section 5.2.4 then draws lessons from this analysis in response to research question two, concluding that active user contributions made the cooperative development meetings truly participatory.

Section 5.3, builds on the analysis of contributing in previous sections of this chapter and on chapter four. Section 5.3.1 presents a definition of the effectiveness of user participation in terms of the assimilation of users' contributions into the

artefacts of the development work. These artefacts include both the internal artefacts or models which represent the participants' understandings and common ground and the external artefacts such as task models and prototypes. Section 5.3.2 examines the assimilation of user contributions into external artefacts in the projects studied. Section 5.4 then summarises the findings from the chapter.

5.1 The nature of participation: contributing to discourse

The development approach adopted in the projects studied here was intended to promote and to support user participation in the development work. So part of this research explored the extent to which user participation actually occurred in the projects. Therefore, an early problem for this analysis was to produce a definition of user participation. This work takes participation to mean more than mere presence. For a user to participate in development work, she must *actively contribute* to the work. It is not enough to be passively present while the professional developers work or for users to act as an on-line database of information on the target work domain for the professional developers to exploit. Bødker and Grønbaek [1991, p.454; emphasis in original] make a similar point regarding prototyping work: 'we see prototyping with *active* user involvement as a way of overcoming problems that current approaches have in developing computer applications that fit the actual needs of the users'. But the argument goes beyond prototyping.

5.1.1 Making a contribution

What is it, then, actively to make a contribution? In a step towards defining contributions to discourse, Clark and Wilkes-Gibbs [1986] present a model of definite reference in conversation as a collaborative process of interaction between the participants. Clark and Wilkes-Gibbs [1986, p.115] argue that 'A and B must *mutually accept* that B has understood A's references before they let the conversation go on. Conversations proceed in an orderly way only if the common ground of the participants accumulates in an orderly way. A and B must therefore establish the mutual belief that B has understood, or appears to have understood, A's current utterance before they go on to the next contribution to the conversation'. The establishment of such a mutual belief constitutes a contribution to the discourse by A in collaboration with B.

Clark and Schaefer [1987a, 1989] extended this model of collaboration in definite reference to produce a general model of contributing to discourse. According to Clark and Schaefer [1987a, 1989], making a contribution to a discourse consists in two collaborative processes:

- (i) *content specification*: the contributor tries to specify the content of her contribution and the recipient tries to register that content;

(ii) *content grounding*: the contributor and recipient try to achieve the state in which they mutually believe that the recipient has understood what the contribution meant, to a criterion sufficient for current purposes.

According to Clark and Schaefer [1987a, 1989], a contribution is a collective act in which the participants add what A meant to their common ground. In this process, A makes the individual participatory act of contributing and B makes the individual, participatory act of registering A's contribution. (Note the use of the term contribution with two senses: A and B's joint, collective act and A's individual, participatory act.)

Clark and Schaefer [1989] propose that for an expression *e* presented by A, B may be in one of four states.

State 0: B didn't notice that A uttered any *e*.

State 1: B noticed that A uttered some *e* (but wasn't in state 2).

State 2: B correctly heard *e* (but wasn't in state 3).

State 3: B understood what A meant by *e*.

For the achievement of content specification and grounding, A and B must both believe that B is in state 3. The collective act of contributing has two phases: presentation and acceptance. In the presentation phase, A presents a contribution, awaiting evidence from B that it has been heard and understood. In the acceptance phase, B provides such evidence. Clark [1996] proposes four main classes of evidence which B may provide of her understanding of A's contribution.

1. *Assertions of understanding*. Simple responses such as 'uh-huh', 'mm', 'I see' or just a nod or smile.
2. *Presuppositions of understanding*. Simply in taking up a next, relevant¹ turn, B provides evidence that she presupposes that she has understood A's contribution.
3. *Displays of understanding*. The content of B's next relevant turn often displays B's understanding of A's contribution, for example in answering a question.
4. *Exemplifications of understanding*. B may respond with a paraphrase or verbatim repetition of A's contribution or may make some nonverbal expression of her understanding.

In each case, B provides evidence of understanding in the expectation that A will accept it. Contributions are commonly made with straightforward, trouble-free

¹ The notion of relevance is taken up in section 5.1.2.

presentation and acceptance. Take the following exchange on an envisioned software design supporting an envisioned work situation. A developer has asked if a user could assume more than one role in the work situation and, therefore, should require the facility to switch between what are strictly separate tasks in the current work situation. The developer asks if this should be presented as a capacity to switch between windows or if it should be presented as running different applications. The user takes this up in the extract below.

C133108

U: If he was multitasking uh and it was decided, right, it's a tiny small little thing, he's going to do every single thing. We would authorise him to have multi win, multi choice of windows so he can say right, he can go to, he can select uh actions and do anything that he wants with actions, he can select uh uh a statement and read it, he can receive it, register it, whatever. He could flick from window to window, from task to task. If he was authorised to do so.

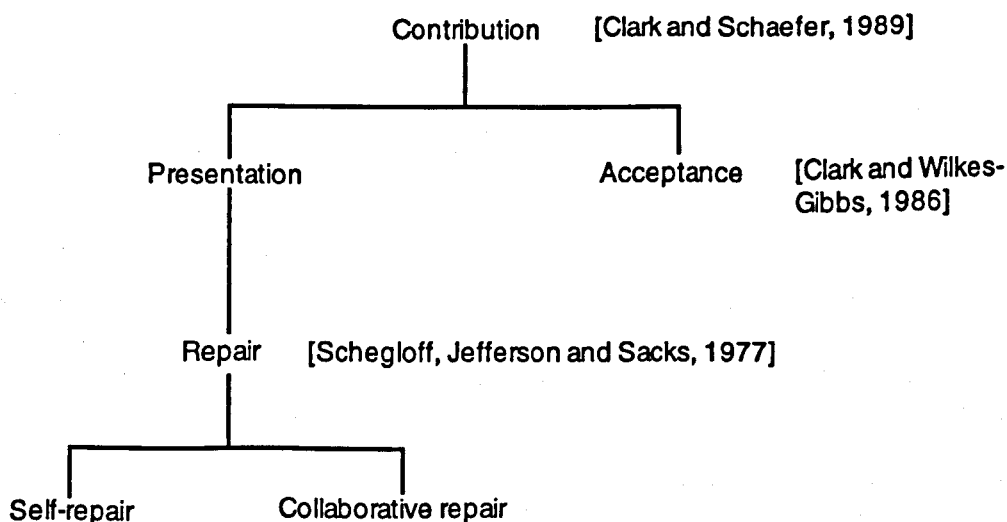
D: Yeah. Mm. It's not a question of presentation. It's a question of what tasks he's authorised to perform.

U: Right.

D first provides an assertion of understanding (with 'Yeah. Mm.') and then an exemplification of understanding (with 'It's a question of what tasks he's authorised to perform.'). U accepts this evidence of D's understanding with 'Right'.

The presentation phase may become more complicated if the contributor attempts to repair an anticipated difficulty with the presentation or if the recipient gives evidence of trouble in understanding all or part of the contributor's presentation of *e*. In such cases, the presentation may be refashioned by repair [Schegloff, Jefferson and Sacks, 1977], expansion or replacement until the recipient provides evidence of acceptance. The collaborative structure of a contribution is illustrated in Figure 5.1.

Figure 5.1: Contributing to discourse



The notion of grounding contributions was further developed by Clark and Brennan [1991]: 'participants try to establish that what has been said has been understood. In our terminology, they try to ground what has been said - that is, make it part of their common ground' [p.128]. Clark [1996, p.221] also notes that 'to ground a thing, in my terminology, is to establish it as part of common ground well enough for current purposes'. Thus, Clark and his colleagues argue that achievement of the mutual belief that B has heard and understood the assertion as A intended constitutes the assimilation of A's contribution into the common ground which A and B hold.

The argument that once a presentation has been accepted, it thereby is added to the participants' common ground derives from the view of common ground as common ground *about the discourse*, that is CG(P) in the terminology of chapter four of this work. However, that B has understood the content of A's contribution does not entail that this content has been reconciled with B's extant beliefs, understandings and knowledge. In the terms of the analysis here, the content of that contribution has not yet been offered up to the participants' extant internal models (see Figure 5.2). For example, A may state proposition *p*, thereby contributing *p* to their discourse. When B has registered the content of *p* and the participants have each established the belief that B understood what A meant by *p*, then, according to Clark, *p* has been added to the common ground of A and B. However, it may, for example, be the case that B has heard *p*, understood it as A intended and yet rejects or ignores its content. Thus, it may at best be said that it is common ground for A and B that (A believes *p*), but not that *p*. The interaction processes which may then ensue were described in chapter four above and are discussed in this chapter with specific regard to the assimilation of contributions from users in user-developer discourse.

Clark and Schaefer [1989] argue that all contributions which are established as heard and understood are thereby assimilated into the participants' common ground, even when the contribution contradicts their existing common ground. Stalnaker [1978] describes common ground as the presuppositions of the participants. Lewis [1969, p.339] observes that these 'presuppositions can be created or destroyed in the course of a conversation'. Clark and Schaefer [1989] acknowledge this and comment that 'even when presuppositions are destroyed, the participants know they have been destroyed, and that knowledge itself becomes part of their common ground. So we can say that the common ground of the participants *accumulates* in the course of a conversation' [Clark and Schaefer, 1989, p.146; emphasis in original].

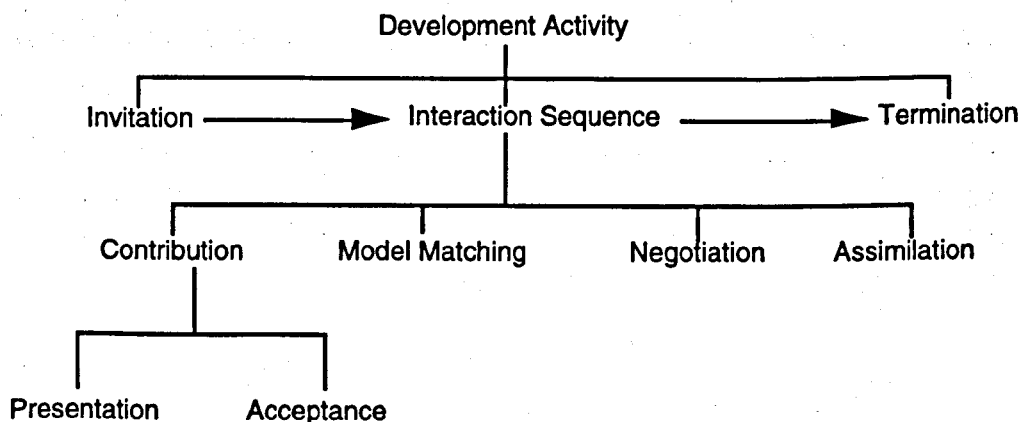
As developed in chapter four, CG(O) is the common ground about the objects of the development activities, about the work situations, requirements and designs. It includes *what* is represented in external shared models, such as task models and prototypes. CG(P) is the common ground about the development activities, about

the relationships amongst the participants, about the notations and conventions in use, about who has contributed what to the discourse. It includes *how* external shared models, such as task models and prototypes, represent the objects of the development activities.

Consider the following situation in a cooperative development discourse. Presupposition p is part of $CG(O)$ for A and B. It is also part of $CG(P)$ for A and B that A contributed p at some previous point in the discourse. A then asserts $(NOTp)$. If B accepts, i.e. hears and understands, this assertion and A thereby successfully contributes $(NOTp)$ to the discourse, then $CG(P)$ accumulates now to include the presuppositions that A contributed p , that p became part of $CG(O)$ and that A subsequently contributed $(NOTp)$. However, whether or not p is destroyed or replaced by $(NOTp)$ in $CG(O)$ depends upon B's reaction to A's contribution of $(NOTp)$.

Thus, once again, Clark and Schaefer's [1989] view of the orderly accumulation of common ground even from conflicting contributions is adequate for $CG(P)$ - common ground about the joint activity - but not for $CG(O)$. Figure 5.2 illustrates the interactional processes which occur in cooperative development sessions. Contributing to the discourse of such sessions - and to $CG(P)$ - may entail only the process of contribution (or of invitation) and its constituents (see Figure 5.1), while contributing to $CG(O)$ entails the interaction sequence of Figure 5.2 and its constituent processes: contribution, model matching, negotiation and assimilation.

Figure 5.2: Contributing to common ground



Clark and Schaefer's [1989] definition of contributing usefully covers making the contribution to the discourse, as it were putting the content on the table. However, it does not adequately capture the full processes of making the content available for consideration and assimilating it into the participants' common ground about the objects and artefacts of the development work. For that, we need to consider not just whether the content of a contribution has been heard and understood, but also

what is then done, or not, to relate the content of that contribution to the participants' extant understandings and common ground.

The collaborative making of a contribution to discourse, i.e. its presentation and acceptance, is *per se* a necessary but not a sufficient basis for the accumulation of common ground; or at least, in the terminology of chapter four, of Common Ground (O). Making a contribution to the discourse, establishing that the contribution has been heard and understood (well enough for current purposes), is not synonymous with making a contribution to CG(O). The successful achievement of such a contribution establishes *only* that B has heard and understood A's presentation.

Figure 5.3: Successful and effective contributing

Status of contribution <i>p</i>	Recipient B's state (after [Clark and Schaefer, 1987a])	Clark's [1996] view	This work's view
Heard	2	$p \notin CG$	$p \notin CG(P)$ $p \notin CG(O)$
Understood	3	$p \in CG$	$p \notin CG(P)$ $p \notin CG(O)$ (A believes p) (contribution is successful) $\in CG(P)$ (A contributed p) $\in CG(P)$
Assimilated	4	n/a	$p \in CG(P)$ \vee (contribution is effective) $p \in CG(O)$

Consider that participant A presents a proposition *p* (whether intended as a contribution to CG(O) or to CG(P)). If both A and recipient B come to the mutual belief that B has understood what A meant by *p*, then metapropositions such as (A believes *p*) and (A contributed *p* to the discourse) are thereby assimilated into

CG(P). However, this does not entail that p has been assimilated into either CG(O) or CG(P), as intended by A. While A has *successfully* introduced p to the discourse, she has not yet *effectively* contributed p to the participants' common ground. To achieve the latter requires that A and B have the mutual belief that B has reached a further state (let us call it state 4, after [Clark and Schaefer, 1987a]) in which she has assimilated p into her internal models or beliefs. See Figure 5.3.

Having achieved state three for A's contribution e , participant B may assimilate the content of e with her internal models and/or with the participants' external shared models. B may, however, follow other courses which do not lead to state four. Evidence that a contribution has been assimilated into an external shared model is by its nature public and accessible. The participants therefore have a ready check for their mutual belief that state four has been achieved. However, the immediately available evidence that a contribution has been assimilated into the recipient's *internal* models may at times be no stronger than evidence that the recipient has reached state three. It is, therefore, correspondingly difficult for contributor and analyst to be sure that the contribution has indeed been assimilated into the recipient's internal models.

Whether or not the content of e is assimilated into the participants' common ground is a question of the *effectiveness* of A's participation in the joint activity. This issue is taken up in section 5.3 below. In the meantime, we may move on to section 5.2 with a definition of contributing to discourse similar to Clark's [1996] definition. We take the collaborative process of successfully making a contribution to discourse (in distinction from effectively making a contribution to common ground) as having two phases, presentation and acceptance. A contribution to the discourse has been made when contributor and recipient both believe that the recipient has heard and understood, well enough for current purposes, the contribution as the contributor intended it. Thus, active user participation in a software development meeting may be said to consist in the occasions on which a user makes a contribution to the discourse which user and developer both believe the developer has heard and understood as the user intended it.

5.1.2 Types of contribution

A participant's contribution may invite or provide² information or verification on any of the objects of development work: for example, the users' current work situation, requirements for an enhanced work situation and system, an envisioned enhanced work situation or an envisioned software system to operate in that situation. Therefore, each contribution to CG(O) consists in a particular development activity; for example, designing an envisioned work situation. This allows further refinement of the interaction analysis in terms of contributions, since a long or complex turn by a participant may be divided into several contributions, each potentially contributing to a distinct development activity.

²Such contributions are referred to in this work as invitational and provisional contributions, respectively.

Clark and Schaefer [1989] identify two dominant patterns or types of contributions in a discourse: by turns and within turns. In the most simple cases of contributions by turns, a new contribution is initiated with every cooperative change in turns. Such a contribution either

- (a) closes the current contribution when it is relevant and at a level as high as the current contribution - subject to a veto by the current contributor; or
- (b) opens a side sequence that is closed when a relevant new contribution is initiated at a level above the side sequence [Clark and Schaefer, 1989].

Relevance is similar to the concept of 'adjacency pairs' [Schegloff and Sacks, 1973] whereby a contribution creates an expectation of a limited range of relevant next contributions. An example is the contribution 'What would you like to drink?', to which 'Lemonade, please' is a relevant next contribution but 'Three pounds per metre' is not. A side sequence is an artefact of the recursive (chapter four), or hierarchical [Clark and Schaefer, 1989], nature of contributions. Thus, the following exchange consists of four contributions, the middle two of which form a side sequence.

A: What would you like to drink?

B: What have you got?

A: Lemonade and water.

B: Lemonade please.

In this exchange, each turn corresponds to a single contribution. However, multiple contributions may also be made within a single turn [Clark and Schaefer, 1989]. When a participant takes a long turn, her coparticipants generally express their acceptance of separate parts of the turn by presenting acknowledgements or backchannels, such as 'mm-mm' or 'yes'. Acknowledgements themselves generally do not constitute turns and each acknowledgement marks the end of the scope of its acceptance. Acknowledgements are generally placed at or near the ends of major grammatical constituents [Oreström, 1983] and 'divide extended turns into units that are practical for establishing understanding and correcting misunderstandings' [Clark and Schaefer, 1989, p.166]; that is, into contributions.

5.2 Were the studied projects participatory?

With participation defined, in section 5.1, in terms of actively contributing to discourse, the level of user participation in the studied projects may be assessed by comparing observed user contributions to the user-developer discourse with predictions of what potential contributions users could be expected to bring to the system development work. This section makes such a comparison and thereby seeks to provide an answer to research question two: were the studied projects truly participatory?

5.2.1 Expected user contributions to user-developer collaboration

Having established how users may contribute to the activities of cooperative development through user-developer discourse, we now consider what users could be expected to contribute to such work. This will provide a baseline from which to assess what users were observed actually to contribute in the cooperative development sessions studied.

Several strands of research and practice in computer science have in recent years promoted and encouraged the more or less active involvement of users in software development projects. Perhaps the most enthusiastic of such researchers and practitioners, as noted in chapter two above, have been proponents of 'participatory design'. Adherents of participatory design, and various similar approaches such as 'user centred software design' [Norman and Draper, 1986] argue that users bring to the development work knowledge and skills to which the developers alone cannot have access [O'Neill, 1996].

For example, Bødker and Grønbaek [1991, p.454] suggest that their 'cooperative prototyping approach aims to establish a design process where both users and designers are participating actively and creatively based on their differing qualifications'. Similarly, Kyng [1995, pp.87-8] argues that 'since neither professional designers nor end users can fully understand each others' practices or meanings, we must actively bring these experiences closer together, with the aim of jointly creating both new computer applications and a design process that makes sense to all those involved'. Kensing and Munk-Madsen [1993, p.79] argue that 'at the outset the users have some knowledge of their present work and of organizational options. The system developers have some knowledge of the technical options with regard to hardware and software. At the outset this is all they need to know'.

So, what skills and knowledge can users and developers respectively be expected to bring to the software development process? In the broadest terms, professional developers typically have knowledge of the technical possibilities and constraints of potential hardware and software products, skills and experience in developing and combining such products and knowledge of the kinds of artefacts and procedures involved in the development process. Developers, however, often have no previous experience of the user's work domain, its requisite skills and knowledge. Users, on the other hand, typically have a great deal of knowledge and skill based on and contributing to their experiences of the very work situation into which the software system is to intrude. It is a rare user, however, who has the knowledge, skills and experience of the professional software developer. It should be clear that the two diverse sets of knowledge and skills must be brought together in the course of developing a software product to fit the work situation. The challenges for the professional software developer lie in encouraging the users' contributions of domain knowledge and skills, in contributing the developer's own knowledge and

skills and in cultivating in all participants an appropriate receptivity to others' contributions.

The exchange of knowledge between user and developer depends, rather obviously, on communication. The analysis presented so far suggests the following characteristics of this communication. Any participant in a cooperative software development discourse, user or professional developer, may in principle invite a communication and any may provide a communication in response to such an invitation. Whilst the latter provision has been described above as a contribution, it should be noted that in discourse terms the invitation itself may also be described as a contribution. Chapter four described how in user-developer discourse, a typical interaction pattern is an invitation followed by a sequence of contributions which may themselves entail further sequences at lower levels. Within these interaction sequences, communications may be provided in response to an explicit invitation from one's interactional partner or in response to one's perception of a partner's implicit need for information or clarification.

What is communicated in the cooperative development situation is generally of two forms [O'Neill, 1996]. *Information* may be communicated to fill a perceived gap in the recipient's knowledge and *verification* may be communicated to confirm or to disconfirm an aspect of the recipient's current understanding. For example, the provision of information in current work situation analysis may take the form of a user explaining to a developer how she performs a particular task. The provision of information in software design may take the form of proposing a particular design solution, say a range of menu options. As noted in chapter four, true common ground is impossible to achieve since only omniscient participants could know that they hold the same understanding or belief. At best, a participant holds an understanding and assumes that her coparticipant holds the same understanding. Thus, the communication of information and verification becomes the cornerstone of *attainable* common ground. Information allows a participant to construct understandings and verification allows her to assess these and to compare them with her coparticipants' understandings.

So, a particular contribution to the user-developer discourse

- may be an invitation by either user or developer,
- may be a provision by either user or developer,
- may entail new information or the verification of an already established understanding,
- may relate to a development activity object or artefact³,

³ The distinction here is between, for example, the envisioned software system as the object of the development activity and the prototype as the artefact of the development activity.

- may contribute to a specific development activity.

As described in chapter four, participants in software development work may utilise an external, shared model (which may be relatively formal, such as a task model, or informal, such as a simple list) to represent information provided through interactional contributions on a development work object and also to support the invitation of contributions to validate the information represented in the model and to provide new information where gaps are apparent in the model. Use of such an external, shared model prompts a third form of contribution which may be invited and provided by participants.

This third form of contribution is *direct representation*: any participant may manipulate, or invite another to manipulate, an external, shared model in order to represent information about a development activity object. This representation may be of information which already has been made public through a previous contribution but hitherto has not been assimilated into the external, shared model. (In section 5.3 below, this form of representation is viewed as a measure of the effectiveness of the original contribution.) On the other hand, the manipulation of the external, shared model may constitute *per se* a contribution of new information to the current representational content of the model.

For example, in the following exchange, the developer D makes a change to the task model and then explicitly makes public his internal model of what the external shared model now represents.

CI11052

[D writes on task model]

D: I've changed it and the reason we got confused is there seems to be two circumstances. There are some uncompleted actions, for example the guy is on holiday.

U: That goes to the action allocator.

D: I got the two muddled. Here [points to TM] I've said that he has accepted the uncompleted action, and then an uncompleted action comes through the system, so he says I've accepted this uncompleted action. It comes through here [points to TM] to the action allocator, and there is a thing here [points to TM] about what you do with uncompleted actions.

In this example, the developer makes a contribution to CG(O). But rather than expressing his contribution verbally, he represents it directly in the external shared model, drawing directly on the task model chart on the wall. He provides a verbal utterance only as a *post hoc* commentary on and explication of his contribution.

Direct manipulation of the external, shared model is a particularly effective form of contribution to the development activity, since there is immediate, *de facto* assimilation of the content of the contribution into the development artefact which is

the external, shared model. It does, however, raise questions of cooperation which are taken up briefly below.

The conceptual analysis above has emphasised that in a truly cooperative development situation, any participant, user or professional developer, may invite or provide any type of contribution. Indeed, examples of all kinds of combination of participant and contribution type may be found in the video records of development meetings. However, the principle that users and developers bring different knowledge, skills and demands to the development process implies that each may be expected to invite and to provide different types of contribution in different frequencies.

Kensing and Munk-Madsen [1993] present a model of six areas of knowledge in user-developer communication. These areas are abstract knowledge and concrete experience of the users' present work, the available technological options and the envisioned new computer based system. This model is presented here as Figure 5.4. Kensing and Munk-Madsen [1993] propose that users generally bring to the development process knowledge only of area one, while developers generally bring to the development process knowledge only of areas three and four. Kensing and Munk-Madsen [1993] argue that developers' responsibilities therefore include applying tools and techniques which allow users to develop knowledge of areas two, five and six and which allow developers to develop knowledge of areas one, two, five and six.

Figure 5.4: Six areas of knowledge in user-developer communication [Kensing and Munk-Madsen, 1993]

	Users' present work	Technological options	New system
Concrete experience	(1) Concrete experience with users' present work	(3) Concrete experience with technological options	(5) Concrete experience with the new system
Abstract knowledge	(2) Relevant structures on users' present work	(4) Overview of technological options	(6) Visions and design proposals

In the development projects reported here, developers' knowledge of area one was developed in a range of ways including observing users at work, experimenting with users' current software, interviewing users and through the user-developer interactions which comprised the cooperative development of models of the users' current work situation (TM1). This latter cooperative work also largely constituted the development of users' and developers' knowledge of area two, with the developing models providing the 'relevant structures'. The development of users'

and developers' knowledge of area six included envisioning and designing both a new work situation for the users and a new software system to support that work situation. The development of proposed task models (TM2) supported this process for the envisioned work situation, while the development of paper and software prototypes supported this process for the envisioned software. Use of the prototypes also supported the development of users' and developers' knowledge of area five. Developers' knowledge of areas three and four and users' knowledge of area one facilitated their respective contributions to requirements analysis.

Thus, in the cooperative development meetings we should expect, for example, developers' contributions to the cooperative analysis of the users' current work situation to consist frequently of invitations for information and verification from the user. We should expect rather less frequent invitation and provision by the developer of representations of the users' current work situation (as the results of sequences of interaction periodically are assimilated into the external shared model of the latter). Least frequent of all should be contributions by the developer which provide the user with information or verification about the users' current work situation. Figure 5.5 below summarises the relative frequencies, on a gross ordinal scale of high, medium and low, with which each type of contribution was predicted to occur in the cooperative development meetings studied.




Several patterns are apparent in Figure 5.5. Examples include the following. Across contribution type: users should make more *provision* than *invitation* contributions to work situation analysis. Across development work activity: developers should make more provision contributions to *software design* than to *work situation analysis*. Across participant type: *users* should make more invitation contributions than *developers* in designing proposed software.

So, Figure 5.5 presents a summary of predictions of the gross, relative frequencies with which users and developers may make different kinds of contributions to the discourse of their cooperative development meetings. These predictions furnish a base line against which to assess the actual observed frequencies with which these contributions were made in the cooperative development meetings studied.

It is worth noting once again that the discussion presented here addresses only contributions to the user-developer discourse which directly constitute one of the four identified development work activities. Consideration is not given in the current analysis to other contributions which may constitute, for example, project management activities or socialising activities within the development group. It is recognised, however, that such other activities form an important part of development work [Minneman, 1991; Olson, Olson, Carter and Storrøsten, 1992] and may account for contributions by both users and developers. Neither does this analysis assess contributions which the participants intended to make directly to CG(P), such as proposing a particular notational convention.

Figure 5.5: Expected relative frequencies of contributions from users and developers

		Analysing current work situation		Analysing requirements		Designing envisioned work situation		Designing envisioned software	
		User	Developer	User	Developer	User	Developer	User	Developer
Invite	Information	Medium	High	Medium	High	Medium	Medium	High	Medium
	Verification	Medium	High	Low	Medium	Medium	High	High	Medium
	Representation	Medium	Medium	Low	Low	High	Low	High	Medium
Provide	Information	High	Low	Medium	Low	Medium	High	Low	High
	Verification	High	Low	High	Low	Medium	High	Low	High
	Representation	High	Medium	Medium	High	Low	Medium	Low	High

Predicted frequency:  High  Medium  Low

5.2.2 Methodology in assessing contributions

During the development projects, user-developer meetings were called to address a specific development activity (for example, a meeting to work on designing an envisioned enhanced work situation). In analysing contributions, samples of user-developer interaction were taken from the video records of three separate development meetings: one whose declared purpose was to analyse the users' current work situation and to construct a task model thereof; one to design a new work situation and to construct a task model thereof; and one to design a new software system and to construct a paper prototype thereof. It was hoped, therefore, to collect a range of data across different development activities.

The samples were taken from the CS project since it had the wider range of recorded meetings. Samples were taken from meetings with the same participants, one user and two developers, in order to avoid introducing personality biases across

the samples [Hawk, 1993]. The first video tape, CS1, lasted 79 minutes. The second, CS5, lasted 70 minutes. The third, CS7, lasted 72 minutes. (See section 3.3 of chapter three.) Each sample began fifteen minutes from the start of the meeting, to allow time for settling down to the work, and was of ten minutes' duration. This provided quite large contribution counts (70, 44 and 57 respectively for the three samples) and the high AT:ST ratio [Sanderson and Fisher, 1994], in the region of 100:1 just to fill in a table such as Figure 5.6, mitigated against using longer or more numerous samples.

The analyst first transcribed the samples. The transcriptions were then used to fill in blank tables similar to Figure 5.5. A count was made in the appropriate table cell for each informing or verifying contribution. The video sample was then reviewed with the corresponding table to count direct representation contributions since these could not reliably be assessed from the transcripts. A discrete contribution was identified in the terms of section 5.1.2 above. Only contributions to the four identified development activities were counted for these purposes. Contributions to other activities, such as social or project management activities, and direct contributions to CG(P), such as defining notation, were not counted. This is illustrated in the following example in which a developer (D1) wants to have assimilated into the external shared model a contribution which the user (U) has just made. The external shared model is a task model developed on a whiteboard and D1 looks for a cloth which is used to erase parts of the current representation.

CS21541

- (1) D1: That way? [*points to TM*]
- (2) U: Ah more usually from here [*points*] to one of these [*points*] two [*points*].
- (3) D1: Right.
- (4) D1: [*looking around*] Where'd the cloth go? [*D1 looks at D2.*]
- (5) [*U laughs. D2 hands cloth to D1.*]
- (6) D1: The alternative is to use our hands and it ends up very messy very quickly.
- (7) [*D1 erases part of model.*]
- (8) D1: Right so can you show us where? [*D1 picks up a marker and hands it to U.*]
- (9) U: Yeah OK.

The first contribution is D1 inviting verification in (1). In (2) U makes a contribution by providing the requested verification. In (3), D1 accepts U's contribution. (4-6) were not direct constituents of the development activities in the column headings of Figure 5.5 and so were not counted for the purposes of this analysis. In (7), D1 provides a direct representation contribution to the external shared model of the users' current work situation. In (8), D1 invites a direct representation contribution from U. Finally, in (9) U accepts D1's contribution of this invitation and the interaction continues. So, in this example we have a total of only four distinct contributions, three by D1 and one by U.

5.2.3 Observed user contributions to user-developer collaboration

The series of tables below summarises the results of assessing the numbers of various types of contribution to cooperative development activities by both developers and users. Figure 5.6 presents these results for the sample of interaction from a meeting to analyse the users' current work situation and to construct a task model thereof (from tape CS1). Figure 5.7 presents these results for a sample of interaction from a meeting to design a new work situation and to construct a task model thereof (from tape CS5). Figure 5.8 presents these results for a sample of interaction from a meeting to design a new software system and to construct a paper prototype thereof (from tape CS7). Finally, Figure 5.9 presents the accumulated results across the three samples.

Figure 5.6: Contributions in CS1 meeting to analyse the users' current work situation

		Analysing current work situation		Analysing requirements		Designing envisioned work situation		Designing envisioned software	
		User	Developer	User	Developer	User	Developer	User	Developer
Invite	Information		8		2				
	Verification		13						
	Representation		1						
Provide	Information	22		3	1		1		
	Verification	14							
	Representation	3	2						

It must be cautioned that the figures and interpretations presented here should be viewed within the context of the overall interaction analysis. While providing interesting data, any technique which reduces human interaction to numbers loses the interactional richness of the primary data. As noted above, there are the limitations of attempting to count only particular types of contributions. Perhaps

more importantly, simply recording that there was a contribution dispenses with the pragmatics of the contribution. Examples of patterns of interaction which were identified during the interaction analysis are provided in section 5.2.4. These patterns often could not be inferred solely from the bare numbers presented in the figures below, but their identification was nonetheless supported by these figures.

Figure 5.6 shows that the vast majority (63/70) of contributions made in the sample interaction were to the 'official' or declared development activity of the meeting: analysing the users' current work situation. No contributions were observed to the activity of designing envisioned software, while only one contribution was made to the activity of designing an envisioned work situation. Six contributions were observed to the activity of analysing requirements.

Figure 5.7: Contributions in CS5 meeting to design a new work situation

		Analysing current work situation		Analysing requirements		Designing envisioned work situation		Designing envisioned software	
		User	Developer	User	Developer	User	Developer	User	Developer
Invite	Information		2		1		1		
	Verification		3		1		1		
	Representation								
Provide	Information	6	4	2	4			1	5
	Verification	3	4	1			2		
	Representation						3		

Contributions which invited information, verification or representation were all made by a developer. Contributions which provided verification were all made by a user. Twenty-five of twenty-seven contributions which provided information were made by a user, with the two developer provisions contributing respectively to requirements analysis and to work situation design. Of the five contributions which

provided direct representations, three were made by a user and two by a developer. The two developer contributions and two of the three user contributions were used to demonstrate routes which telephone calls could take between roles in the work situation. This was easy for the participants to convey with a direct representation contribution.

Figure 5.7 paints a rather different picture of the contributions made in a meeting whose declared development activity was designing an envisioned enhanced work situation for the users. Figure 5.7 shows that only 15.9% (7/44) of contributions to development activities were made to this activity, none by a user. 13.6% (6/44) were made to the activity of designing envisioned software, 20.5% (9/44) to the activity of requirements analysis and 50% (22/44) to the activity of analysing the users' current work situation.

Figure 5.8: Contributions in CS7 meeting to design a new software system

		Analysing current work situation		Analysing requirements		Designing envisioned work situation		Designing envisioned software	
		User	Developer	User	Developer	User	Developer	User	Developer
Invite	Information		3		4			1	
	Verification		1		3			1	5
	Representation								
Provide	Information	5	1	6	1			1	5
	Verification	1	2	3	2		1	5	5
	Representation								1

Once again, contributions which invited information or verification were all made by a developer, although in this sample they were spread more evenly across development activities. There were no contributions inviting direct representation. There were only three contributions which provided direct representations, two made by one developer, one made by the other developer (again his sole

contribution) and all three made to the activity of designing an envisioned work situation.

Contributions which provided information or verification were concentrated not in the declared development activity of designing an envisioned work situation but in analysing the users' current work situation. Only one such contribution to the two designing activities was made by a user, the other seven by a developer.

Figure 5.8 shows that almost half (24/57) of recorded contributions were made to the declared development activity of designing a new software system. Only one contribution was made to designing the envisioned work situation, nineteen were made to analysing requirements and thirteen to analysing the users' current work situation.

As in the other two samples, contributions which invited information or verification on requirements or on the users' current work situation were all made by a developer. In contrast, two of the seven contributions which invited information or verification on the envisioned software design were made by a user.

Figure 5.9: Aggregate contributions across three CS meetings

		Analysing current work situation		Analysing requirements		Designing envisioned work situation		Designing envisioned software	
		User	Developer	User	Developer	User	Developer	User	Developer
Invite	Information		13		7		1	1	
	Verification		17		4		1	1	5
	Representation		1						
Provide	Information	33	5	11	6		1	2	10
	Verification	18	6	4	2		3	5	5
	Representation	3	2				3		1

Of the nine contributions which provided information or verification in analysing the users' current work situation, six were made by a user and three by a developer. Of the twelve contributions which provided information or verification in analysing requirements, nine were made by a user and only three by a developer. A developer made the solitary contribution to designing the envisioned work situation.

There was only one instance of a direct representation contribution, made by a developer in designing the envisioned software system.

Figure 5.9 shows an aggregate total of one hundred and seventy-one contributions recorded across the three samples, seventy-eight by a user, ninety-three by a developer. Of this number, a user made two invitational contributions to designing the envisioned software, one contribution which invited information and one which invited verification. No other invitational contributions were made by a user. There were forty-nine invitational contributions by a developer. The only contribution inviting a direct representation contribution was made by a developer in analysing the users' current work situation.

Seven direct representation contributions were made by a developer and three by a user. Seventy-three user contributions were made which provided information or verification, while thirty-eight such contributions were made by a developer.

5.2.4 Conclusions on participation

The results shown in Figure 5.6 were largely as expected. Most contributions were to the declared activity of the meeting. Only a few were to other activities and they dwindled across the activities, from sixty-three contributions to current work situation analysis to no contributions to software design. User contributions were all provisional, while developers were predominantly invitational. The two developer contributions which provided information were to requirements analysis and work situation design respectively, activities to which the developer is expected to bring knowledge of technological options. The other two provisional developer contributions were direct representation contributions. In both these cases, the developer made the contribution and immediately sought verification from the user.

In Figure 5.7, the results were not as expected. Half of all the noted contributions in this sample were concerned not with the declared development activity of designing an envisioned work situation, but with analysing the users' current work situation. One intuitive explanation for this is that the latter activity had not adequately been conducted in previous development work and had to be pursued in this meeting before other activities could continue. However, this is not borne out by the data. Whilst a few contributions fitted this pattern, many of the contributions of information and verification on the users' current work situation were made by the developer. This contrasts with the monopolisation by the user of these contributions in Figure 5.6. By the meeting of Figure 5.7, the developers had

worked with several users and had gathered knowledge about the work tasks and domain which any one user is unlikely to have had. In the course of their work, they then passed some of this knowledge on to other users and this is reflected in the pattern of contributions here. Thus, a user's knowledge of her own working environment may be enhanced through collaboration with developers. This finding is reinforced by the comments of several users, across both the CI and CS projects, who spontaneously and independently remarked that one of the biggest gains for them from the cooperative development work was a more extensive and more detailed understanding of what their department actually did and how their tasks fitted with those of their colleagues than they had ever had before.

Perhaps the most unexpected finding from Figure 5.7 is that no contributions were made by a user to the declared activity of designing an envisioned work situation. This result is all the more surprising since a developer made two contributions inviting a user to contribute to this activity. The user responded to the invitation for information with 'Right. I thought [U]⁴ had already given you some suggestions', and to the invitation for verification with silence which the developers seem to have taken as confirmation.

It was to be expected that provisional contributions to work design activities should be made more often by developers than by users (see Figure 5.5). That the expected low number for users should in fact be zero may be a result of the short sample time. This may also explain the generally low number of contributions to the declared activity in the sample. Certainly, interaction analysis of the overall video records reveals a large number of contributions to work situation design, many of them made by users. Given that interaction sequences are at times played out over relatively long periods, it requires only one or two sequences on a particular activity heavily to influence the results across a short sample of interaction. In the sample used here, there were many contributions to CG(P), communicating what had gone on in previous meetings and planning what should be covered in this and subsequent meetings. For example:

CS414221

D: Mm. So there's questions there. Uh, we're going to go through it with [U]⁵ and sort of agree which are the best to do. But for the moment five uh... I, I don't want to break down recording a query.

Time spent on these contributions left correspondingly less time to make contributions to designing. It is likely that samples from later in the same meeting, with the user more *au fait* with what has gone on in his absence, should show fewer CG(P) contributions and more contributions to design activities.

In Figure 5.8, most provisional contributions to the activity of designing envisioned software came, not unexpectedly, from a developer. Five of the six contributions

⁴ Another user.

⁵ Another user.

of these types which came from a user were verifications of design proposals in response to an invitation from a developer. In contrast, most of the developer contributions were spontaneous, with only two invitations from a user. Figure 5.9 reveals that these were the only invitational contributions from a user across the three samples of development work.

This overall low number of user invitational contributions, especially in comparison with the number of invitational contributions by a developer, may reflect a reticence on the part of the user to becoming proactive, with the developer viewed by all the participants as 'in charge' of the meeting and the user remaining largely in 'on-line database' mode. However, in mitigation of this, we should expect many invitational contributions from a developer and few, if any, from a user in analysing the users' current work situation and requirements - and this is what we find. Similarly, we should expect more invitational contributions from a user in design activities and again this is what we find.

The quite high numbers of requirements analysis and current work situation analysis contributions in Figure 5.8 are largely the result of a pattern of interaction in which interaction sequences which included contributions of design proposals were often followed by long clusters of sequences inviting and providing information and verification on current work situation analysis and requirements analysis. Typically, a design proposal sparked interaction on the requirements which it was intended to meet. This in turn meant going back to the needs for enhancements to the users' current work situation from which those requirements derived. When this chain had been established, the interaction sequence often terminated with a return to the original design proposal.

This is an example of a distinct pattern of temporal relations between contributions which is not readily apparent from the bare numbers presented above. At a higher resolution analysis than the making of individual contributions, there is a pattern of invitational contributions from one participant being followed by provisional contributions from another. In a simple form, this is quite intuitive: one asks for some piece of information or verification or for a direct representation, one's coparticipant provides it. However, the analysis here suggests a more complex pattern in which the initial invitational contribution is followed by several, often related, provisional contributions of various types. These contributions in their turn may entail recursive invitational and provisional contributions. This entailment forms the basis for the interaction sequence as a unit of user-developer discourse, as described in chapter four.

At a still higher resolution of analysis, there are further patterns. For example, across all the projects and meetings, there was a recurring pattern of a cluster of interaction sequences whose constituent contributions invited and provided information, followed by a cluster of interaction sequences whose constituent contributions invited and provided verification of the informational content of the

previous cluster's contributions. In design meetings, at a still higher level of analysis, we have the pattern, described above, of design contributions bracketing contributions to requirements analysis and current work situation analysis.

In addition to the loss of richness in counting contributions, a further difficulty with such analysis is that higher resolution patterns, such as those described above, often become apparent only over longer periods of interaction than the relatively short samples for which the micro analysis was performed. Thus, Figures 5.6 to 5.9 and their interpretation are presented above as illustrative support for the overall interaction analysis effort.

Over the three samples, as illustrated in Figure 5.9, the number of contributions by developer and user were quite evenly balanced. However, there were distinct differences in the types of contributions made by each. Clearly, the vast majority of invitational contributions were made by a developer, while the vast majority of user contributions provided information or verification. This does not, however, present a picture of user and developer coming together simply in order that the developer can access what the user knows. As noted by Kensing and Munk-Madsen [1993], user and developer bring different areas of knowledge to the development meetings. Thus, participation by each of them is liable to be constituted largely in contributing information and verification from their respective areas of knowledge.

Since most contributions in the three samples were made in analysing the users' current work situation, the user's participation was primarily constituted in providing information and verification on this, his initial area of knowledge. Similarly, the developer's initial knowledge of areas three and four (see Figure 5.4) and the user's initial lack of this knowledge meant that the developer's participation in designing activities primarily was constituted in providing information and verification.

The lack across the three samples of user contributions to designing an envisioned work situation does not suggest active user participation. Reasons for this lack have been noted above in terms of longer interaction patterns and users did make many contributions to this activity in development work outside the short samples analysed above.

The apparent reluctance of the user to make invitational and direct representation contributions suggests areas in which user participation might fruitfully be encouraged. This reticence suggests that the user was still on far from equal terms with the developer in directing the sessions, as does the user's reluctance (across all meetings and projects) personally to assimilate contributions into an external shared model. In the modelling work, ultimate authority remained with the holder of the pen.

However, as argued in [O'Neill, 1996], user participation does not mean handing control of the project or the development meetings to users. The professional

developer remains responsible for producing the system and for moving the development work along to that end. It might also be expected that users' active participation need not include personally constructing the external shared models. As Kensing and Munk-Madsen [1993] note, while the user brings to the development work knowledge of the users' current work situation, through concrete experience, she does not bring knowledge of how to model that work situation. The developer, on the other hand, can be expected to bring to the development meetings the skills and knowledge necessary to do this modelling.

Similarly, the low numbers of direct representation contributions by both user and developer may be no bad thing. Direct representation contributions are in many ways anathema to cooperation. While verbal contributions are generally discussed, negotiated and agreed before being assimilated into an external shared model, direct representation contributions are placed directly in the model, with no consultation or negotiation on content or form. In the samples here, direct representation contributions were never observed to be open to rejection or ignoring in the way of verbal contributions.

Overall, then, the interaction analysis suggests that the development work in the samples analysed here may legitimately be characterised as participatory. Users and developers did come together. As described in chapter four and in section 5.2.1, various techniques and tools were used to expand users' and developers' areas of knowledge. Users actively made contributions to the development discourse from their initial area of knowledge and invited and received contributions from developers' areas of knowledge. Indeed, they received directly and indirectly from other users extended knowledge of their own work domain. In the samples analysed in detail here and across the video records generally, user-developer meetings were awash with user contributions to the discourse. Indeed, there was very little time or activity in the meetings studied which was not devoted to eliciting, verifying and assimilating contributions, from both users and developers, which directly constituted the cooperative development activities.

5.3 Effective participation: contributing to artefacts

Having established that there was extensive active user participation in the cooperative development sessions, with almost 46% of all noted contributions coming from a user, it remains to ask whether or not this participation was effective. The reader is reminded of the distinction between *successful* and *effective* contributions in section 5.1.1. A contribution successfully has been made when the participants believe that it has been heard and understood by its recipient(s) as intended by the contributor. However, there is little or no point in having contributions from users to the software development process if these contributions do not have an identifiable impact on that process and on its artefacts, including ultimately the software itself. Thus, for our purposes, a contribution effectively has

been made when it has not only been heard and understood but also has been assimilated into these artefacts (see Figure 5.3 above).

Instances are reported by Walz, Elam and Curtis [1993] of successful but ineffective user contributions to user-developer meetings in a software development project. For example, 'most of the information given to the [development] team concerning the nature of the requirements of the system was given orally. And, interestingly, a large amount of this information was lost. A very influential customer ... offered many elaborate scenarios of use to explain his views, needs and preferences. While the designers listened attentively, made comments, asked questions, expressed disagreement, and otherwise interacted with this customer, very little of the information contained in the interactions was recorded' [Walz, Elam and Curtis, 1993, p.70]. In the terms defined in this work, the user successfully contributed to the development discourse, the developers achieving state three for his contributions (see Figure 5.3). But these contributions were ineffective in as much as they were heard and understood but were 'lost' and were never assimilated into the artefacts of the development process.

5.3.1 Contributions and evidence of effectiveness

Two broad types of software development artefacts have been recognised in this analysis: internal and external artefacts (see chapter four). The latter are the notes, sketches, task models, prototypes and so on which are produced as models or representations of various aspects of the development activity objects. Internal artefacts, in the terms of this work, are participants' mental models or understandings of the objects of the development work, of what is represented in the external artefacts and of their coparticipants' internal models.

Thus, a contribution may be viewed as effective if its content has been assimilated into either an internal or an external artefact. Moreover, as described in chapter four, the assimilation of a contribution into one internal model is likely to cause reassessment and assimilation regarding the recipient's other, related internal models. Also, participants' updated understandings, as represented in their internal models, may subsequently be assimilated into external models. Similarly, assimilation of a contribution into an external model is also generally accompanied by corollary assimilations in participants' internal models.

There are at least four types of effective contribution which a participant may make in cooperative development activities. First, the participant may make a contribution to CG(O) which is quickly and explicitly assimilated into the content of an external model or artefact. This may be a direct representation contribution or may involve an informing or verifying contribution which is assimilated into an external model. A direct representation contribution, by definition, is assimilated immediately into an external model. An informing or verifying contribution may be assimilated into an external model on termination of an interaction sequence which may or may not include negotiation about the content of the contribution. (See Figure 5.2 and

chapter four.) In the interactions studied, it was occasionally possible to note the assimilation of a contribution into an external shared model some time after the termination of the corresponding interaction sequence.

Secondly, the participant may make a contribution to CG(O) whose content is assimilated into the internal models of the recipient but which is not assimilated into an external model. Evidence of such assimilation may appear in subsequent verbal and nonverbal communications by the recipient or in subsequent assimilation into external models. Since, however, there may or may not be explicit acknowledgement or evidence of the contribution's original provenance, it is practically impossible for contributor or analyst to establish the effectiveness of many such contributions. Thus, some contributions to CG(O) may have effect downstream which is impossible to trace. An assimilated contribution may lurk unevicenced in a participant's internal models, influencing development processes and artefacts in unseen ways, explicitly appearing in external artefacts only at much later points in the development work. For example, the senior developer in the CI project remarked on several occasions that he had the task models developed in cooperation with the user in mind frequently during later design activities even if they were not explicitly referred to.

Thirdly, the participant may make a contribution to CG(P) which is quickly and explicitly assimilated into the form of an external model or artefact. As with the first type, the effectiveness of such contributions is relatively easy to assess. For example, in the following exchange D2 has just spent some considerable time proposing and describing the use of JSD notation [Jackson, 1983] for modelling the users' work situation. The user rejects the JSD notation for the modelling and D1 proposes another notation. It is straightforward to trace the subsequent use of this latter notation and the absence of the former.

CS21614

D2: Perhaps if you're more comfortable with that, we could just use that.

U: Uh

D2: If it would make things clearer.

U: Well, I mean, it just seems to me, sorry first impression, that that might work beautifully with something that's printed but it's perhaps going to be a little bit hard to spot, these teeny circles

D1: Yeah.

U: in the top right hand corner seem to be just scribbles don't they?

D1: An altern an alternative to that, a simple alternative in terms of what you're saying, in making it clear, because it's ah it's using the same idea but basically just making it bigger so you can notice it

U: Right.

D1: is to label the arcs.

Fourthly, as with the second type of contribution above, some contributions to CG(P) may be assimilated into internal models but not have an apparent effect on an external model until much later, if at all. Again, the effectiveness of such contributions is impossible accurately to assess.

Indeed, many contributions to other aspects of CG(P) are less easy to assess in terms of their effectiveness because, while contributing to common ground about the joint activity and so having a real effect on CG(P), they may have no representation in the external models of software development. Such contributions include, for example, those which affect relations amongst the participants or how the cooperative development activities are carried out. The effectiveness of such contributions falls outside the scheme defined here, as does an analysis of the secondary development activities to which they contribute. Also outside this scheme are contributions which form an invitation. Such contributions cannot *per se* be assimilated into the development artefacts which we are considering here, although they may adequately fulfil their purpose in the development discourse and, in that sense, be considered effective.

Any attempt to assess the effectiveness and consequences of a particular user contribution to the software development discourse is assailed by difficulties of traceability [Gotel and Finkelstein, 1994] and confounding variables. However, contributions to CG(O) and to CG(P) which quickly and evidently are assimilated, respectively, into the content and form of an external model may serve to assess the effectiveness of user contributions to the development discourse and are taken here as the primary measure of such effectiveness. External models, by their nature, provide concrete evidence, both for the participants and for the analyst, of the assimilation of contributions. Internal models, on the other hand, afford no direct access to coparticipants or analyst. Participants and analysts alike must rely on indirect evidence of the assimilation of a contribution into the recipient's internal models. Thus, the much less easily traceable contributions to the content and form of internal models were noted when possible during the analysis, but are not reported as a reliable measure.

As noted above, contributions to CG(O) may be assimilated into the *content* of external models and contributions to CG(P) into the *form* of external models. There are two main ways in which this explicit assimilation may occur.

- A direct representation contribution, as described in section 5.2, inherently results in its assimilation into an external model. Here, evidence of assimilation is explicit and direct.
- An interaction sequence, as described in chapter four, may terminate with the assimilation of an informing or verifying contribution into an external model. In its simplest form, this may involve the presentation of a contribution, its acceptance, perhaps signalled by a brief verbal acknowledgement, and its

immediate representation in the external model. A more complex instance may include negotiation and recursive invitations, or side sequences, with the eventual termination of the top level sequence accompanied by the representation of the contribution in the external model.

Assuming that a contribution to the discourse has successfully been made, there are two ways in which this contribution may yet be prevented from being effective. In one way, the recipient may simply *ignore* the content of the contribution. This corresponds, in the terms of Figure 5.2, to the recipient's simply terminating the interaction sequence without engaging in its last three subprocesses: model matching, negotiation and assimilation. In another way, the recipient may explicitly *reject* the content of the contribution. In this case, the rejection generally follows acceptance of the contribution and model matching by the recipient. Rejection out of hand corresponds then to terminating the interaction sequence without either negotiation or assimilation. More usually, an expression of rejection may be followed by negotiation which may or may not elicit the agreement of the contributor to the rejection.

Incidents of recipients' ignoring or rejecting contributions may illuminate the political and social relations amongst the participants. To render another's contribution ineffective, by ignoring or rejecting it, is a strong statement of power distribution amongst the participants. Indeed, there are widespread social customs surrounding such ignoring or rejecting.

Very explicitly to ignore another's contribution, for example by making a completely irrelevant next contribution, is generally taken to be disrespectful and rude. Exceptions to this are usually given to those considered outside the bounds of peer discourse, for example those wielding extreme power or suffering an illness. Conventionally respectful participants, however, may mask their ignoring a contribution by presenting an explicit acknowledgement of it. A danger in this is that the contributor (and the analyst!) may take the mask at face value as evidence of acceptance and assimilation of the contribution. Indeed, this effect may provide a mask with its utility. A brief 'Yes' or 'Got it' in response to a contribution is not likely to provoke negotiation and therefore is likely to bring the current interaction sequence to a quick termination.

Instances of masked ignoring are difficult to detect precisely because the main evidence available to the analyst is the acknowledgement presented to the contributor and a mask serves, intentionally or not, to deceive the contributor that her contribution has been effective when it has not. This is particularly true when the contribution may have been assimilated only into an internal model.

Rejecting a contribution, in this terminology, implies that the contribution has at least been accepted and considered. However, as with blatantly ignoring a contribution, explicitly rejecting a coparticipant's contribution out of hand, without negotiation, may also be viewed as disrespectful and rude. It also evidences and

reinforces a marked power imbalance in favour of the recipient of the contribution. Instances of rejection out of hand, as with instances of ignoring contributions, may be masked by apparently positive acknowledgements from recipients. This results in similar analytical difficulties in identifying rejections.

If a recipient does not wish to appear unduly rude or powerful in rejecting a contribution, a constructive alternative to masking the rejection is to make an explicit rejection which allows for negotiation. Such negotiation may even result in the contributor's agreement to the rejection. On the other hand, negotiation may result in the assimilation of the contribution, with the recipient withdrawing her rejection. An interesting example of this occurred in a cooperative paper prototyping session in the CI project. The senior developer proposed having a button which could be clicked to perform a particular task. The user and the other developer voiced objections to this design. Negotiation resulted in the senior developer withdrawing the proposal, accepting its rejection. An interesting follow up to this example is that a later, separate discussion in the same meeting resulted in amendments to CG(O) about the task in question which suggested to all the participants that the senior developer's original proposal had been a good one. Further negotiation resulted in the proposal's acceptance and the participants' agreement to the rejection of their earlier objections.

Even without an explicit assimilation of the contribution as a result of negotiation, it is possible that the negotiations themselves have a real impact on the current or future course of the development work. Again, this is a function of the effectiveness of a contribution which is extremely difficult to trace.

Rejection of a contribution after negotiation which failed to elicit the contributor's agreement for the rejection signals a power imbalance in favour of the recipient which may be ameliorated but not disguised by the sensitivity with which the negotiations are conducted. The professional developer who wishes to practice cooperative development should be aware of this in rejecting user contributions. In the projects studied, several instances were observed of developer's rejecting a user contribution, ensuing negotiation and the confirmation of rejection. The negotiation typically allowed user and developer to draw out their positions and the confirmation of rejection was generally handled sensitively.

5.3.2 Effectiveness of user participation in the projects studied

Direct representation contributions by users were very rare in the cooperative development sessions studied. User were very forthcoming with information, proposals, requests and so on, freely using the shared external models to frame and to support their contributions, but showed a reluctance directly to manipulate the shared external models. This extended to an unwillingness even to mark up external shared models to assimilate contributions which users themselves had made. A frequent result of this unwillingness was complicated exchanges in which a user described to a developer what she wanted to represent and the developer

made the corresponding amendments and additions to the relevant external shared model. In these cases, it often seemed that it should have been more efficient for the user to have made the amendments directly. The following example is from a situation in which the author (D2 here) attempted to encourage a user to do so. U's opening speech here is just the latter part of a long sequence in which U made contributions verbally and D1 explicitly assimilated them into the external shared model (TM2).

C121634

U: Every document in effect will go through him eventually. [*U starts pointing to TMs and reading them out*] Assess all documentation. Delegate reading. Actions, yeah he can raise actions. [*D1 begins writing on TM*] Cause actions, yeah yeah, that's right. [*U stops pointing at TMs*] [*D1 stops writing*] Actions raised

(1) [*U points to TM*] um. He could he would probably do SI SIs statement indexing checks on documents as well

(2) so [*U points to TM*] check cross referencing. You could put a box for check cross referencing.

D2: Do you want to. [*D2 hands pencil to U*]

U: [*U begins writing on TM*] Check cross ref, plus, SI. [*U stops writing on TM*]

D1: What's that mean?

U: Statement indexing.

D2: Statement indexing.

U: So he can get a statement through. Process that statement to see what it's linked to.

D1: Would he uh use the word audit there?

U: [*U hands pencil to D1*] Not really no.

At (1), the user makes an informing contribution about a particular task in the work situation. At (2), he proposes the representation in the task model of a corresponding node and even points specifically to where he wants the node to appear. When invited by D2 to amend the task model, the user quickly wrote the amendments on the wall chart and handed the pencil to the senior developer (D1), relinquishing the tool, and with it the role, of 'model amender'. It is interesting, too, that U does not return the pencil to D2, who had provided it, but to D1 who had been marking up assimilated contributions in the task model and who still held his own pencil.

An unwillingness personally to amend external shared models characterised users across all the projects. In the following example from the CS project, a user again is voluble in making contributions and in describing how the external shared model should be amended to assimilate the contributions. And again the user relinquishes the drawing tools, in this case a whiteboard marker and cloth, as quickly as possible when invited to make a direct representation contribution.

CS21610

U: Yeah I mean you could [*U moves forward to point at TM*] in a sense say that these two kind of are subdivisions of the help desk person not being able to solve the problem themselves.

D1: Yeah. Right.

U: So you might want to draw that as one line coming out of that box and then splitting into a second box. [*U stops pointing; takes step backwards; D1 takes step forwards*] Maybe.

[*D1 offers marker to U*]

D1: Mmm.

U: Ah. [*U takes marker and begins to move forward to TM; D1 picks up cloth and moves quickly forward; U moves back*]

D1: Ah. [*D1 erases part of TM*] So. Yeah.

[*U steps forward to TM while D1 still erasing; D1 stops erasing and moves back; U begins to draw lines on TM*]

D1: A-and this

[*U writes labels on lines*]

U: Where's the cloth? [*looks around quickly*] Can I borrow the cloth?

[*D1 hands U cloth; U erases and rewrites label*]

D1: Right. So. If the problem's not solvable it's either never solvable so just go away

[*U puts cloth and marker on desk and steps back*]

As in the previous example, the user is happy working with and interpreting the external shared model and specifically describing what and where amendments should be made. Again, too, he is uncomfortable actually making these amendments. Kensing and Munk-Madsen [1993] observe that users bring to a software development project concrete knowledge of their work but must acquire, through communication with developers, knowledge of how to model that work (see Figure 5.4). This research suggests that even when they have acquired relevant modelling skills and knowledge and can display them, they still may not feel empowered to utilise them.

As noted previously, no external shared model of requirements was used or maintained in the cooperative development sessions. Thus, contributions to requirements analysis were often recorded in other external forms, typically as informal notes. The senior developer in one project frequently used the external shared model as an external repository for contributions to requirements analysis, recording such a contribution on a task model chart next to the task, object or role to which the requirement related. In this way, the task model provided a structure for the requirements. As with contributions to other development activities, contributions to requirements analysis typically were explicitly represented in an

external shared model on termination of the interaction sequence in which the contribution was made.

For example, in the following interaction, the participants have been discussing the users' current work situation. A requirement for an improvement in the new system arises and on termination of the interaction sequence D1 records this requirement on the external shared model (TM1) beside the task which is required to change in the new system.

C113245

D1: What what would you, what is really required? Just those people who've had documents registered to them

U: Everybody.

D1: Or everybody.

U: Everybody.

D1: So

U: Every new entry on the system should be PNCed.

D1: So that every nominal that you enter in the system

U: At the moment

D1: is PNCed.

U: Yeah, at the moment those people do not, who are not authors of documents, right, are not getting picked up

D2: So

U: or are being difficult to pick up.

D1: Mm.

U: Now eventually most do end up having some kind form of document registered to them.

D1: Mm.

U: but not not all.

D2: Would it be better if you could pick up everybody at that stage?

U: Yes. Yeah. Because [*moves quickly forward to point at TM; D1 turns to look at TM*] let's be fair if I mention [Joe Bloggs]⁶ in my statement, I get registered here. [*turns head to look at D2; continues to point to TM*] Right? And we'll say it's a it's a uh [*stops pointing, moves back, continues to look at D2*] a very type, the MO is really unusual, duh-duh duh-duh, y'know, a one off. [*moves forward and points to TM*] [Joe Bloggs] doesn't, has a conviction for this type of offence, the exact MO, y'know everything every minute detail, and we don't pick it up for five days or a week possibly, because he's got to go right through this system actioned right round here. Now if you pick it up here, you could save a lot of time. It would certainly it would certainly alter the type of action that would be raised.

⁶ A pseudonym used here instead of the developer's name which was used in the interaction.

D1: Yeah.

U: It might make say it might make them raise a high priority action, saying hey let's get this guy seen and eliminated, or put into the or y'know out of the frame very quickly.

D1: Mm. So. [*starts writing on TM*] There's a change there isn't there? [*D2 gets up and moves to D1's side at TM char*] Uh. All nominals

D2: Mm.

D1: [*speaks very slowly as he writes on TM*] not just nominals registered to documents.

As described in chapter four, an interaction sequence is typically bounded by verbal and physical references to a shared external model. In many cases of user contribution of information within the interaction sequence, the terminating reference involved the explicit representation of the new information in the shared external model. Similarly, in cases of users' validating the shared external model, where the contribution disconfirmed and suggested amendments to the current representation, these typically were assimilated into the model on termination of the referring interaction sequence.

In cases of validation which *confirmed* the extant representation, the verbal termination of the interaction sequence typically signalled acceptance of the contribution although no explicit amendment to the representation of the shared external model was required. Moreover, such contributions frequently did not simply confirm the extant representation but elaborated or expanded upon it. Often in these cases acceptance of the contribution, with its new or extended information, was explicitly signalled but no amendment was made to the shared external model. It may be inferred that, instead, the participants amended their respective *internal* models of what the shared external model represented to take account of the extra information.

On the other hand, it may be the case that the recipient's acknowledgement of the contribution was a mask for ignoring it. This may, in the worst case, lead to the participants' holding very different understandings of what the shared external model represents. The potential for such divergent interpretations of the data render the assimilation of contributions into participants' internal models difficult to trace with confidence. Again as a more detailed and illustrative support for the overall analytic observations, specific results are presented below for the effectiveness of user contributions in the three samples of interaction analysed in section 5.2.

In the sample of Figure 5.6, a user made three direct representation contributions. As described previously, such contributions are effective by definition, since they are assimilated immediately and directly into an external shared model.

There was also a total of twenty-five user contributions which provided information, twenty-two to current work situation analysis and three to requirements analysis. Of this total, twenty contributions (including the three made

to requirements analysis) were assimilated into a developer's internal models, three were explicitly assimilated into an external shared model and two were ignored.

In the same sample, there were fourteen user contributions which provided verification of existing understandings. Twelve of these contributions confirmed the existing understanding and two refuted it. Both of the latter two contributions were assimilated into an external shared model.

Of the twelve user contributions which verified and confirmed extant understandings, nine were explicitly assimilated into a external shared model (eight into a task model and one into an informal list). A developer provided evidence of having assimilated the other three into his internal models. None was ignored or rejected.

In addition to the contributions to CG(O) recorded in Figure 5.6, five user contributions to CG(P) were observed in this sample. Two of these verified and confirmed a developer's use of terminology and appeared to be assimilated into the developer's internal models. Two proposed notation, one of which was explicitly assimilated into an external shared model (TM1) and one ignored. One was an invitation for information for which the recipient achieved state three (see Figure 5.3), responding appropriately.

In the sample of Figure 5.7, no direct representation contributions were made by a user. There was a total of nine user contributions which provided information, six to current work situation analysis, two to requirements analysis and one to software design. All nine contributions were assimilated into a developer's internal models.

In the same sample, there were four user contributions which provided verification of existing understandings. Three of these contributions refuted the existing understanding and one confirmed it. All four contributions concerned a single topic. There was evidence of their having been assimilated into internal models by a developer but no evidence within the sampled interaction of their assimilation into an external shared model.

Three user contributions to CG(P) were observed in this sample. One of these invited verification that another user had already covered a topic. This invitation was followed by a relevant response, evidence that the recipient had reached state three for the invitation. The other two each suggested going back to that other user to seek further verification. These two also appeared to have been understood as intended and influenced the development work situation, curtailing discussion of the topic in the ongoing meeting.

In the Figure 5.8 sample, there were no direct representation contributions made by a user. There was a total of twelve user contributions which provided information, five to current work situation analysis, six to requirements analysis and one to software design. The five contributions to current work situation analysis were

assimilated into a developer's internal models, as were two of the contributions to requirements analysis. Two of the contributions to requirements analysis were explicitly assimilated into an external shared model (a paper prototype). The remaining two contributions to requirements analysis were ignored, as was the contribution to software design.

Nine user contributions provided verification of existing understandings. One was of the users' current work situation and was assimilated into a developer's internal models. Three were of requirements and were also assimilated into a developer's internal models. Five were of an envisioned software design, three of which were assimilated into a developer's internal models and two into an external shared model. Only one of these verification contributions refuted the existing understanding, all the others confirming it. This one was one of the contributions to software design which was assimilated into an external shared model.

There was also a user contribution inviting information on an envisioned software design and one inviting verification of the same. Both these contributions elicited appropriate responses, evidencing their achievement of at least state three.

Two user contributions to CG(P) were observed in this sample. One stated an assumption about the way the design work was conducted. The other confirmed that the user had not been present at a previous discussion of a topic. Both of these contributions were understood by the recipient who provided appropriate responses in each case.

So, across the three samples we find ten user contributions to CG(P). The interaction analysis suggests that six of these were understood as intended by their recipient, one was ignored, one was assimilated into an external shared model and two were assimilated into a developer's internal models.

There was also a total of three direct representation contributions and two invitational contributions by a user. The former three were, obviously, assimilated directly into an external shared model. The latter two were both understood as intended and drew appropriate responses.

There were seventy-three user contributions to CG(O) across the three samples which provided either information or verification. Five of these were ignored, fifty were assimilated into a developer's internal models and eighteen were assimilated into an external shared model.

At first sight, we are presented with a very high rate of effectiveness of user contributions, as measured in terms of assimilation into development artefacts, internal and external. Certainly, very few user contributions were ignored and none was explicitly rejected. It is, however, of some concern in assessing effectiveness that assimilations into internal models outnumber those into external models by almost 2.4 to 1 (52 to 22, including CG(P) and direct representation contributions).

This finding may be explained by the patterns of user-developer discourse. As described in chapter four, more complex interaction sequences may involve side sequences. Contributions made in these side sequences are often assimilated into the participants' internal models as they are made. On termination of the interaction sequence, a single act of assimilation into an external shared model may subsume several contributions, thus registering one external assimilation against several internal assimilations.

Another important factor which may aggravate this effect is that any form of representation used for an external shared model constrains what may be represented in it [Palmer, 1978]. Contributions were at times made to development activities which were not directly supported by the available external shared models. In some cases, a representation of the contribution could be appended to the model. An example of this was described above in which contributions to requirements analysis were appended to a task model. However, in many cases, the available external shared models did not lend themselves to the assimilation or appending of contributions made, particularly within complex recursive interaction sequences. Common examples of this occurred when software design activities, with software prototypes as the active external shared model, included contributions to requirements analysis and users' work situation analysis. Whilst such contributions could readily be assimilated into the participants' internal models and might influence future versions of the software prototypes, they were not, indeed could not be, assimilated into the prototype in use.

At best such contributions might be assimilated into a subsidiary external shared model, such as public lists or notes, or into a participant's personal notes. However, the analysis here suggests that participants generally committed such contributions only to internal models, seemingly relying on elements of the external shared model to trigger recall of such contributions. A similar approach appears often to have been adopted when contributions which validated and expanded on the contents of an external shared model were assimilated into participants' internal models both of the development activity's object and of what the external shared model represented about that object, but the external shared model was not then amended to represent the changes or greater detail.

Thus, the concern in assessing effectiveness is raised not *per se* because the ratio of internal model assimilations to external model assimilations is high, but because it is less reliable to infer assimilation into an internal model than to observe assimilation into an external model. At times one may confidently make such an inference from the record of discourse. For example, in the following exchange the developer D wants to know where a particular customer enquiry would go in the users' current work situation. The user U says 'liaison', mentioning this role for the first time. It may be inferred that D assimilates this contribution into his internal models because he then relates the liaison role to the role of 'administrative people' in his extant model of the work situation. U then verifies this relation with 'Yeah'.

CS21543

D: Right. Who would that go to?

U: Uh liaison.

D: Right. Who does, uh, that's the administrative people.

U: Yeah.

On other occasions, however, it is difficult to say with confidence that a user contribution has been assimilated into a developer's internal models, as opposed to the developer's simply having reached state three for the contribution. Consider the following example.

CS41417

U: I don't see at the moment, unless other people suggest it to you that there's any reason to [pause]

D: No

U: try to set up a form that as it were prompts you to go through a script of now ask the customer this question.

D: Yeah. Right. Uhh, let's, there's no point in breaking down stuff into detail that isn't relevant.

Here, the participants are trying to establish whether or not there is a requirement for the software to support a list of questions asked by the user. The user U proposes that the software should not prompt the user through a list of questions. It is difficult to judge on the basis of D's immediate response whether he has assimilated this contribution into his internal model of the requirements or has simply understood the contribution. Analysis of later interaction and subsequent design efforts support the conclusion that D has assimilated this contribution. However, it remains in general difficult to trace the provenance of a contribution over time.

In contrast, a contribution may at times appear clearly to have been assimilated into a recipient's internal model, only for later evidence to suggest otherwise. Here we have a seemingly clear case of D's internal model of the users' current work situation being revised to assimilate a user contribution. The participants are discussing the types of customers which users of the system have to serve.

CS61747

D: Those elements, you've got customer name, and login, department, and contact, and staff or student.

U: With student you're supposed to record whether it's undergraduate or postgraduate.

D: What's the word? Type?

U: Yes.

D: Three?

U: There is of course the quibble there that we do deal with people from outside the College so you can have somebody who is none of those three.

D: OK. Right. So there is at least a fourth category.

U: Mm.

It seems clear that D has assimilated into his internal model of the work situation that there are at least four types of customer. However, less than ten minutes later, in the same meeting, we get the following exchange between the same two participants.

CS61755

D: So uh maybe we could use that. Customer type again. Just three types?

U: Well no four because there's other

D: Yeah.

U: when they're not actually a member of the College.

Hence, once again the longer term patterns in the interaction reveal the difficulties in reliably interpreting samples of interaction in development work. With this *caveat*, however, meetings, supported by the detailed figures for three short samples, suggests that the majority of user participation was effective. Many user contributions were assimilated explicitly into the shared external models or artefacts of the development activities, including task models and prototypes. These contributions were made to both CG(O) and to CG(P). In many cases, it was possible to infer the assimilation of user contributions into the internal models of coparticipants, again contributing both to CG(O) and to CG(P). Very few user contributions to the development discourse were observed to have been flatly ignored or rejected.

The assessment of the effectiveness of user contributions provided by this analysis may in fact be an underestimate. In addition to direct representation contributions and assimilation into external models on termination of the interaction sequence containing the contribution, contributions may have been assimilated into the corresponding external model at some later time in the development process. This later assimilation may not have been traceable by the analyst to its originating contribution and so may not have evidenced the effectiveness of the contribution. Also, a user contribution may have been assimilated into an external artefact other than the one on which the interaction sequence, and subsequent analysis, was focused. For example, a contribution to requirements analysis in an interaction sequence referring to a task model may have been assimilated not into the task model itself but into a note or sketch or other external artefact which subsequent analysis did not trace to the originating contribution. Finally, as noted above, it was not possible comprehensively to trace the assimilation of user contributions into coparticipants' internal models.

5.4 Conclusion

This chapter addressed research questions two (were the projects studied *participatory*?) and three (was user participation in the studied projects *effective*?). It presented a definition of participation in terms of making contributions to the discourse and a definition of the effectiveness of participation in terms of the assimilation of contributions into the artefacts of the development process. The chapter then used these definitions in analysing the two software development projects for the extent and effectiveness of user participation.

In answer to research question two, the projects may legitimately be characterised as participatory. 45.6% of the 171 contributions made in the analysed samples of development work were user contributions and 54.4% were developer contributions. Taking into account that leadership of the projects remained in the hands of the developers, this distribution suggests a balanced, participatory interaction between user and developer. Most of the contributions came from the areas of knowledge and skill which users and developers' brought to the projects. But as the development work progressed, both users and developers made contributions based on the knowledge and skills they had acquired through their participation. For example, users made contributions of work and software design proposals while developers made contributions of target work domain knowledge.

In answer to research question three, user participation appears to have been highly effective in terms of having contributions assimilated into the internal and external models produced through the development work. 93.2% of user contributions to CG(O) were assimilated in the three samples of user-developer interaction. However, only 13.1% of the total CG(O) contributions were observed to have been assimilated into external models. 68.5% were assessed to have been assimilated into developers' internal models and, as noted above, this inference must be less reliable than the direct observation of external models.

The contributions of this chapter include both a further detailed analysis of user-developer interaction in particular participatory software development projects and a framework of contribution types (see Figure 5.5) which other researchers may use to investigate other such projects. This framework may also be applied in conjunction with coding schemes other than that based on the four types of development activity used here.

The chapter also provides lessons for software development practitioners. One such lesson is that in 'participatory development' projects a user may successfully make a contribution (i.e. have a contribution heard and understood) which nevertheless is not assimilated into the professional developers' understanding, the common ground of the participants or the external development artefacts. It may be profitable to investigate means of capturing these lost contributions - perhaps through video records of development meetings - and feeding them back into the development process.

Another lesson for practice is that basing a meeting around a form of model which supports the declared development activity of the meeting, for example a task model when the development meeting is called to analyse the users' work situation, may inhibit the recording of potentially significant contributions which are made in the meeting. It may be more effective to focus a meeting on the relevant form of model but also to have permanently visible, and accessible for changes, other forms of representation and structuring. For example, when prototyping it might be productive also to have available modifiable task models and requirements specifications in which to assimilate contributions to work situation analysis or design and requirements analysis.

This chapter distinguished between successful and effective contributions. It is also important to distinguish between effective and valuable contributions. A contribution may successfully have been made and effectively have been assimilated into external and internal artefacts and may yet not be a valuable contribution. An effective contribution may prove to lack value because, despite being assimilated into earlier artefacts, it failed to be assimilated into or to affect the resulting software. An effective contribution may have a positive value if it has a beneficial effect on the resulting software or a negative value if it has a detrimental effect. The following chapter takes up this distinction by examining the influence on software usability of user contributions to the development projects studied.

Chapter 6

User participation and software usability

The preceding chapter addressed research questions two and three, examining the extent and effectiveness of user participation in the studied projects. The work reported in this chapter addresses research question four, asking how valuable user participation was in the studied projects. As noted in chapter two, proponents of PD have in recent years argued that a major reason to employ PD practices is to improve the usability of the systems developed. Hence, this chapter assesses the value of user participation in software development work in terms of its impact on the usability of the resulting software.

Chapter five defined participation in terms of contributions to the development discourse. This chapter analyses the relations between specific user contributions to the development discourse and specific aspects of the resulting software's usability, thereby providing an assessment of the value of the contributions.

If user-developer cooperation in software development really is to influence the software product, the common ground constructed between user and developer during the development process should provide a basis for the software design. In particular, in the terms of chapters four and five, this basis should be provided by CG(O), since this is common ground about the object of the development work, ultimately the implemented software system, while CG(P) is common ground about the development work itself.

Pertinent questions for an analysis of the relationships between user participation and software usability are:

- did a contribution to the development process become reflected in the implemented software or not?
- what was the course of the contribution's fate: how did it come to be reflected (or not) in the software?

- what impact did the contribution's fate have on the usability of the implemented software?

In attempting to answer these questions, this chapter traces the relations between user participation and software usability in two directions, downstream from development work contributions to features of the software and upstream from features of the software to contributions.

As noted in chapter three, usability is rather difficult to define. The definition of usability used here or, rather, the *aspects* of usability which are most relevant for the purposes of this research, is in terms of quality of support for the users' tasks.

In order to assess the effects on software usability of the contributions considered, we need to evaluate the implemented software. A usability evaluation of the software reveals issues of task support which may be traced back through the development process. A thorough presentation and analysis of approaches to and methods of usability evaluation is beyond the scope of this work but section 6.1 outlines some approaches to usability evaluation, explains why a particular method, Cooperative Evaluation [Wright and Monk, 1991], was selected and reports the usability evaluations which were conducted, using Cooperative Evaluation, for the CI and CS software.

Section 6.2 summarises the usability evaluation results, providing a starting point for the upstream trace. Section 6.3 introduces the work of tracing the relationships among contributions, development artefacts and software, describing the available data, how the analysis of the data was scoped and how the analysis attempted to trace the course from contributions in development meetings through to specific features of software usability.

Section 6.4 presents the results of the analysis effort to trace the fate of contributions to the cooperative development activities. Section 6.5 discusses these results and section 6.6 closes the chapter, and the main analytic work of the thesis, with conclusions drawn from those results.

6.1 Evaluating the software

As noted in chapter two, software usability is a major topic of research interest and effort within the HCI community. The nature and scope of the topic make it difficult to produce a succinct, explicit definition of usability. Shackel [1991] defines the usability of a system as 'the capability in human functional terms to be used easily and effectively by the specified range of users ... to fulfil the specified range of tasks, within the specified range of environmental scenarios'. Sweeney, Maguire and Shackel [1993] define usability as follows.

Usability is an emergent quality of an optimum design which is reflected in the effective and satisfying use of the IT. ... Usability may be measured by

the extent to which the IT affords (or is deemed to be capable of affording) an effective and satisfying interaction to the intended users, performing the intended tasks within the intended environment at an acceptable cost.

[Sweeney, Maguire and Shackel, 1993, p.690]

Measures of usability are measures of the support which the software provides for the tasks which the user carries out. At the heart of participatory development is the assumption that active user participation in software development should facilitate the integration of task knowledge and user requirements into the software design. Thus, user-developer cooperation in task analysis, task modelling and prototype design should improve the usability of the resulting interactive software. The usability factors most directly influenced by this approach include:

- User motivation to use the system.
- User acceptance of the design.
- Task fit - the level to which the task represented by the interface matches the task understood by the user and the task supported by the system.
- Task coverage - the level to which system facilities cover all the users' tasks.
- Context fit - the interaction between user and system matches environmental demands.

There are many published comparisons of methods for evaluating software usability (see, for examples, [Karat, 1994; Karat, Campbell and Fiegel, 1992; Karat, 1988; Nielsen and Phillips, 1993]). Sweeney, Maguire and Shackel [1993] note that the choice of evaluation method is influenced by factors such as development schedules, available resources and access to users. With some variation, most writers suggest three main approaches to usability evaluation. The first encompasses formal modelling and theory based evaluation. The archetype here is predictive evaluation from a GOMS model of the system [Card, Moran and Newell, 1983; Olson and Olson, 1990]. The second approach is subjective evaluation by a usability 'expert'. This approach includes, for example, scenario-based evaluation [Nielsen, 1995], evaluation against heuristics [Nielsen, 1994], guidelines [Mosier and Smith, 1986; Smith, 1986] or standards [ISO 9241-11] and cognitive walkthroughs [Kieras and Polson, 1985; Lewis, Polson, Wharton and Rieman, 1990; Polson, Lewis, Rieman, and Wharton, 1992]. The third approach involves empirical observation of users working with the software. Data collection techniques may range from user surveys and questionnaires to formally controlled experiments [Karat, 1988].

Theoretical, model based approaches tend to be little used by professional software developers, who often lack the resources or motivation to conduct such studies (see

chapter two and [Bellotti, 1990]). For example, 'GOMS experts are not readily available in most organizations' [Nielsen and Phillips, 1993]. Subjective evaluation has become increasingly popular in recent years because it is quick and relatively inexpensive to apply and is reasonably effective. However, even a staunch proponent of subjective evaluation such as Nielsen concedes that user based testing is the most effective approach to evaluation (see [Nielsen and Phillips, 1993]).

The two most common criticisms of user based evaluation, that it is expensive and that controlled experimental results may not accurately reflect usability in the field, have encouraged the development of user based evaluation methods which may be applied cheaply and easily by non-specialists in a less formal setting. An example of this approach which has achieved widespread use is Cooperative Evaluation [Wright and Monk, 1991].

Cooperative Evaluation was chosen as the evaluation method for assessing software usability in this work. There were several reasons for this choice, on both commercial and academic grounds. From a research perspective, the author was keen to maintain the philosophy and practice of user-developer cooperation throughout the software development projects and so favoured user-based evaluation.

However, a controlled experimental setting for user based evaluation does not encourage the kind of user-developer cooperation which is the focus of this work. Experimental subjects tend to feel powerless in the face of those running the experiments and might better be described as acquiescing than as cooperating.

In addition, there was the continuing desire concretely to ground this research and its results in industrial practice which, as noted above, often demands quick, cheap methods. Indeed, both the main projects studied in this research were pursued within very tight time constraints. In each case, it was important to conduct the usability evaluations and to present their results as quickly as possible. At the same time, there was a lack of resources made available to expend on usability testing in each project.

The research requirements of this chapter also demanded an evaluation method whose results could be (as directly as possible) compared with and related to contributions to the development discourse. Thus, the evaluation results provided by the Cooperative Evaluation method, couched in terms of users' comments about their tasks and requirements, were directly comparable with users' contributions to the development meetings which were in similar form. The results of some other evaluation methods, couched for example in terms of lists of performance times or numbers of errors, were not so suitable.

Finally, Cooperative Evaluation was chosen because it has been widely used and has been established as an effective approach to usability evaluation [Monk, Wright, Haber and Davenport, 1993]. That said, the choice of evaluation method is always

a trade-off and the effectiveness of Cooperative Evaluation as applied in this work may better be judged on the basis of the following sections of this chapter.

The method of Cooperative Evaluation arises from the position that 'a full account of human-computer interaction has to contain both the behaviour of the user with the system and the intentions of the user when taking those actions' [Wright and Monk, 1991, p.893-4]. This motivates the analysis of concurrent verbal protocols [Ericsson and Simon, 1993] by the user. A user thinks aloud while performing a task, describing her goals, how she is attempting to achieve these goals, what she interprets as the current state of the system and difficulties she encounters. This verbal protocol provides a detailed 'intentional context' for the user's actions [Wright and Monk, 1991]. In earlier empirical evaluation work, the same authors found that 'some of the most important usability problems can only be detected in this way' [Wright and Monk, 1989, p.356].

Cooperative Evaluation is interested mainly in two kinds of evidence of usability problems: critical incidents [Monk and Dix, 1987] and breakdowns [Winograd and Flores, 1987]. 'A critical incident is behaviour from the user which is sub-optimal with respect to the intentions of the user and functionality of the system. The designer may have provided an efficient way of performing some task goal but the user chooses to do it in some much less efficient way. Users may take some actions which are redundant or ineffective. They may take actions which take them further away from the desired state than they were when they started so that they have to undertake further actions to back up or undo these unwanted effects' [Wright and Monk, 1991].

Critical incidents may therefore be identified, by an evaluator familiar with the 'official' means of performing a task with the system, as unexpected behaviour on the part of the user. 'Unexpected behaviour is detected when the evaluator's knowledge of the user's goals [and what needs to be done on the system to achieve those goals] suggests a different action to that which the user actually takes' [Monk, Wright, Haber and Davenport, 1993, p.14].

Breakdowns occur when a tool, such as a software system, loses its transparency. In normal use, a tool is transparent in so far as the user is unconscious of the properties of the tool, concentrating instead on the task for which the tool is being used. The tool is no longer transparent if it breaks or is awkward to use. In such cases, the user is obliged to give conscious consideration to the tool itself and its properties in attempting to perform the task.

A verbal protocol from the user is necessary to indicate breakdowns since the user may otherwise silently perform the task in the manner in which the designers intended the system to be used, but find it awkward or laborious. With verbal protocols, when the system is working well, i.e. transparently, the user's comments are at the level of goal-directed tasks (e.g. 'I want to link these people'). A comment at the level of the interface (e.g. 'I can't drag a line between these

icons') indicates that the user's goal-directed work has been interrupted by the need to think in detail about how to perform actions with the system.

So, verbal protocols by the user provide the intentional context for the identification of critical incidents and breakdowns. The elicitation of concurrent verbal protocols in Cooperative Evaluation is more active than in many applications of the approach, in which the subject is asked to 'think aloud' without intervention from the analyst (see [Ericsson and Simon, 1993]). In Cooperative Evaluation, the analyst may ask questions of the user, such as *why did that happen?*, *what will happen if ...?*, *what are you trying to do?*. The user may also ask questions and is generally encouraged to view herself not as a subject but as a coevaluator of the system. According to Wright and Monk [1991, p.896], this 'encourages [the users] to be critical and so maximises the detection of breakdowns'. It is recommended to have two evaluators present, one concentrating on the user's activities and intervening when necessary, the other recording the interaction.

Monk, Wright, Haber and Davenport [1993] present a guide to performing Cooperative Evaluation, based on their experiences of applying the method. The main steps of Cooperative Evaluation are defined as:

- recruit users;
- prepare tasks;
- interact and record;
- summarise your observations.

In this work, the Cooperative Evaluation method was used to assess the usability of the two software systems which were the products of the CS and CI development projects respectively. This section describes how Cooperative Evaluation was used to assess the software in each case. The evaluation followed the stages described above of (i) recruiting subjects to act as users; (ii) preparing a set of tasks for the users to perform; (iii) accompanying each user through the performance of their tasks, noting comments and problems; (iv) analysing and reporting the observations.

6.1.1 Recruiting subjects

6.1.1.1 CI subjects

Due to the commercially sensitive nature and timing of the study, this evaluation was conducted with employees of the development organisation as representative users, rather than people from the intended customer population. Three types of representative user were sought: (i) those with experience of the use of computer based systems to perform the tasks in the work domain; (ii) those with experience of interactive software systems but not of the work domain; and (iii) those with

secretarial skills but no experience of the work domain and very little experience with interactive software in general.

The chief developer for the project drew up a list of potential subjects and sought their agreement via email and personal contact. A pilot study was conducted with the user who had been heavily involved in the participatory development activities. The main study then involved two subjects from category (i) (a third falling ill at the time of the evaluations); seven subjects from category (ii); and one from category (iii).

6.1.1.2 CS subjects

The evaluation was initially scheduled to occur some weeks after the introduction of the new software across the working environment. In fact, for managerial reasons, take up of the system was patchy across the sections of the department with the result that at the time of the evaluation some users were quite familiar with the software while others had barely looked at it. In addition, while most users were very experienced with the work domain and tasks, a few had newly been recruited into the organisation. There was also a desire to involve in the evaluation both users who had been involved in the development activities and some who had not. Some users who had been involved in the development work had subsequently left the organisation (being replaced by the new recruits).

Initial discussions about who might be available for the evaluation work were conducted between the author and the lead user¹. A preliminary list was drawn up of potential coevaluators across all the sections of the organisation. The people on this list were then emailed with an explanation of what was required and an invitation to participate. Some agreed immediately, others agreed after follow up emails or phone calls and still others declined. In the end, nine users were involved in the Cooperative Evaluation sessions. The intention was to have one subject with experience of each of the Help Desk, Technical Query, Fault Repair and Reception roles who had been involved in the development activities and one subject with experience of each of these roles who had not. This was achieved with the exception of the Technical Query role for which both subjects had not been involved in the development work. The ninth subject was introduced to duplicate the running of the uninvolved Help Desk subject as the latter insisted that he was not a representative user.

6.1.2 Defining tasks

6.1.2.1 CI tasks

The results of the evaluation were intended to feed back into the immediate development work on the system. So the chief developer and the author worked together to identify important application areas for which the use of limited

¹As described in chapter three, one person, referred to as the lead user, had originally been charged with developing the system for the organisation and was the main point of contact between the author and the organisation.

evaluation resources could prove particularly valuable. The general area identified was entering data on to the system. The chief developer and the author were familiar with the relevant tasks from their work on developing the system. Three high level tasks identified as important were registering a document on the system, typing a document's textual content on to the system and indexing a document². It was decided to evaluate the system for users working in three different roles:

(i) single user: where one person performs all the work, receiving an incoming paper document, registering it on the system, entering its textual content and indexing it;

(ii) indexer: where the user works with a document which has already been registered and typed on to the system, using the system to index and research the document and then comparing the results with manual mark-ups on a paper copy of the document;

(iii) typist: where the user receives an incoming paper document, performs a basic registration of the document and enters the main text of the document on to the system.

This selection of roles was intended to help evaluate the system's flexibility in being used by a single specialist user in a low resourced work situation and by a combination of a few specialist users and a non-specialist typing pool in a larger, more richly resourced, work situation.

6.1.2.2 CS tasks

Again a primary consideration was to evaluate the usability of the software in supporting important tasks performed in the various user roles. As with the CI project, the selection of tasks for evaluation was informed by the preceding development work.

The system was evaluated for users working in the four roles which the system had been designed to support:

(i) reception: where the user receives queries and requests directly from customers on a wide range of topics and by various media and either provides a solution, if it is a relatively simple request, or refers it on to an appropriate colleague;

(ii) help desk: where the user receives queries from customers, primarily on the use of software packages, provides a solution and the logs the query and answer; queries which the user cannot resolve are passed on to a technical consultant;

(iii) technical consultant: where the user receives specialist technical queries referred by colleagues on the CS system and sometimes directly from users; again a

²Commercial considerations prevent the detailed description of these tasks here.

solution is provided or the query referred to a colleague and a log is entered on the system;

(iv) fault repair: where the user deals with hardware and computer infrastructure fault reports, generally from reception or help desk staff but occasionally directly from customers, repairs the equipment (assuming that it is a matter for repair and that it is repairable) and logs the work.

This set of roles covered the whole department apart from management and inventory keeping work. As noted in chapter three, the elements of the system to support these roles had not been implemented during the project period studied.

6.1.3 Running the evaluations

6.1.3.1 *CI evaluation*

The software evaluated was an early implemented version. Before this evaluation it had undergone only limited testing by the developers. Subsequently, changes were implemented in response to the results of this evaluation before the software went out to customer sites for beta testing.

Evaluations were run over two days to accommodate the availability of subjects. In the preceding two days all subjects were given a two hour training session on the system. This involved a demonstration by the chief developer of the use of the software to perform a range of tasks, provision of a 'user manual' consisting of instructions on how to perform the tasks to be encountered in the evaluation, a glossary of terms, notes on the background to the use of the system, including the surrounding work practices, and up to sixty minutes unaccompanied access to the system.

During the evaluations, each subject was allowed as long as she needed to tackle the set tasks. Each subject was given a fresh copy of the user manual and a task sheet consisting of a numbered list of tasks on a single A4 sheet of paper. The subject performed the tasks in numerical order. Each subject performed a combination of registering, indexing and typing tasks, according to which of the three roles, single user, indexer or typist, she had taken.

At the beginning of a run, the subject sat down at a computer with *Microsoft Windows* running on screen. The subject was first presented with an information sheet reminding them of the purpose of the evaluation. (A copy of this sheet for the CS evaluation is reproduced as Appendix x.) The task sheet was then presented to the subject, the author explained what was required and reminded the subject that the evaluation was of the *software* and not of the subject. The subject was also provided with the paper documents which were to be processed. A sample database of documents had already been entered on the system. The subject was then asked to begin the tasks by starting the CI software from *Windows Program Manager* and continuing.

The author sat on the subject's right, talking with and questioning the user. A member of the development team sat behind the subject, taking notes of the user's interaction with the software and of the user's comments. (After the first two runs, the chief developer was so impressed by the developer's enthusiastic response to hearing first hand users' problems that he decided to rotate all the members of the development team as recorder in subsequent runs.) If the subject fell silent, the author prompted the subject to continue her commentary on what she was doing. If the subject gave up attempting to perform a task or asked for advice, the author asked the subject to suggest what the answer might be. If the subject became completely stuck, the author told the subject how to proceed.

6.1.3.2 CS evaluation

The software evaluated was the first version which was delivered for real use. The software had been made available throughout the users' organisation approximately one month before the evaluation. It had been intended that it should be in active use from the time of delivery. In fact, due to a very low key launch, its uptake had been very patchy. Some of those who had used the software had made various complaints and requests to the implementor in the period before this evaluation. A record of these along with the results of this evaluation provided a basis for updates which were subsequently implemented.

Evaluations were run over a week to accommodate the availability of subjects. Each subject was allowed as long as she needed to tackle the set tasks. Each subject was given a task sheet consisting of a numbered list of tasks on a single A4 sheet of paper. The subject performed the tasks in numerical order. Each subject performed one of reception, help desk, technical consultant or fault repair tasks, according to which role she usually fulfilled in her day to day work.

All subjects were also given free access during the evaluation to a copy of two documents which had been produced respectively by the lead user and the head of the fault repair section. Each document explained how to perform all the necessary tasks with the software. These documents had been made available throughout the users' department in the weeks preceding the evaluation.

At the beginning of a run, the subject sat down at a computer with the CS software running on screen. The subject was first presented with an information sheet reminding them of the purpose of the evaluation. (See Appendix y.) The task sheet was then presented to the subject, the author explained what was required and reminded the subject that the evaluation was of the software and not of the subject. The subject was then asked to begin the tasks from the point at which the CS software had been started.

The author sat on the subject's right, talking with and questioning the user. Another researcher sat behind the subject, taking notes of the user's interaction with the CS software and of the user's comments. If the subject fell silent, the author

prompted the subject to continue her commentary on what she was doing. If the subject gave up attempting to perform a task or asked for advice, the author asked the subject to suggest what the answer might be. If the subject became completely stuck, the author told the subject how to proceed.

6.2 Summary of the usability evaluation results

A wide range of usability issues was revealed in each of the evaluations. This is typical of usability evaluation exercises and it is often useful to categorise the issues in such terms as their severity, the resources needed to address them and the desirability of resolving them [Brooks, 1994; Nielsen, 1994].

The evaluations reported here were conducted within ongoing software development projects and the results of the evaluations were initially reported in that context. Thus, the evaluation reports initially were written for and presented to the developers. The structure of the reports followed the task structure of using the software, each section reporting on issues around a particular task. It was intended to present a second edition structured by the severity of usability problems across the software as a whole, but the developers were happy with the task structured format and used it as a basis for ongoing development activities. A second edition was therefore never presented to the developers.

For each development project, a list of usability issues identified by the evaluation and categorised by severity was produced for the purposes of this research. Since this work assumes that user participation in the task based development work should enhance the usability of the software primarily in terms of the software's support for the user's situated tasks, the usability issues were categorised according to their increasingly disruptive effect on users' task performance. Thus, level one issues were minor irritations to the user which could be easily overcome, whereas level five issues effectively made the task impossible for the user to perform.

This section reports usability issues which came out of the evaluations. Sections 6.2.1 and 6.2.2 present overviews of the usability issues which were identified for each of the two systems evaluated. Specific examples of usability issues which arose from each evaluation are presented for each of the five levels of usability issues. Multiple reports of the same usability issue were rationalised into one, since for the purposes of this work we wanted simply to identify a set of software features and associated usability issues (see section 6.3). It should be noted that usability issues did not always fall neatly into a single category. Usability issues and suggestions for addressing them almost always have consequences for other aspects of usability at various levels. In general, the time and resources needed to address and to resolve an issue increase with the level of severity. With this in mind, where an issue may have fitted at more than one level, a balance was struck between the impact of the issue on overall application usability and the potential

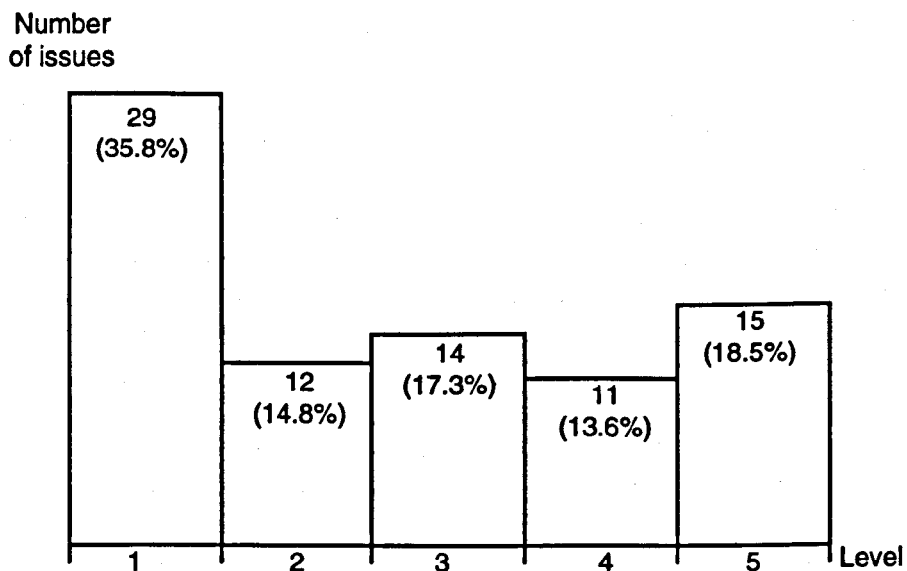
costs of addressing the issue. If the problem was quick and easy to resolve through minor reimplementations, it was placed at the lower of two levels.

6.2.1 Summary of the CI evaluation results

The CI software was generally well liked by users in the evaluation. Novices to the application domain reported being comfortable performing their allotted tasks with the software. Users with experience of the domain were also comfortable using the application and were very excited by what the software could do and how it did it compared to existing systems.

A total of eighty-one usability issues were identified with the CI software. Issues may be something of a euphemism for problems. As noted earlier, usability evaluation methods tend to identify problems rather than positive points. In fact, the management of the development company became quite nervous when they saw the evaluation results containing so many 'issues' and required reassurance that the software design was not a disaster. Figure 6.1 illustrates for the CI software the distribution of reported usability issues across the five levels.

Figure 6.1: CI software usability issues at each level of severity



One factor which provided some reassurance is the distribution of identified usability issues across the categories of severity. 35.8% of the issues were at level one, the most trivial level. The least serious and most easily resolvable of these issues included, for example, users preferring 'OK' to 'Ok' on a button. Slightly more complicated was the issue of users preferring a page number to a percentage as an indication of the distance of their currently visible text from the start of a document. One of the most serious issues at this level was the use of an icon like 'X' to represent 'cross-reference'. Many users initially interpreted it as either

'error' or 'delete', causing some confusion. Even this example, however, was easily overcome and did not substantially hamper the use of the software.

14.8% of the issues were at the slightly more severe level two. An example of an issue here was the poor labelling of a drop-down 'navigation box'. The idea was for users to select from it a part of the system which they wished to use. However, rather than meaningful entries, the user was confronted with a list simply saying 'Section 1', 'Section 2' and so on. This was wholly useless to users who had yet to associate these numbers with particular aspects of the system and imposed an unnecessary cognitive load in making the association.

Level three accounted for 17.3% of usability issues. One feature of the software which raised an issue at this level was the facility to search a database for, amongst other things, events which had been recorded along with the time at which they had occurred. The list of 'hits' returned by a search for any other entity was presented to the user in descending order of fit with the search criteria. The list of hits for event searches was presented, with no indication that this was the case, in descending chronological order of when the events had taken place. This proved to be quite misleading for users.

There were 13.6% of the usability issues at level four. A usability issue was raised here by the system's most technologically advanced feature: an automatic document parsing and analysis facility. A structured group of linked icons was presented to the user representing objects which had been identified in the analysed document. A design intention was that clicking on an icon transferred the user to the piece of text in the document which had given rise to the identified object. This worked well with some icons but, to the users' frustration and bafflement, nothing at all happened when other icons were clicked. The cause of this inconsistency could be traced back to the document's having two parts: a set of structured fields giving, for example, information about the author and a free text area. The linking worked for icons created from objects identified in the free text but not for those from the fields. There was no way to distinguish these two types of icons other than by clicking on them and then discovering whether or not the linking worked.

Level five usability issues made up 18.5% of those identified for the CI software. One example was a feature of the task management facilities of the software which users found incomprehensible. As described in section 3.2.2 of chapter three, users worked with a 'task list' which provided their work agenda. Task lists typically had many columns presenting attributes of the task-object pairings with which users worked. They could also have a very large number of tasks (the rows) in their task list. In order to specify which columns should be visible and how to order the rows, the user had to enter a so-called Query mode. When this was selected, the task list disappeared to be replaced by what looked like a large, blank spreadsheet. There was no indication how this was to be used. In fact, the user had to type the sorting criteria of his choice into some of the uppermost rows of the

'spreadsheet'. The remaining large number of rows was entirely redundant. Users were simply baffled by this aspect of the software.

6.2.2 Summary of the CS evaluation results

As noted above, the software developed in the CS project had been released to users a month before the usability evaluations were held. Users presented a wide range of reactions to the software, both from their experience of using it for real and from the evaluation runs. A few users saw it as potentially very useful in supporting their work and were keen to take it up. A large number of users was unwilling to use it. Three main reasons contributed to this reluctance.

First, there was a number of poor usability features, the sources of which are elaborated later in this chapter. Of themselves, these usability issues discouraged use of the system to some degree. But users could have been expected to overcome such reluctance if they had been highly motivated to use the system. Unfortunately, the opposite was the case.

The second reason for the quite poor take up of the system in the month preceding the evaluation was a lack of support for its delivery. In fact, the system was not so much delivered as abandoned into the department. There was no introduction to or training on the system for its intended users. These omissions engendered a feeling amongst would be users that the system was of no significance and they could ignore it.

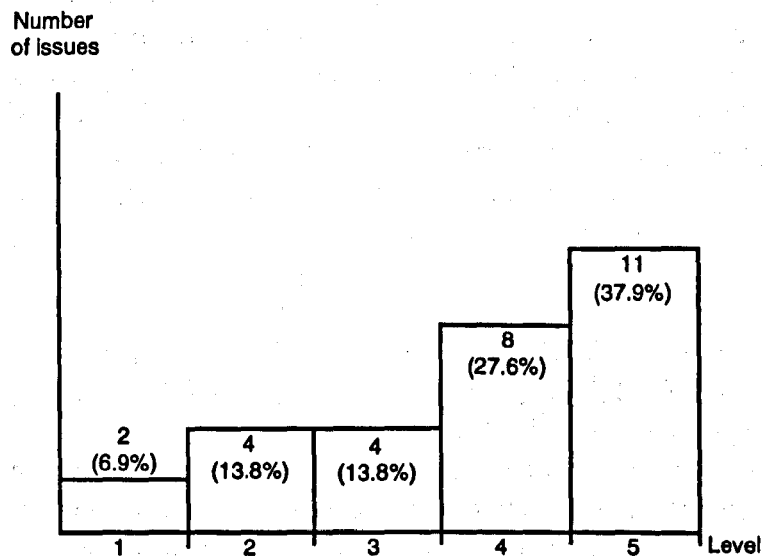
The third factor contributing to users' not embracing the system was an antipathy amongst many of them to recording their work. Most users wanted the form filling aspects to be as brief as possible. They saw their job as solving customers' problems, not keeping records. Many of them regarded the availability of a database of past queries and solutions as very valuable in supporting their work but there was little willingness to invest time and effort in building up that resource.

A total of twenty-nine usability issues were identified through the evaluation sessions for the CS software. Figure 6.2 illustrates for the CS system the distribution of reported usability issues across the five levels of severity.

There were substantially fewer usability issues identified for the CS software than for the CI software. This does not, however, imply that the CS software had much less severe usability problems than the CI software. The lower number of identified issues is due in large part to the relative simplicity of the functions tested. The CI software was built to support a much wider range of more complex tasks than the CS software. In addition, as described in chapter four, two areas of the CS system were not implemented and so were not covered in the evaluation. For this reason it is more useful to describe the numbers of issues at each level of severity as a proportion of the total number of issues identified for that software, rather than as

an absolute number which might suggest a misleading comparison between the results for the CS and CI systems.

Figure 6.2: CS software usability issues at each level of severity



Only 6.9% of the usability issues identified with the CS software were at level one, the most trivial level. The least serious and most easily resolvable of the issues included, for example, the constant presence on screen of a 'query form' which the user rarely if ever used but which could not be removed. This form did not interfere with task performance other than in taking up already limited screen space and irritating almost every user.

13.8% of the issues were at the more severe level two. An issue here was that many users were confused by the sheer number of fields on the forms, some of which were wholly irrelevant to the particular task in hand. For example, most users never read the date and time fields; they expected them to be correct. The 'intended date of solution' field was also generally ignored.

Level three also made up 13.8% of the identified usability issues. One which confused many users was that when they referred a query to a colleague on the system, clicking the 'Submit' button sent the query but did not close the open window. There was no obvious feedback that the query had been sent and some users frequently sent multiple copies of the same query.

There were 27.6% of the usability issues at level four. A very confusing feature was the presence of two forms, Display and Modify, which seemed identical to many users. As their names suggest, one was intended for read-only viewing of a query while the other allowed editing of the query. However, they were virtually indistinguishable on screen and users were not clear about their respective purposes.

Level five issues made up 37.9% of the usability issues. As discussed in detail in section 6.4.2.3 below, Reception added several defaults (extension/phone number, location and room number) to the summary field to make sure these things are asked by the person taking the query:

This suggests that these should be included as standard fields on the reception form. Several users expected a customer phone number field. Users tend to put the customer's phone number in another field and it is not always obvious in which field the phone number has been recorded. Similarly, some users put the customer's email address in the work done field. Phone numbers, email addresses etc were supposed to be put in the 'Method of contact' field.

6.3 Tracing relationships between contributions, software features and usability

In addressing research question four, the work reported in this chapter attempted to trace the downstream fate of contributions to cooperative development activities and the effects on software usability of those contributions. The analysis included tracing upstream from particular features of the software which gave rise to usability issues to determine which contribution, if any, to the cooperative development activities was reflected in that feature. The analysis also included tracing downstream from particular contributions to determine whether or not they were reflected in the implemented software and to examine the effects their inclusion or exclusion had on the usability of that software.

Even more than in previous chapters, the analysis here covered a huge range of heterogeneous data. Available sources of data included the video records of the cooperative development meetings in the CI and CS projects. There were also the external artefacts produced as a result of the development activities. These artefacts included the formal public artefacts of the cooperative development process, task models, prototypes, the implemented software. Also included were less public artefacts produced by and for the developers, such as the usability evaluation reports and project databases containing requirements specifications, change requests, bug reports and so on. The third form of external artefacts available were personal, informal products of the development activities, such as private notes and sketches. The analyst had access to such artefacts produced by himself in his capacity as participant in the projects and access to some other such artefacts volunteered by other participants.

In addition, various supporting artefacts not produced by project activity but used by the participants were available for analysis. These included such artefacts as copies of recommended working procedures and practices, departmental telephone and room lists, forms from current paper and software based systems and user manuals.

The analyst also had access to his own internal models or understandings of the work and artefacts developed through the participant side of his participant-observer involvement in the projects. On the other hand, from the observer side of his participant-observer involvement, the analyst had access to more analytical and theoretical insights into the development work and products.

6.3.1 Tracing upstream

Given the range of available data, it was necessary to refine the scope of the analysis. Rather than attempting to devise and to use a codification of every feature of the implemented software systems (a mammoth and perilous undertaking in itself), the research used the results of the evaluations reported in sections 6.1 and 6.2 to define the scope for upstream tracing. Each usability issue identified in the evaluation was related to a particular feature of the software. 'Software feature', as used here, was defined by the users in the evaluation. If a user reported a usability issue with x then x was considered to be a software feature (with a related usability issue); it might be a button, a colour or a task structure or whatever was identified by the user as raising the usability issue.

Thus, we had a defined set of software features, 81 for the CI software and 29 for the CS software, for which we could attempt to trace the source. Moreover, with research question four in mind, we knew that each of the software features in the set had an explicit influence on the usability of the system. So, with this refined scope, the analysis traced upstream from the software features and associated usability issues identified in section 6.2.

Of the other forms of data available, the external artefacts provided a recorded history of contributions to the development work and of their subsequent progress, while the author's internal models and analytical insights supported the analysis and interpretation of that history. The structure of the software development method also supported the upstream and downstream tracing, by illuminating a course for the stream. Hence, in the development method, a task model of the current work situation (TM1) provided a basis for the derivation of a task model of the proposed work situation (TM2). TM2 in turn provided a basis for the production of a prototype. Finally, a prototype provided a basis for the implementation of the software system. Allowing for iteration and refinement both between and within these phases, the development structure provided a series of staging posts at which the progress of contributions could be checked.

Thus, upstream tracing began by addressing each feature of the evaluated software which had raised a usability issue. For each such feature, previous development artefacts (e.g. task models and prototypes) and records of development work (e.g. videos and notes) were examined, in reverse chronological order, to trace the provenance of the software feature. When tracing upstream from a particular software feature, the first place searched for its antecedent was in a prototype of the software in question. Similarly, having identified a contribution to a prototype, the

first place searched for its antecedent was in TM2. In the same way, the antecedent of a contribution to TM2 was sought in contributions to TM1.

6.3.2 Tracing downstream

The scope was also refined for downstream analysis, tracing from contributions to development discourse down to the software product. As noted above, if CG(O) constructed between user and developer during the development process is to provide the basis for the software design, we should expect contributions which have been assimilated into CG(O) to be reflected in the implemented software. On the other hand, we may disregard, for our present purpose in this analysis, contributions to CG(P), since the latter concerns the software development process rather than the software product.

Chapters four and five have analysed contributions to CG(O) through four distinct development activities: analysis of the users' current work situation, requirements analysis, envisioned work situation design and software design. Not all contributions may be expected to have equally direct relations with features of the implemented software. The most direct relationship is with software design proposals. It is relatively straightforward to check whether or not the implemented software embodies the content of such a contribution.

Contributions which provide design proposals for the users' work situation may also be reflected in the software, though perhaps less directly. If a user has to perform a task in the envisioned work situation, the software should support that task performance even if it is not directly performed with the software. The distinction between work and software design proposals is at times fuzzy, the latter (reflected primarily in prototypes) often being a refinement of the former (reflected primarily in TM2).

The content of many contributions to work situation design, while influencing the design and eventual usability of the system, may have no *direct* embodiment in the implemented software. Typical examples of this kind of contribution concern the layout of the working environment. These contributions may influence the design of the proposed new work situation and its software support but may have no direct and explicit representation in the software interface.

Contributions of requirements should also be reflected in the implemented software since the software is designed to meet such requirements. As with design proposals, requirements for the work situation may have a less direct embodiment in the implemented software than do requirements for the software *per se*.

Finally, contributions to analysis of the users' current work situation have at most only an indirect relationship with features of the new software system. Such contributions concern the situation *before* the new software is designed and introduced. Their content may be reflected in the new software only in so far as

their content remains unchanged between the current and envisioned work situations and, furthermore, is then reflected in the software design (prototype).

As a further refinement of the scope of contributions from which to trace downstream, this analysis examined only provisional contributions and not invitational contributions, since it is clear from the above discussion that it is the *provision* of requirements or design proposals which may produce particular software features.

The available data included a set of video records of cooperative development meetings. The video records included identifiable provisional contributions to CG(O) of requirements or design proposals. In a final refinement of the scope of the downstream tracing, analysis of the video records was restricted to the samples analysed in detail in chapter five. As described in chapter five, sample CS1 was taken from the video record of a meeting called to analyse the users' current work situation, sample CS5 was from a meeting called to design an envisioned work situation and CS7 was from a meeting called to design prototype software.

The analyst reviewed the sample video records, noting provisional contributions of requirements or design proposals. Records of subsequent development work and artefacts were then examined in chronological order to trace subsequent incorporations of the contribution, ultimately assessing if it became reflected in the evaluated software.

Tracing downstream was also supported by the structure of the development method. As a corollary of the upstream trace, if a contribution is identified in a particular external shared model, efforts to trace its subsequent history may usefully be directed towards the immediately downstream external shared model. Thus, the downstream tracing examined if a requirement or design contribution reflected in TM2 was subsequently reflected in a prototype and if one reflected in a prototype was subsequently reflected in a software implementation.

However, downstream tracing was made complicated by several factors. As discussed in chapter five, the appearance of a contribution in the video record of cooperative development meetings does not necessarily imply the effective assimilation of the contents of that contribution into an external shared model. A contribution may be ineffective or may be effective but assimilated into an external shared model at a later, perhaps unidentified, time. Determining a contribution's effectiveness is the first stage in tracing its downstream history (see chapter five).

A related complication, noted in chapters four and five, was that not all contributions in a cooperative development meeting were made on the declared theme of the meeting. So, for example, a contribution to software design might be made during a cooperative development meeting whose declared purpose was current work situation analysis. A further complication was the absence of an explicit external shared model of requirements. This led to requirements being

appended to other types of external shared model or requirements being transformed into corresponding design proposals which could be assimilated into the available external shared model, i.e. into TM2 or a prototype.

In each case, comparison was made between the contribution to the discourse and the stable version of the artefact which was used as a basis for moving on to a new phase of artefact construction. Hence, some contributions were made with reference to an earlier version of the artefact. Thus, the artefact TM2 used to trace the downstream representation of a contribution was the stable version from nine days after the sampled discourse in which the noted contributions were made. The version of TM2 being constructed on the whiteboard during the sampled interaction was considerably more primitive. Hence, subsequent meetings or later contributions during the same meeting may have involved contributions which developed or superseded those made in the sample. For example, the list of Reception Form query types established in the TM2 construction sample of interaction was further developed in a later meeting with another user and the later contributions were reflected in the paper prototype.

6.4 Results of tracing analysis

6.4.1 Results of upstream tracing

Upstream tracing from the usability evaluation results identified several sources of software features which raised usability issues. There were three sources of *features which appeared in the interface* and which raised usability issues. In addition, four sources were identified for the *absence of features* whose presence users felt should have enhanced usability. As an example of the latter, the absence of form customisation facilities in the CS software was reported as hampering usability. Upstream tracing identified one source of such absences as *unvalued* contributions, where a requirement or design proposal for the absent feature had been contributed during the development work but the feature was never implemented. Such a contribution had been effective (i.e. assimilated into the common ground of the development team; see chapter five) but had not been carried forward into the software. The corollary is that an effective contribution may be *valuable*, in the sense of being transformed into a corresponding feature of the implemented software. In the latter case, the contribution may be positively valued, having an enhancing effect on usability, or may be negatively valued, having a debilitating effect on usability.

A second source of absent features was that a corresponding contribution had been made to the development discourse, stating a requirement or proposing a design, but had been ineffective (i.e. not assimilated into the common ground of the development team; see chapter five). Thirdly, some software features were absent because they had never been requested or suggested during previous development work - or, at least, tracing did not turn up any evidence of such a request or suggestion. Fourthly, some allegedly absent features were in fact present in all or

in part. In the latter cases (five, all from the CS software evaluation), the users had not received sufficient training to be aware of the features and the design and/or implementation of the feature was not sufficiently intuitive without such training. Thus, a software feature which was present and only appeared to the user to be absent had its source in one of standard flow, deus ex machina or technology contributions (see below).

One source of features which were *present* in the software and which raised usability issues may be termed *standard flow* through the development activities. A participant makes a contribution to the development discourse which is effectively assimilated into an external shared model and subsequently undergoes one or more transformations (depending upon the external shared model into which it was initially assimilated) to become a feature of the implemented software. Present software features developed through standard flow raised 20.7% of the usability issues in the CS evaluation and 7.4% in the CI evaluation.

The second source of software features which raised usability issues was the implementation technology. Each project bought and used a commercially available software development environment (SDE) with which to implement the software. The SDE was different in each project and in each case was the leader in its sector of the SDE market. In addition, the CI implementors were extending the limits of state of the art document parsing technology in their product. These technological factors forced the introduction of features which raised usability issues, 12.4% of all those raised by the CI evaluation and 17.2% for the CS evaluation.

The third source of such features was deus ex machina: features which had not appeared in any of the cooperative development activities or in any previous external artefacts and which then suddenly appeared in the implemented software. These features accounted for 20.7% of the usability issues raised by the CS software and 74.1% for the CI software. The reasons for and implications of this phenomenon are discussed in section 6.5 below.

Figures 6.3 and 6.4 summarise the sources of software features or the absence thereof which raised usability issues for the CS and CI software respectively. The tables list usability issues raised by software features present in the interface through standard flow contributions, implementation technology and deus ex machina and usability issues raised by software features absent from the interface through new contributions, ineffective contributions, unvalued contributions and inadequate training. Sources which are not listed at a given level were not found to have raised issues at that level. For example, no *Absent: training* or *Absent: ineffective* sources were identified for CI usability issues at any level.

Deus ex machina features accounted for the vast majority (74.1%) of usability issues raised by the CI evaluation. In contrast, while deus ex machina features were also the joint most common source in the CS evaluation, they were only slightly ahead of two other sources, at 20.7% compared to 17.2%.

Figure 6.3: Sources of CS software usability issues at each level of severity

	Issues	% of level	% of total
Level 1			
Standard flow	1	50	3.5
Technology	1	50	3.5
Level 2			
Deus ex machina	1	25	3.5
Standard flow	1	25	3.5
Absent: unrequested	1	25	3.5
Absent: training	1	25	3.5
Level 3			
Standard flow	2	50	6.9
Technology	1	25	3.5
Absent: training	1	25	3.5
Level 4			
Deus ex machina	4	50	13.8
Absent: unrequested	1	12.5	3.5
Absent: training	1	12.5	3.5
Absent: unvalued	2	25	6.9
Level 5			
Deus ex machina	1	9.1	3.5
Standard flow	2	18.2	6.9
Technology	3	27.3	10.4
Absent: unrequested	1	9.1	6.9
Absent: training	2	18.2	6.9
Absent: ineffective	2	18.2	6.9
All levels			
Deus ex machina	6		20.7
Standard flow	6		20.7
Technology	5		17.2
Absent: unrequested	3		10.4
Absent: training	5		17.2
Absent: ineffective	2		6.9
Absent: unvalued	2		6.9

Sources of usability issues were more evenly spread in the CS project, with standard flow also accounting for 20.7%, implementation technology and features thought to be absent through lack of training or poor design each accounting for 17.2% of the total. Absent features which had not previously been requested accounted for 10.4%. Absent features which had previously been effectively

assimilated into CG(O) accounted for only 6.9% and absent features for which an earlier contribution had been ineffective also accounted for 6.9%.

Figure 6.4: Sources of CI software usability issues at each level of severity

	Issues	% of level	% of total
Level 1			
Deus ex machina	22	75.9	27.2
Standard flow	3	10.4	3.7
Technology	3	10.4	3.7
Absent: unrequested	1	3.5	1.2
Level 2			
Deus ex machina	9	75	11.1
Technology	1	8.3	1.2
Absent: unrequested	1	8.3	1.2
Absent: unvalued	1	8.3	1.2
Level 3			
Deus ex machina	9	64.3	11.1
Technology	3	21.4	3.7
Absent: unrequested	2	14.3	2.5
Level 4			
Deus ex machina	7	63.6	8.6
Standard flow	1	9.1	1.2
Technology	3	27.3	3.7
Level 5			
Deus ex machina	13	86.7	16.1
Standard flow	2	13.3	2.5
All levels			
Deus ex machina	60		74.1
Standard flow	6		7.4
Technology	10		12.4
Absent: unrequested	4		4.9
Absent: unvalued	1		1.2

There were fewer identified sources of usability issues in the CI project, with neither features thought to be absent through lack of training or absent features for which an earlier contribution had been ineffective appearing at all. Aside from deus ex machina features' domination, there was a gradual diminution in frequency across implementation technology at 12.4%, standard flow at 7.4%, absent features which had not previously been requested at 4.9% and absent features which had previously been assimilated into CG(O) at 1.2%.

6.4.2 Results of downstream tracing

As discussed in section 6.3.2, the starting point for a downstream trace was a provisional contribution to CG(O) of a requirement or a design proposal. The analysis of contributions in chapters four and five has shown that contributions do not usually appear in isolation on single subjects. Rather, we find patterns of interaction sequences which may involve several contributions with the same or

similar content. So, in the present analysis, we find clusters of contributions which reiterate or elaborate a particular requirement or design proposal. Thus, in tracing from contributions through to features of the software product, we frequently find several discrete contributions which map to a single software feature or usability issue.

It was also generally, and unsurprisingly, the case that effective contributions to a particular development activity were assimilated into the external shared models which supported that activity and its downstream activities, but not into those which supported its upstream activities. Thus, effective contributions of work requirements or work design proposals were typically assimilated into TM2 (the task model of the envisioned work situation) and often translated into the paper prototypes and running software. They were generally not, however, assimilated into TM1 which continued to reflect current working practices rather than requirements or designs for envisioned work.

Similarly, effective contributions of software requirements or designs were typically assimilated into paper prototypes and often translated into the running software. They were generally not, however, assimilated into TM1. They were occasionally assimilated into TM2 where the software requirement or design substantially impinged on the envisioned work practice. As noted in section 6.3.2, the line between software design proposals and work design proposals was sometimes not a clear one.

Sections 6.4.2.1 to 6.4.2.3 present the results of tracing downstream from contributions made in samples CS1, CS5 and CS7 respectively.

6.4.2.1 Results of tracing downstream from work analysis activity

In sample CS1 analysed in chapter five (see Figure 5.6), there was a total of forty-two provisional user contributions to CG(O). Thirty-nine of these were contributions to analysing the users' current work situation, the declared purpose of the meeting. The other three of these contributions provided requirements. There were no provisional contributions of design proposals to CG(O) by users. There was also one contribution to CG(O) by a developer providing a requirement and one contribution to CG(O) by a developer providing a work design proposal.

So, from the scope identified in section 6.3.2, there were five identified contributions from which to trace downstream, three from a user and two from a developer. These five contributions followed quite closely in the discourse, appearing as part of the interaction patterns described in chapter four. The meeting from which this sample was taken was called to analyse the users' current work situation and much of the early part of the sample consisted of contributions to work situation analysis, mainly invitational contributions from a developer and provisional contributions from a user. The latter contributions to current work situation analysis accounted for the vast majority of the forty-two provisional user

contributions to CG(O). The latter part of the sample then included an interaction sequence involving the five contributions to requirements analysis and design described here. (See Appendix z for a transcript of sample CS1.) Contributions in this sample showed the characteristic clustering into interaction sequences around a single topic or set of related topics. Thus, all the contributions identified in this analysis as provisional contributions to CG(O) of requirements or design proposals came in three contiguous interaction sequences in the final two minutes of the sample.

In UC1³, a user contributed the requirement that Liaison and Reception are logically distinct roles but may be performed by any person working in the reception area. In providing UC2, he noted that the work of Liaison was relocated to the Reception area because the Liaison staff must be physically accessible to customers. In DC1, a developer made the work design proposal of creating a single work role encompassing the tasks of Liaison, Reception and Fault Line. This entailed the requirement, contributed by the developer in DC2, that one person fulfilling such a role needs to be able to access software to support each of the three tasks. In response to this, a user provided, in UC3, the requirement that a user should wish to do only one of these tasks at any time and so should want software support for the tasks to be accessible as separate screens, not within one screen.

UC1 (Work requirement): *Liaison and Reception are logically distinct roles but may be performed by any person working in the physical reception area.*

Looking at the external models, in *TM1* Liaison, Reception and Fault Line were modelled as separate roles. Help Desk, Technical Consultant and Fault and Network Services were also modelled as distinct roles in *TM1*.

In *TM2*, there was one role labelled 'Reception' which subsumed Reception and Fault Line tasks. These were the tasks performed in the reception area before the arrival of Liaison. There was one role in *TM2* labelled 'Technical Queries' which subsumed the Technical Consultant and Liaison tasks and also the Help Desk tasks. Liaison and Help Desk were viewed as special cases of Technical Consultant. The person fulfilling the Reception role filled in a Query Form only for Fault Reports. Otherwise, she referred the *customer* (rather than the query) to another role. The Reception role logged all queries, including Fault Reports, on the Reception Form.

In the *paper prototypes*, there was one form for Reception and a separate form for Fault Line and Technical Queries (i.e. including the erstwhile Liaison and Technical Consultant tasks). There was a button on the Reception Form which allowed the user to bring up a Query Form.

In the new designed *software*, the Query form was implemented. However, the Reception form was not implemented and remained as the existing paper form.

³ User contributions are referenced by UC and a number, developer contributions by DC and a number.

In the new *designed work situation*, Reception staff performed Reception, Liaison and Fault Line roles with Liaison sometimes seen as separate from the other two roles.

Note that in TM2 filling in the Query form came before filling in the Reception form. In the paper prototype, this was (implicitly) reversed by the 'Main Query Form' button on the Reception form.

UC2 (Work requirement): *Liaison must be physically accessible to customers.*

In TM1, Liaison staff were modelled as directly accessible to customers.

In TM2, the Technical Queries role was directly accessible to customers.

In the paper prototypes, the customers' access to the user was not directly reflected in the interface except in so far as one option for method of contact on the Query form was 'in person'.

The new designed software followed the paper prototype in having an 'in person' option for method of contact on the Query form.

In the new designed work situation, customers have direct access to Liaison staff, being able to walk up to their desk at reception or to phone them directly or to be passed to them by other Reception or Help Desk staff.

UC3 (Software requirement): *One person would fulfil only one of the Liaison, Reception and Fault Line roles at any time, so wants software support for their tasks accessible as separate 'screens' and doesn't need simultaneous access to them.*

In TM1, this requirement was not explicitly modelled.

In TM2, UC3 was not modelled (but see UC1).

In the paper prototypes, as noted for UC1, there was one form for Reception and a separate form for Fault Line and Technical Queries (i.e. including the erstwhile Liaison and Technical Consultant tasks). There was a button on the Reception Form which allowed the user to bring up a Query Form.

In the new software, the Query form was accessible to the user (as was the paper Reception form).

In the new work situation, the Paper Reception form was generally not used. The Software Query form was used by many staff.

DC1 (Work design proposal): *Create a work role encompassing the tasks of Liaison, Reception and Fault Line or a 'super-role' such that a person filling it may fill any of the roles of Liaison, Reception and Fault Line.*

In TM1, Liaison, Reception and Fault Line roles were modelled separately.

For representation of these roles in TM2, see UC1.

In the paper prototypes, this proposal was not modelled directly. It was supported by a button on the Reception form to invoke a Query form.

In the new designed software, DC1 was not explicitly modelled. A software Reception form was not implemented.

In the new work situation, see UC1.

DC2 (Software requirement): *One person fulfilling such a role needs to be able to access software to support each of the three tasks.*

In TM1 and TM2, DC2 was not modelled.

For DC2 in the paper prototypes and in the new designed software, see UC1.

In the new work situation, see UC3.

The downstream trace from provisional contributions to CG(O) within the CS1 sample through subsequent development artefacts is summarised in Figure 6.5.

In the CS1 sample, UC1 and UC2 were not *directly* related to software usability since as work requirements they were not directly embodied in the software interface. However, a feature of UC2 was reflected in a usability issue. Liaison staff must be physically accessible to customers in order to interact with the customer, eliciting what their enquiry is and attempting to answer it. A recurring usability issue was that users of the system found it difficult to log details of a query on the system while interacting with the customer. Taking a query requires a lot of concentration because the user has to rephrase the information the customer provides and think about the problem at the same time as continuing to listen to the customer. The system took too much of the user's attention away from the customer. Thus, in practice, the user tended to deal with the customer first, perhaps noting details with pen and paper, and entering the details on the system later. One repercussion of this was that, especially in the case of Help Desk staff who frequently had no breaks between customers, the enquiry might be logged much later - if ever. Another, substantive, issue caused by this was the inability to provide the customer with a reference number. It was intended that the system should generate a unique reference number for each logged enquiry. The customer could then quote this reference number when calling back to check on progress, allowing the logged details to be retrieved quickly. Delaying entering the details meant that the customer could not be provided with a reference number.

Figure 6.5: Embodiment of CS1 contributions in subsequent artefacts

CS1 Work analysis (building TM1)	Task Model 1	Task Model 2	Paper prototype	New software
UC1 (Work requirement)	Liaison and Reception modelled separately	Reception and Fault Line subsumed in reception role Liaison and Technical Consultant subsumed in Technical Queries role	One form for Reception One form for Fault Line and Technical Queries (i.e. including erstwhile Liaison and Technical Consultant)	Query form implemented Reception form remains as old paper form Reception staff perform Reception, Liaison and Fault Line roles with Liaison sometimes seen as separate from other two roles
UC2 (Work requirement)	Liaison accessible to customer	Technical Queries role accessible to customer	Not modelled	Not modelled Customers have direct access to Liaison staff
UC3 (Software requirement)	Not modelled	Not modelled (but see UC1)	See UC1	Software Query form and paper Reception form both accessible to user Paper Reception form generally not used Software Query form is used
DC1 (Work design proposal)	Liaison, Reception and Fault Line modelled separately	See UC1	Not modelled directly Supported by button on Reception form to invoke Query form	Not modelled Software Reception form not implemented See UC1
DC2 (Software requirement)	Not modelled	Not modelled	See UC1	See UC1 and UC3

By TM2, the work design had been elaborated from UC1, DC1 (and UC3) to model one role covering Reception and Fault Line tasks and one role covering Liaison and Technical Consultant tasks. The Reception/Fault Line distinction essentially meant only the type of customer enquiry which was entered and forwarded on the system. Software support was implemented for recording, retrieving and forwarding customer enquiries in both these roles and the Faults (Technical Services) role. Software support was not implemented for the other Reception primary task of recording that an enquiry had been made. This raised a usability issue in so far as the users had to continue using the paper based forms for this task and could not automatically transfer details between the paper and software forms. This issue relates also to DC2 and UC3. Clearly, the software requirement stated in DC2 was not met. The lack of dependency between reception tasks, articulated in UC3, allowed the implementation of one form in software while the

other remained paper based without loss of usability within either form. However, the paper prototypes of the Reception form had included a button which opened a main query form, transferring entered details from the Reception form to the latter. The failure to implement this facility made the combined tasks of filling in both forms less efficient.

6.4.2.2 Results of tracing downstream from work design activity

In sample CS5, there was a total of thirteen provisional user contributions to CG(O) (see Figure 5.7). Again, three of these contributions provided requirements. There was one provisional contribution of a software design proposal to CG(O) by a user. There were also four contributions to CG(O) by a developer providing requirements, five contributions to CG(O) by a developer providing work design proposals and five contributions to CG(O) by a developer providing software design proposals.

So, we had eighteen identified contributions from which to trace downstream, four from a user and fourteen from a developer. Again, these contributions appeared clustered within interaction sequences. For example, DC7, DC11, DC12, DC13, DC14, UC2 and UC4 all contributed to the content and appearance of the Reception form being designed. Related to this, contributions DC1, DC2, DC3, DC4 and DC6 contributed to an understanding of what needed to be recorded by Reception staff on such a form. In contributions DC8, DC9 and DC10, details of recording and referring a query are added to the task model of the proposed work situation. Contributions, such as UC1, UC3 and DC5, which in the list below appear not to cluster with other contributions, were also set within interaction sequences in the discourse. Their apparent isolation is simply because other contributions in their respective interaction sequences did not fall within the scope of the present analysis, i.e. provisional contributions to CG(O) of requirements or design proposals.

UC1 (Software requirement): *The Query Form should not prompt the user through a series of questions to ask a customer making a query.*

In TM2, talking to the customer was modelled simply as 'Elucidate problem details' with no decomposition.

In the paper prototypes, there were no temporal dependencies within the Query form.

In the designed new software, there were no temporal dependencies. There were compulsory fields, Answerer, Time expended and Status but these did not record information from the customer.

UC2 (Work requirement): *Typing in a name on the Reception Form is too time consuming.*

In TM2, the Record Query task for the Reception role included the subtask 'Record Customer Name' (cf. DC4 below).

In the paper prototypes, there was a type-in field on the Reception form for customer name (cf. DC4 below).

In the designed new software, the Reception form was not implemented.

UC3 (Work requirement): *Reception users do not need to know other staff's specialities because they can just refer customers to Help Desk or Liaison staff who are expected to know.*

In TM2, Reception staff referred a customer to the Help Desk or Liaison if they could not identify the type of query or they could not identify an appropriate referee.

In the paper prototypes, there was no software support for identifying a referee.

In the designed new software, there was no software support for identifying a referee (but see DC9 below).

UC4 (Software design proposal): *The customer's department should be recorded on the Reception Form.*

In TM2, the Record Query task for the Reception role included the subtask 'Record Customer Department'.

In the paper prototypes, there was on the Reception form a Customer Department field with a dropdown list of departments.

In the designed new software, the Reception form was not implemented in software.

DC1 (Software requirement): *At times the user wants to fill in only some of the fields in a Reception Form, so filling all of them should not be enforced.*

In TM2, this was not modelled.

In the paper prototypes, there were no compulsory fields on the Reception form.

In the designed new software, the Reception form was not implemented.

DC2 (Work requirement): *A user needs to record the nature of a query on the Reception Form.*

In TM2, the Record Query task for the Reception role included the subtask 'Record Query Type'.

In the paper prototypes, there was a Query Type field with a dropdown list of query types on the Reception form.

In the designed new software, the Reception form was not implemented.

DC3 (Work requirement): *A Reception user wants to be able to charge the appropriate department for time spent on a query.*

In TM2, the Record Query task for the Reception role included the subtask 'Record Customer Department'.

In the paper prototypes, there was a Customer Department field with a dropdown list of departments on the Reception form.

In the designed new software, the Reception form was not implemented.

DC4 (Software requirement): *The software should allow a Reception user to record the customer's name and department.*

In TM2, the Record Query task for the Reception role included the subtask 'Record Customer Department'. Also compare UC2 above.

In the paper prototypes, there was a type-in field on the Reception form for customer name and a field with a dropdown list of departments.

In the designed new software, the Reception form was not implemented.

DC5 (Work design proposal): *The first two identified key tasks for the user are 'just basically talking to the customer' to receive and identify a query, without recourse to software support.*

In TM2, 'Receive query' is modelled simply, without decomposition. 'Identify query' is specified to identifying the query as a technical query or a fault report. The consequent tasks for each of these and 'Don't know' are decomposed.

In the paper prototypes, this was not explicitly modelled

In the designed new software, this was not directly supported.

DC6 (Work design proposal): *In recording a query, a person in the Reception role records: that it was a general enquiry, a sales enquiry, which computer platform, nationally referred, a software enquiry, a timetabling or documentation enquiry or a password enquiry.*

In TM2, 'Record Query Type' was not decomposed. (The paper form used in the current work situation was marked up in a later meeting to reflect this proposal.)

In the paper prototypes, a dropdown list of query types on the Reception form included: General Enquiries, Sales (consumables), Student Facilities [with submenu], Software, Library/Docs, Courses, Timetabling, National Services [with submenu].

In the designed new software, the Reception form was not implemented. (There was a different dropdown list of query types on the Query form.)

DC7 (Software design proposal): *The system will provide a Reception user with a simple means of recording the query type on the Reception Form.*

In TM2, this was not modelled. It was not elaborated in the development discourse either.

In the paper prototypes, see DC6 above.

In the designed new software, the Reception form was not implemented.

DC8 (Work design proposal): *D1 adds to TM2 'record department' and 'record query type' as subtasks of recording a query at Reception.*

In TM2, this was assimilated through a direct representation contribution.

In the paper prototypes, see UC4 and DC3 for 'record department'. See DC3 and DC6 for 'record query type'.

In the designed new software, the Reception form was not implemented.

DC9 (Work design proposal): *D1 draws line on TM2 to represent that 'Identify referee' is a subtask of the user's task 'Refer query'.*

In TM2, this was assimilated through a direct representation contribution.

In the paper prototypes, clicking the 'Refer problem' button (note change from 'refer query' in TM2) invoked a dialogue box which included a 'Name of referee' field. (This in turn implies identifying the referee in order to fill in the field.) See Figure 6.7.

In the designed new software, there was a dropdown list of email addresses for the referee field. The design was changed from the dialogue box invocation in the paper prototype. If 'Submit' is selected with a referee's email address in the field, then the query is referred; if the referee field is blank then the query is just saved to the database.

DC10 (Work design proposal): *D1 adds 'Refer query' subtask box to TM2 as subtask of 'Refer query' and following subtask 'Identify referee'.*

In TM2, this was again a direct representation contribution. Note that 'Refer query' was used twice, both for the task and one of its subtasks.

In the paper prototypes, the dialogue box invoked by the 'Refer query' button had a 'Send' button (renamed from 'Refer' to avoid confusion with the invoking button - cf. TM2).

In the designed new software, this was supported by the software design described in DC9 above.

DC11 (Software design proposal): *The software implemented Reception Form should include checkboxes for the following query types: general enquiry, sales enquiry, computer platform, national service.*

In TM2, this was not modelled.

In the paper prototypes, a dropdown list of query types on the Reception form included: General Enquiries, Sales (consumables), Student Facilities [with submenu], Software, Library/Docs, Courses, Timetabling, National Services [with submenu]. Note that this was not precisely what DC11 proposed.

In the designed new software, the Reception form was not implemented.

DC12 (Software design proposal): *The software implemented Reception Form should include a checkbox for a software enquiry.*

In TM2, this was not modelled.

In the paper prototypes, the drop down list of query types included Software. (See DC11.)

In the designed new software, the Reception form was not implemented.

DC13 (Software design proposal): *The software implemented Reception Form should include a checkbox for a timetabling or documentation enquiry.*

In TM2, this was not modelled.

In the paper prototypes, a dropdown list of query types on the Reception form included: General Enquiries, Sales (consumables), Student Facilities [with submenu], Software, Library/Docs, Courses, Timetabling, National Services [with submenu]. Again, note that this is slightly different from the DC13 proposal.

In the designed new software, the Reception form was not implemented.

DC14 (Software design proposal): *The software implemented Reception Form should include a checkbox for a password enquiry.*

In TM2, this was not modelled.

In the paper prototypes, there was no such checkbox. Query types were chosen from a dropdown list (see DC13). Password enquiry was not on the list.

In the designed new software, the Reception form was not implemented.

The downstream trace from provisional contributions to CG(O) within the CS5 sample through subsequent development artefacts is summarised in Figure 6.6.

Figure 6.6: Embodiment of CS5 contributions in subsequent artefacts

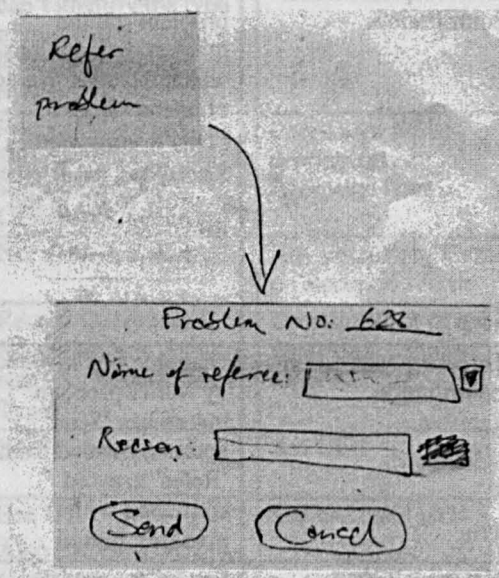
CS5 Work design (building TM2)	Task Model 2	Paper prototype	New software
UC1 (Software requirement)	Talking to customer modelled simply as 'Elucidate problem details' with no decomposition	No temporal dependencies within Query form	No temporal dependencies, but compulsory fields: Answerer, Time expended and Status
UC2 (Work requirement)	Record Query task for Reception role includes subtask 'Record Customer Name' <i>[cf. DC4]</i>	Type-in field on Reception form for customer name <i>[cf. DC4]</i>	Reception form not implemented in software
UC3 (Work requirement)	Reception staff refer a customer to the Help Desk or Liaison if they can't identify the type of query or they can't identify an appropriate referee	No software support for identifying referee	No software support for identifying referee (But see DC9)
UC4 (Software design proposal)	Record Query task for Reception role includes subtask 'Record Customer Department'	Dropdown list of departments on Reception form	See UC2
DC1 (Software requirement)	Not modelled	No compulsory fields on Reception form	See UC2
DC2 (Work requirement)	Record Query task for Reception role includes subtask 'Record Query Type'	Dropdown list of query types on Reception form	See UC2
DC3 (Work requirement)	See UC4	See UC4	See UC2
DC4 (Software requirement)	See UC4 Also compare UC2 <i>(conflict)</i>	See UC2 and UC4	See UC2

DC5 (Work design proposal)	'Receive query' is modelled simply, without decomposition 'Identify query' is specified to identifying the query as a technical query or a fault report The consequent tasks for each of these and 'Don't know' are decomposed	Not modelled	Not supported in software
DC6 (Work design proposal)	'Record Query Type' was not decomposed (<i>The paper form was marked up in a later meeting</i>)	Dropdown list of query types on Reception form includes: General Enquiries, Sales (consumables), Student Facilities [with submenu], Software, Library/Docs, Courses, Timetabling, National Services [with submenu]	See UC2 <i>Different dropdown list of query types on Query form</i>
DC7 (Software design proposal)	Not modelled Not elaborated in discourse either	See DC6	See UC2
DC8 (Work design proposal)	Direct representation contribution	See UC4 and DC3 for department See DC3 and DC6 for query type	See UC2
DC9 (Work design proposal)	Direct representation contribution	Clicking 'Refer problem' button invokes dialogue box which includes 'Name of referee' field (This in turn implies identifying the referee in order to fill in the field)	Dropdown list of email addresses for referees Design changed from dialogue box invocation If 'Submit' selected with referee's email address in field, then query is referred; if referee field is blank then query is just saved to database
DC10 (Work design proposal)	Direct representation contribution	Dialogue box invoked by 'Refer problem' button has 'Send' button (renamed from 'Refer' to avoid confusion with invoking button)	Supported by software design described in DC9
DC11 (Software design proposal)	Not modelled	See DC6 - note differences	See UC2
DC12 (Software design proposal)	Not modelled	See DC6 - menu item is there	See UC2
DC13 (Software design proposal)	Not modelled	See DC6 - note differences	See UC2
DC14 (Software design proposal)	Not modelled	See DC6 - menu item is not there	See UC2

As with contributions in the previous sample, the main usability issue arising from contributions which stated requirements or made design proposals for the Reception form (DC7, DC8, DC11, DC12, DC13, DC14, UC2 and UC4) was that development of this form reached the paper prototyping stage but was not then implemented in software. Thus, none of these contributions was realised. The related requirements stated in DC1, DC2, DC3 and DC4 could all be met through continued use of the existing paper form. However, the software form design proposed in DC11, DC12, DC12 and DC14 and verified in DC6 was different in content and structure from the existing paper form because the users were not satisfied with the existing form. So, while the existing paper form continued to be used, the users continued to be dissatisfied with its usability.

DC9 and DC10 represented in TM2 that subtasks of referring a query were first to identify an appropriate referee and then to refer the query to that referee. In the paper prototypes this was supported by a button labelled 'Refer problem' on the main query form. Selecting this button invoked a dialogue box with the query reference number displayed; fields labelled 'Name of referee' and 'Reason'; and 'Send' and 'Cancel' buttons (see Figure 6.7). Send was renamed from Refer (in TM2) to avoid confusion with the button invoking the dialogue box. 'Name of referee' was a compulsory field and filling it in implied having identified an appropriate referee.

Figure 6.7: Invoking a 'refer problem' dialogue in the CS paper prototype



This design was refined further and in the implemented software there were fields labelled 'Referee's Email Address' and 'Reason for referral' (see bottom of Figure 6.8). The 'Referee's Email Address' field had a dropdown menu of email addresses of all potential referees from which one could be selected. Queries were registered on the system by clicking the submit button (see top left of Figure 6.8 just below the 'Apple menu' icon). If the query was submitted without a specified

referee, the query was simply stored in the system database. If a referee had been specified, the query was stored and a message was sent automatically to the referee informing her that a query had been referred to her and providing the reference number.

Figure 6.8: Submitting a query in the CS software

In general, the software support for referring queries was well used and well liked. However, there were a few usability issues with it. Most of these appeared to result from a lack of even basic training for new users. One user expressed a preference for a 'Forward' button to refer a query, in addition to the 'Submit' button for storing the query. (This is similar to the paper prototype design and runs into the problem that the system cannot make a useful referral until after the query has been submitted.) This user was unsure of the distinction between submitting with and without a specified referee, as described above. Several users referred queries by normal email rather than through the system. They claimed that they did not know it was possible to refer queries on the system. One user claimed to leave unsolved queries open in the vague hope that someone would find it! These users had usability problems with the system which resulted from an almost complete absence of introduction to and training on the system. One user wanted to refer queries based on problem type instead of to a specific person. Such an approach could ameliorate the difficulty of identifying an appropriate referee. However, one of the problems with the current system which the new system was intended to resolve

was queries remaining unanswered because no individual took responsibility for them.

UC1 suggested that users should not be forced to fill in particular fields in a particular order on the main query form and in TM2 the corresponding subtask was decomposed no further than simply 'Log details'. In the implemented software, three fields were made compulsory on the query form: Answerer, Time expended and Status. The labels of these fields were in boldface but users were universally unaware that this indicated compulsory fields. Status was set to Open by default and could be set to Open, Solved or Insoluble. It could not have all three turned off. It was possible to select the Submit button with one of the other two compulsory fields unfilled. In this case, the query was not submitted and a message box appeared listing the compulsory fields which had not been completed. Clicking 'OK' in this message box returned the user to the submit form. Users lack of awareness of the compulsory fields suggests they had not tried to submit queries without filling in these fields.

DC5 suggested that software support was not required for the task of identifying the type of a problem or enquiry. In the implemented software, the 'Problem/Enquiry Type' field had a dropdown list of types from which one could be selected. This served to save typing time and provide some guidance and consistency to what was logged as a type. This facility was well received by all users. In addition, the ease with which users could refer queries around to colleagues in the new system provided a support for recovery from sending the query to the wrong person. This was especially useful to Reception staff who had to determine from what a customer said whether to refer the query to a technical consultant or to the fault repair staff. Misdirection of queries in this situation was a substantial problem in the existing system. The new design accepted that queries might be misdirected and built in support for recovery. All users liked this facility. However, it is worth noting that the fault repair staff often did not refer a misdirected query on to an appropriate technical consultant, as intended in the design, but instead referred it back to the Reception user who had originally sent it, requiring the latter to redirect it. In this way, the fault repair staff used the ease of referral to draw the Reception user's attention to the initial misdirection.

6.4.2.3 Results of tracing downstream from software design activity

In sample CS7, there was a total of twenty-one provisional user contributions to CG(O) (see Figure 5.8). Nine of these contributions provided requirements. There were no provisional contributions of work design proposals to CG(O) by users. There were six provisional contributions of software design proposals to CG(O) by users. There were also three contributions to CG(O) by a developer providing requirements, one contribution to CG(O) by a developer providing a work design proposal and eleven contributions to CG(O) by a developer providing software design proposals.

So, we had thirty identified contributions from which to trace downstream, fifteen from a user and fifteen from a developer. Once again, these contributions appeared clustered within interaction sequences on particular topics. For example, UC1 and UC2 formed part of a sequence in which a developer sought a user's preference for the order of presentation of 'customer details' fields. UC3 and UC4 then appear in an immediately succeeding sequence verifying the information provided in the UC1 and UC2 sequence. There followed a short but interesting sequence in which the user had an insight into the work requirements and the software support needed to meet them (see transcript CS71600 below). At the end of the preceding interaction sequence, the developer D had asked for verification that 'contact' was the most important field of customer details. The user U provided this verification with 'Yes' and then, in UC5, tentatively suggested a relationship between customer name and contact details. In UC10, the user went on to elaborate the content of the customer contact field.

In UC7, the user noted that a single contact field may not be an appropriate design and that a phone number is a necessary adjunct to a customer's name. Throughout this sequence, the developer repeatedly tried to interrupt the user's contributions on the content of the contact field and to return to verifying the order of customer details fields. He finally achieved this with his verification of a requirement in DC1, moving the discourse on with his questions at the end of the sample transcript below. The analysis of effectiveness in chapter five suggested that contributions UC5, UC7 and UC10 were ignored by the developer and never assimilated into CG(O). At this time, the developer was simply interested in establishing an ordered list of fields and missed the point that the nature and use of the fields had not adequately been thought through. The present analysis suggested that this had direct consequences for the usability of the developed software. This point is taken up below.

CS71600⁴

D: Did you say contact would be first?

U: Yes. I'm just thinking actually sort of name as far as we're concerned kind of forms part of contact

D: Right.

U: details. **[UC5]**

D: So. Right.

U: Umm. I mean I guess what we're talking about in the little box that we're calling contact is things like phone number and email

D: Yeah.

U: and so on. **[UC10]**

So is there necessarily even a single box? Um. I mean it varies in fact because if you've got the email you don't need the personal name as well. But if you've got a phone number you do need the personal name as well. **[UC7]**

D: Yeah. Uhh. So, so contact there is important both in terms of what they give you and what you need. **[DC1]**

What about the other three? Is there any greater importance or order to them?

⁴ Underlined transcript indicates an individual contribution referenced by the contribution number (UC or DC) following the underlining.

UC6 appeared towards the end of the next interaction sequence, which was mainly taken up with a user providing information on the current work situation and work requirements in response to invitations from a developer. Then UC15, UC8, DC4 and UC11 were clustered in an interaction sequence which verified the ordering of customer details fields. In this sequence, the assumption that a user will be entering the information from the customer as the customer provides it remains unquestioned. Thus, a developer says, 'presumably it's also to your convenience if when they phone you up if they tell you in an order which is reasonably close [to the order of fields in the form]'. Failure to examine this assumption also led to usability issues.

Contributions DC5, DC6, DC7, DC8, DC9, UC12, DC10, DC2, UC9, DC11, UC13, DC12, UC14, DC13 and DC14 then followed in an interaction sequence devoted to designing dropdown lists of options for the 'customer type' and 'department' fields of customer details. In DC5 and 6, D2 proposed a dropdown list for the customer type field and in UC14, the user reminded the developers that there should be four items on this list. DC7, UC12, DC12, DC13 and DC14 propose having a dropdown list of departments grouped by faculty for the department field. DC8, DC9, UC12, DC2 and UC9 propose having a dropdown list of forms of contact (email, phone, postal address and fax) for the contact field. In DC10 and DC11, D1 notes that subfields of contact 'would need to be fields which you filled in. Clearly it's the content of the field you're interested in not just the label'.

Finally, DC15 and DC3 appeared in the last interaction sequence of the sample which verified the software interface design established in the preceding sequences.

UC1 (Work requirement): *Customer details could be entered in an order which is most convenient for the system user or in the order in which the customer provides them.*

In the paper prototypes, the fields were arranged approximately from left to right and top to bottom in the order: Customer name, Contact, Login, Dept, Customer type.

In the designed new software, the fields were in the order: Customer Name, Login Id, Department, Customer Type, Method of Contact, Previous contact.

UC2 (Work requirement): *The most useful order for the user is customer contact details, customer login, department, customer type, then name.*

In the paper prototypes, see UC1.

In the designed new software, see UC1.

UC3 (Work requirement): *Customer name is the last element of customer details that the user wants to enter (verifying UC2).*

In the paper prototypes, see UC1.

In the designed new software, see UC1.

UC4 (Work requirement): *Contact details is the first element of customer details that the user wants to enter (verifying UC2).*

In the paper prototypes, see UC1.

In the designed new software, see UC1.

UC5 (Work requirement): *The customer's name actually forms part of the contact details.*

In the paper prototypes, see UC1. Analysis of effectiveness suggested that this contribution was ignored.

In the designed new software, see UC1.

UC6 (Work requirement): *Usually customer's department is more important to the user than customer type.*

In the paper prototypes, see UC1.

In the designed new software, see UC1.

UC7 (Work requirement): *The user needs to know a customer's name if he has a phone number but not if he has an email address. (Cf. UC5.)*

In the paper prototypes, see UC1. Analysis of effectiveness suggested that this contribution was ignored.

In the designed new software there were no dependencies between the fields.

UC8 (Software requirement): *It should be convenient for the order of fields in the form to match the order in which the customer provides the information.*

In the paper prototypes, see UC1.

In the designed new software, see UC1.

UC9 (Software requirement): *The required subfields of contact are email, phone, postal address and fax.*

In the paper prototypes, the subfields of Contact were: e-mail, phone, post, fax.

In the designed new software, the subfields of Method of Contact were: In person, Telephone, Electronic mail, Post, Facsimile, No re-contact.

UC10 (Software design proposal): *Contact field should contain the customer's phone number and email.*

In the paper prototypes, see UC9 and DC9. Analysis of effectiveness suggested that this contribution was ignored.

In the designed new software, either the phone number or the email may be selected from a dropdown list, but selecting a second overwrites the current contents of the field. Anything may be entered in the field manually but the field was limited to fifteen characters maximum.

UC11 (Software design proposal): *Fields should be in order of importance from left to right and from top to bottom.*

The interface was not designed in the simple grid-like way implied by this contribution. However, in the paper prototypes, the sections of the interface were in top to bottom order: Customer Details, Problem details, Answer details, a row of buttons (Problem resolved, Refer problem, Audit trail and Customer contact). Within the Customer details section, left to right order was: Customer name, Contact, Login, Department and Customer type.

For the layout in the running software, see Figure 6.8.

UC12 (Software design proposal): *The form interface will include a dropdown menu of subfields for customer contact (verifying DC8).*

In the paper prototypes, see UC9 and DC8.

In the designed new software, see UC9 and DC8.

UC13 (Software design proposal): *The form interface will include a dropdown menu of departments.*

In the paper prototypes, the Department field of Customer Details had a dropdown list of faculties which had been developed in prototyping the Reception form. See DC7, DC12 and DC13.

In the designed new software, there was a dropdown list on the Department field containing the following items: Departments- Academic, Departments- Administrative, Departments - Services, Institutions - Linked to QMW, Institutions - External to QMW.

UC14 (Software design proposal): *The form interface will include a dropdown menu of four customer types.*

In the paper prototypes, the dropdown list on the Customer type field contained: Staff, Postgrad, Undergrad, Other.

In the designed new software, the dropdown list on the Customer Type field contained: Staff, Postgrad, Undergrad, Other.

UC15 (Software design proposal): *Verifying layout of paper prototype customer details fields is in order of convenience for the user.*

In the paper prototypes, this confirmed the layout from UC1.

In the designed new software, see UC1 and Figure 6.10.

DC1 (Work requirement): *Contact details is the element of a customer's details which is most important both to the user and to the user's customer.*

In the paper prototypes, the Contact field was placed second in the Customer Details after Customer name. See UC2, UC3, UC4, UC5 and UC 7.

In the designed new software, fields for the customer's contact details were redesigned, with a Method of Contact field coming in the middle of the second row of fields (see Figure 6.10).

DC2 (Software requirement): *Contact field should contain the customer's phone number and email.*

In the paper prototypes, the Contact field dropdown list included e-mail and phone. See also UC5, UC9 and UC10.

In the designed new software, the Method of Contact field had a dropdown list which included e-mail and phone. The design intention was that the user would select one of these from the list and then type beside it in the field the actual email address or phone number.

DC3 (Work design proposal): *Description of the user's problem should be broken down into initial description and further details.*

In the paper prototypes, Problem details included an 'Initial problem description' field. Answer details included a 'Most recent work' field (which was a history field which should contain subsequent problem descriptions).

In the designed new software, there was a Summary Description field which held the description of the problem originally given by the user and a Work done field which held a history of work done on the problem, including more detailed descriptions of the problem elucidated during the work (see Figure 6.8).

DC4 (Software design proposal): *A similar order of fields on the form is suggested by the order in which customers volunteer information and the order which is most convenient for the users.*

In the paper prototypes, see UC1, UC2, UC3, UC4, UC6, UC8, UC11 and DC1.

In the designed new software, see also UC1, UC2, UC3, UC4, UC6, UC8, UC11 and DC1.

DC5 (Software design proposal): *The form interface should include a dropdown menu for the customer type.*

In the paper prototypes, there was a dropdown list on the Customer type field with items Staff, Postgrad, Undergrad and Other.

In the designed new software, a dropdown list on the Customer Type field also contained: Staff, Postgrad, Undergrad, Other.

DC6 (Software design proposal): *Customer types staff, postgrad and undergrad could be on a dropdown menu.*

In the paper prototypes, see DC5.

In the designed new software, see DC5.

DC7 (Software design proposal): *The form interface should include a dropdown menu for departments.*

In the paper prototypes, the Department field of Customer Details had a dropdown list of faculties which had been developed in prototyping the Reception form. See UC13, DC12 and DC13.

In the designed new software, there was a dropdown list on the Department field with the following items: Departments- Academic, Departments- Administrative, Departments - Services, Institutions - Linked to QMW, Institutions - External to QMW.

DC8 (Software design proposal): *The form interface should include a dropdown menu for the contact field.*

In the paper prototypes, there was a Contact field with a dropdown list of subfields labelled e-mail, phone, post and fax. See UC9 and UC12.

In the designed new software, the subfields of Method of Contact, which were on a dropdown list, were: In person, Telephone, Electronic mail, Post, Facsimile, No re-contact.

DC9 (Software design proposal): *The dropdown menu for the contact field should include phone number and email.*

In the paper prototypes, see DC8, UC9 and UC10.

In the designed new software, see DC8.

DC10 (Software design proposal): *Subfields of contact must allow the user to type in the means of contact.*

In the paper prototypes, the dropdown list on the Contact field was comprised of labelled subfields into which the user could type.

In the designed new software, the dropdown label selected was placed in the Method of Contact field. The user could then type in the corresponding phone number, email address or whatever.

DC11 (Software requirement): *The user is interested in filling in the contents of the subfields of contact, not just in their labels.*

In the paper prototypes, see DC10.

In the designed new software, see DC10.

DC12 (Software design proposal): *Form should have cascading menus for customer's department.*

In the paper prototypes, see UC13.

In the designed new software, see UC13.

DC13 (Software design proposal): *Places Post-It™ with list of faculties.*

This was a direct representation contribution to the paper prototype (see UC13, DC7 and DC12 and Figure 6.z).

In the designed new software, the dropdown list of departments appeared but the content was changed from the paper prototype version (see UC13).

DC14 (Software design proposal): *Highlighting a faculty produces a cascading menu of departments within that faculty.*

In the paper prototypes, this was represented by a triangle symbol to the right of each faculty name listed on the dropdown list. The effect of selecting one of the names was not explicitly modelled.

In the designed new software, there were extensive cascading lists of faculties and departments.

DC15 (Software design proposal): *The user will type in customer name, login and contact details and select from menus customer type and department.*

In the paper prototypes, Customer type and Department may be selected from dropdown lists. Customer name, Contact and Login may be typed in.

In the designed new software, Customer type and Department were selectable from dropdown lists. Customer name, Method of Contact and Login Id were typed in. A label for Method of Contact could also be selected from a dropdown list (see DC10).

The downstream trace from provisional contributions to CG(O) within the CS7 sample through subsequent development artefacts is summarised in Figure 6.9.

Figure 6.9: Embodiment of CS7 contributions in subsequent artefacts

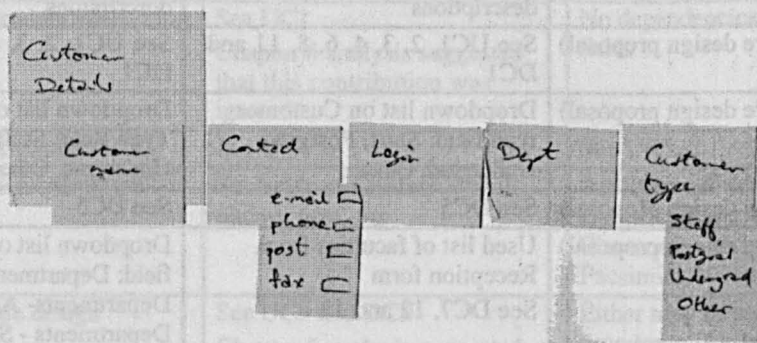
CS7 Software design (paper prototyping)	Paper prototype	New software
UC1 (Work requirement)	Order: Customer name, Contact, Login, Dept, Customer type	Order: Customer Name, Login Id, Department, Customer Type, Method of Contact, Previous contact
UC2 (Work requirement)	See UC1	See UC1
UC3 (Work requirement)	See UC1	See UC1
UC4 (Work requirement)	See UC1	See UC1
UC5 (Work requirement)	See UC1 Chapter 6 analysis suggested that this contribution was ignored	See UC1
UC6 (Work requirement)	See UC1	See UC1
UC7 (Work requirement)	See UC1 Chapter 6 analysis suggested that this contribution was ignored	No dependencies between the fields
UC8 (Software requirement)	See UC1	See UC1
UC9 (Software requirement)	Subfields of Contact: e-mail, phone, post, fax	Subfields of Method of Contact: In person, Telephone, Electronic mail, Post, Facsimile, No re-contact
UC10 (Software design proposal)	See UC9 and DC9 Chapter 6 analysis suggested that this contribution was ignored	Either may be selected from dropdown list, but selecting a second overwrites the current contents of the field. Anything may be entered in the field manually but the field was limited to 15 characters maximum

UC11 (Software design proposal)	Top to bottom: Customer Details, Problem details, Answer details, row of buttons (Problem resolved, Refer problem, Audit trail and Customer contact)	Top to bottom: Date, Time and Reference number, Customer details, Problem/enquiry details, Referral
UC12 (Software design proposal)	See UC9 and DC8	See UC9 and DC8
UC13 (Software design proposal)	Used list of faculties from Reception form See DC7, 12 and 13	Dropdown list on Department field: Departments- Academic, Departments- Administrative, Departments - Services, Institutions - Linked to QMW, Institutions - External to QMW
UC14 (Software design proposal)	Dropdown list on Customer type field: Staff, Postgrad, Undergrad, Other	Dropdown list on Customer Type field: Staff, Postgrad, Undergrad, Other
UC15 (Software design proposal)	Confirms layout from UC1	See UC1 and Figure 6.10
DC1 (Work requirement)	Contact field placed second in Customer Details after Customer name See UC2, 3, 4, 5, 7	Method of Contact field placed in middle of second row of Customer Details fields
DC2 (Software requirement)	Contact field dropdown list includes e-mail and phone See also UC5, 9 and 10	Method of Contact field dropdown list includes e-mail and phone User could type actual number or address beside label
DC3 (Work design proposal)	Problem details includes 'Initial problem description' field Answer details includes 'Most recent work' field (history field which should contain subsequent problem descriptions	Summary Description field holds customer's original problem description Work done field holds history of work done, including subsequent detailed problem descriptions
DC4 (Software design proposal)	See UC1, 2, 3, 4, 6, 8, 11 and DC1	See UC1, 2, 3, 4, 6, 8, 11 and DC1
DC5 (Software design proposal)	Dropdown list on Customer type field: Staff, Postgrad, Undergrad, Other	Dropdown list on Customer Type field: Staff, Postgrad, Undergrad, Other
DC6 (Software design proposal)	See DC5	See DC5
DC7 (Software design proposal)	Used list of faculties from Reception form See DC7, 12 and 13	Dropdown list on Department field: Departments- Academic, Departments- Administrative, Departments - Services, Institutions - Linked to QMW, Institutions - External to QMW
DC8 (Software design proposal)	See UC9 and 12	Method of Contact field subfields on dropdown list: In person, Telephone, Electronic mail, Post, Facsimile, No re-contact
DC9 (Software design proposal)	See UC9 and 10	See DC8

DC10 (Software design proposal)	Dropdown list on Contact field is comprised of labelled subfields into which the user may type	Label from dropdown list placed in field User could also type in field See DC2
DC11 (Software requirement)	See DC10	See DC10
DC12 (Software design proposal)	See UC13	See UC13
DC13 (Software design proposal)	See UC13 and DC7 and 12	Dropdown list on Department field but different from prototype See UC13
DC14 (Software design proposal)	Submenus availability represented by symbols on main menu Submenus not explicitly modelled	Extensive cascading lists of faculties and departments
DC15 (Software design proposal)	Customer type and Department may be selected from dropdown lists Customer name, Contact and Login may be typed in	Customer type and Department selected from dropdown lists Customer name, Method of Contact and Login Id typed in See also DC2

Contributions in the CS7 sample on the nature and order of customer details fields led in the paper prototype to Customer name, Contact, Login Department and Customer type. As noted above, a user tried to elaborate the requirements for the contact field but a developer, interested simply in ordering the fields at that time, ignored the user's contributions. This resulted in the developers' missing the point that the contact aspect had not been thought through.

Figure 6.10: Customer details fields in the CS paper prototype



In the paper prototype, Customer details had fields for Customer name, Contact, Login, Department and Customer type. Customer type had a dropdown list of options: Staff, Postgrad, Undergrad and Other (see Figure 6.10). The design proposal was that selecting one of these options filled in the Customer type field with that option. Contact had a dropdown list of e-mail, phone, post and fax. Following DC10 and DC11, each option had an accompanying field (see Figure 6.10). The design proposal was that selecting one of these options placed that

option as a label in the contact field to which the user would then append the relevant email address, telephone number and so on.

We then see in the implemented software, instead of a single contact field, a 'Method of Contact' field and a 'Previous contact' field. Both of these fields caused considerable usability problems for users. The meanings of both fields were unclear to users. Many users complained that there should be a 'Customer phone number' field. They were unaware of the intended use of the Method of Contact field for this purpose. The implementation of the Method of Contact field did not facilitate discovering this purpose. Clicking on the arrow beside the field (see Figure 6.11) invoked a list of options (In person, Telephone, Electronic mail, Post, Facsimile, No re-contact) in place of the arrow. Selecting one of these inserted the option in the Method of Contact field (embodying contributions DC8, DC9, UC12, DC2 and UC9). The design intention was that the user should then type the corresponding phone number, email address or whatever into the Method of Contact field to the right of the inserted label (embodying contributions DC10, DC11 and DC15). This intention was not conveyed to the users. They assumed that either you typed into a field or you selected from a list to fill it, not both. This assumption was reinforced by the other fields with dropdown lists (Department, Customer type and Referee) in which the user simply selected from the list to complete the field. There was nothing to distinguish the Method of Contact field from these other fields.

Figure 6.11: Customer details fields in the CS software

Customer details

Customer Name Login Id Department

Customer Type Method of Contact Previous contact

In person

Users failed to see the point in recording that a customer had made contact by, for example, telephone and were disturbed by the apparent lack of a field in which to record the customer's phone number, but did not make the connection between the two usability issues. Had anyone tried to enter, for example, a phone number in the field along with a selected label, she would have come up against a further problem that the contact field was restricted to a maximum of fifteen characters. This was particularly restrictive when the label selected was 'Electronic mail' since it takes up exactly fifteen characters! A user selecting this option could reasonably conclude that no further entry was expected in the Method of Contact field.

In practice, users recorded the customer's phone number or email address in diverse fields and it was then not always clear in which field the information had been recorded. Reception staff instituted a policy of recording the customer's phone number, room number and location of the problem in the 'Summary Description'

field. This clearly suggests a failing in the development work to design into the system appropriate support for recording this information.

Users were completely baffled by the 'Previous contact' field in the implemented software. They could not divine its purpose or what should be entered in it. Such a field had not appeared in any previous artefacts (such as the paper prototypes) and appears to have been added unilaterally by the implementor.

DC3 verified that in previous discussions the description of the reported problem to be recorded on the system was broken down into 'initial description' and 'further details'. In the paper prototype, a distinction was drawn between 'Problem details' and 'Answer details' (the other section of the form being 'Customer details'). A field for 'Initial problem description' was proposed within the Problem details section, while. No field for further details was designed. Within the Answer details section there was a field labelled 'Most recent work'. The design intention was that this field by default displayed the most recently performed work on solving the problem. The user could also scroll back through previous entries in this field for earlier work. It was intended that further details of the nature of the problem should be recorded in the course of the entries on work performed.

In the implemented software, the problem and answer details were collapsed into one section headed 'Problem/enquiry details'. This section included a 'Summary Description' field into which the user typed the initial problem as described by the customer. There was also a 'Work done' field which implemented the 'Most recent work' field of the paper prototype, with the label changed to reflect that the field recorded not only the most recent work but also previous work.

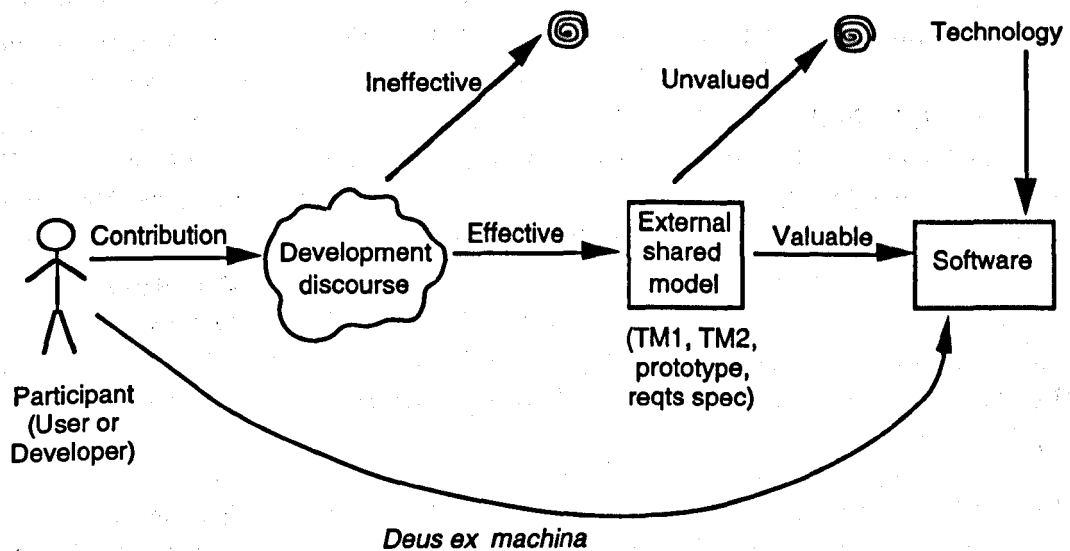
As noted above, Reception staff instituted a policy of recording the customer's phone number, room number and location of the problem in the 'Summary Description' field. The default forms presented on the system to Reception staff were customised to include headers for these details in the Summary Description field. While this ensured that these essential details were recorded, it aggravated another usability issue that the maximum size of the summary description field was felt to be too restrictive. This in turn led to users recording summary description information in the 'Work done' field. Similarly, given the usability problems with the Method of Contact field, some users recorded the customer's email address in the 'Work done' field.

6.5 Discussion

Figure 6.12 illustrates the routes identified in this analysis by which contributions may become embodied in the implemented software or may fall by the wayside during the development process. What this chapter has termed *standard flow* for a contribution is the primary route by which a contribution made by a participant to the development discourse may be transformed through the various development

activities, ultimately being reflected in the software. Such a contribution is termed *effective* in being assimilated into an external shared model and *valuable* in being embodied in the implemented software (see chapter five). Contributions may also be embodied in the software, as *deus ex machina*, without having previously been introduced to the cooperative development activities. Contributions to the development discourse which do not find their way into the software may have been *ineffective* and not assimilated into the development artefacts which preceded the software; or they may have been effective but *unvalued*, appearing in earlier artefacts but not transformed into features of the software. In addition, features of the software may appear as an epiphenomenon of the implementation *technology*.

Figure 6.12: Contribution routes into software



6.5.1 Deus ex machina

Downstream tracing was not applicable to deus ex machina software features. By definition, contributions of these software features could not be traced downstream from the cooperative development meetings, since they appeared in the implemented software without an apparent source in such meetings.

Perhaps the most striking aspect of the upstream tracing results is the clear majority of deus ex machina usability issues. Features of the software which were introduced deus ex machina accounted for almost three-quarters of usability issues in the CI software and for just over twenty per cent of usability issues in the CS software. Looking at the distribution of deus ex machina usability issues across the five levels of issues for the CS software (Figure 6.3), we find none at levels one and three, one at level two, four at level four and one at level five. For the CI software (Figure 6.4), deus ex machina features accounted for twenty-two level one issues, nine at levels two and three, seven at level four and thirteen at level five.

It is somewhat surprising that in two projects which espoused cooperative development, so many features seem to have been introduced to the software in an apparently uncooperative manner. The main reason for deus ex machina features was a number of 'specification gaps' between design and implementation. In both projects, development work used task models (TM2) to help in the design of an envisioned work situation and paper prototypes to help in the design of envisioned software. Both design representations, the task models and the paper prototypes, failed fully to specify their represented designs. And in both projects, underspecification in TM2 tended to lead to deus ex machina features which raised usability issues at the higher of the levels of task disruption, while underspecification in the paper prototypes tended to lead to deus ex machina features which raised lower level usability issues.

Specification gaps in task models were of two kinds. First, there were areas where those constructing the task model explicitly agreed that there was a gap which they could not fill. In the CI project, for example, a separate developer was producing technically advanced software to support a particular task at the same time that TM2 was being produced. Those constructing TM2 did not know how this new technology would work and so could not model in detail the tasks which relied upon it. That left the implementors to do detailed design of those tasks on the fly. This in turn led to features of the software which raised most of the higher level usability issues in the evaluation. A potential lesson there is that the balance might be shifted to doing more detailed task design *before* embarking upon software design.

Ironically, the second reason for specification gaps in task models was the very strength of common ground built within the teams constructing the task models. As described in previous chapters, CG(O) established within the development team consisted in participants' internal models and shared external models. Thus, part of this common ground (i.e. the internal models) became unavailable when software implementation was not performed by the same people, revealing gaps in the explicit external representations. Also, CG(P) established during construction of the task models could allow a relatively simple symbol in the external representation to carry a lot of meaning which was not conveyed outside the cooperative task modelling situation. A frequent example of unspecified gaps in both CG(O) and CG(P) was a detailed decomposition of a particular subtask being discussed and agreed by the participants but not marked up on the external shared task model, leaving the subtask's representation as a leaf node in the task model and an implicit understanding amongst the participants of its decomposition. This generally occurred when to the participants the decomposition seemed 'obvious'. Specification gaps bridged by common ground *within the group creating the external representation* were more insidious than explicitly recognised gaps in that the existence of the former kind of gaps often did not become apparent until software implementation was attempted, again leaving the implementors to attempt quick and dirty task design.

Specification gaps due to the limitations of paper prototypes are widely reported (e.g. [Nielsen, 1990], [Rudd, Stern and Isensee, 1996]). Some studies suggest no significant difference between paper and software prototypes in supporting detection of usability problems but argue that high fidelity prototypes are better for communicating intended system behaviour to implementors [Virzi, Sokolov and Karis, 1996]. Paper prototypes poorly model the dynamic aspects of software. This resulted, in both projects, in navigational features of the software - how to move from one task to another - being underspecified at the paper prototyping stage. Hence, it was necessary for the implementors to perform detailed software design - otherwise known as hacking! Paper prototypes did not draw out other issues which were later raised by the usability evaluations as absent software features (see unrequested features below). Thus, specification gaps in the paper prototypes led to implementors making poorly supported software design decisions which often led to lower level usability issues, while specification gaps in the task models of the envisioned work led to implementors making poorly supported task design decisions which often led to higher level usability issues.

6.5.2 Technology

Technology driven software features also could not be traced downstream from contributions to cooperative development meetings, since they appeared in the implemented software as a result of the technologies used to implement the software produced in each project.

Features of the software which were introduced by the implementation technology were the second most common source of usability issues in the CI software evaluation and the joint second most common cause in the CS evaluation. Looking at the distribution of implementation technology usability issues across the five levels of issues for the CS software, we find one level one issue, none at level two, one at level three, none at level four and three at level five. For the CI software, implementation technology features accounted for three level one issues, one at level two, three at levels three and four and none at level five.

The distribution of and reasons for technology driven usability issues were rather different in the two projects. In the CI project, the implementors used a popular prototyping and development environment to produce the early versions of the software. All but one of the implementors were not very experienced with this development environment. At times they were unsure what facilities were provided by the development environment to implement particular software features. This led to relatively minor usability issues such as inconsistencies in the effects of mouse clicking or double-clicking on interface objects. In addition, the CI system included very advanced document analysis software. This was technically very new and had some teething problems which led to more severe (level four) usability issues. For example, the software led the user to believe that it could handle date and time information in much more sophisticated ways than it actually could, resulting in considerable confusion.

In the CS project, a software development environment specific to the type of application being produced was used to support implementation. This environment enforced many unwanted features on the implemented software, resulting in usability issues of varying severity. For example, a 'query form' was on screen while the software was running. Users very rarely wanted to use this, but it could not be closed or disabled. Similarly, it was not possible to include a piece of text in a window unless it was attached to an interactive object. Thus, in order to include headings for the sections of a form, the implementor was forced to introduce spurious radio buttons as the least intrusive object to which a heading could be attached.

The implementation also relied on the software development environment for the provision of several navigation and customisation features which raised usability issues because users were unaware of the presence of these facilities. These features are described in section 6.5.7 below.

6.5.3 Standard flow

The upstream tracing analysis identified this as a third source of usability issues raised by features present in the software. Looking at the distribution of standard flow usability issues across the five levels of issues for the CS software, we find one at each of levels one and two, none at level four and two at levels three and five. For the CI software, standard flow features accounted for three level one issues, one level four issue, two level five issues and none at levels two and three.

Upstream tracing suggested that two of the level one standard flow usability issues raised for the CI software were the result of the new design slavishly following current practices. Both these issues were raised by features of current paper forms which were embodied directly in newly designed electronic forms. The third level one issue was raised by one of the evaluation subjects with a lot of domain experience contradicting a contribution made by the primary user involved in the development work. This contribution had been made in the course of current work situation analysis and had been transformed through various development artefacts until eventually embodied as a feature of the implemented software.

The level four standard flow usability issue with the CI software had overtones of *deus ex machina*. The primary user had commented on a very early software prototype. The implementors had made changes on the basis of his comments. The features introduced by these changes were disliked by most users in the evaluation. Although this contribution was standard flow in that it was presented in a user-developer discourse, assimilated into the development artefact under construction during the discourse and featured in the implemented software, it was introduced to the prototype software by the implementors in isolation from consideration of the overall design. This may have been exacerbated by the fact that the part of the software in question was one for which the implementors were obliged to fill a specification gap.

The first level five standard flow usability issue with the CI software was a fundamental problem with the underlying notion of the application presenting a group of documents on which the user worked. Users became confused between a model of their work with the application as opening, editing and closing documents and a model of it as completing a task through engaging in a dialogue with the interface. The interface design had been the subject of user-developer discourse particularly at the paper prototyping stage, but, as noted in section 6.5.1, paper prototypes did not adequately represent dynamic aspects of the interaction between user and software. Thus, this issue became apparent only when running software was evaluated.

The second level five standard flow usability issue with the CI software was a case of the design and implementation of the software being perhaps rather too effective. A primary goal of the development project was to provide automation for the user's hitherto manual work of indexing incoming documents. This feature was delivered in the implemented software. However, users in the work domain are taught to 'read, research and record' the document which they are indexing. Users in the evaluation were so impressed by the automatic document parsing feature that they universally reduced this work practice to 'research and record' based on this feature's output, neglecting to read the incoming document. Whilst this may make for efficient work, it may also undermine established working practices and, perhaps more importantly for ultimately effective work, the user's overview knowledge of the documents which have passed through her hands.

The level one standard flow usability issue with the CS software also had overtones of *deus ex machina*. The size of the summary description field was discussed during cooperative development meetings. The conclusion was that it could reliably be determined only after some use of the software. Hence, a specification gap was deliberately created and filled by an implementation decision. The evaluation revealed that users felt the maximum size of the field was inadequate.

The level two standard flow usability issue with the CS software was due to users' difficulty in selecting a customer's department from the large cascading list. Difficulties included the sheer number of departments which appeared, a lack of familiarity with the hierarchical structure reflected in the lists and the obscuring of the parent list by sublists which popped up. (This same problem may be seen, for example, with the hierarchical *Bookmarks* menu in Netscape Navigator 3.0.) Some frustrated users ended up guessing an appropriate department or choosing one at random. This inevitably undermines the integrity of the database.

The first level three standard flow usability issue with the CS software was that many users were confused by the sheer number of fields presented to them, were unclear which fields were essential and had no idea of the purpose of some fields. The task design work in the project had concentrated on removing redundancy from the system by designing to support the common features of work performed in

different roles. Inadequate attention was paid to the *differences* in required task support between different roles. This problem was exacerbated by deus ex machina issues when the implementor copied substantial sections of interface code between different forms for ease of implementation, regardless of the specific requirements for the different forms.

The second level three standard flow usability issue with the CS software was that users sometimes forget to change the status of an enquiry which they had worked on because the status buttons happened to be outside the currently visible portion of the window when they had finished their work on the enquiry. Again, this was an example of a design feature which had come through the cooperative development process, including paper prototyping, and whose potential problems had not become apparent until running software was evaluated.

The first level five standard flow usability issue with the CS software was the inability, described in section 6.4.2.1 above, to provide the customer with a reference number when the user did not enter the enquiry details as the customer provided them.

The second level five standard flow usability issue with the CS software was the confusion created amongst users by the names of fields 'method of contact', 'previous contact', 'current answerer', 'past answerer' and 'date of latest action'. The requirements and designs for these fields had been discussed at length during the course of the cooperative development activities. Relevant contributions had been assimilated into development artefacts and eventually transformed into the features of the implemented software. However, as noted in section 6.4.2.3, the implementation of these features was undermined by developers' ignoring the user's contributions which suggested that the requirements and designs had not adequately been thought through. The software features were then further undermined by the detailed and unsupported redesign carried out in the final stages of implementation by a newly hired developer and the lead user.

Standard flow contributions were also embodied in features of the CS software whose apparent but false absence raised usability issues. These are described in section 6.5.7 below.

6.5.4 Ineffective contributions

Upstream tracing identified this as a third source (after implementation technology and standard flow) of usability issues associated with features which users wanted but were absent from the software. Only two such usability issues were identified by the upstream analysis. One was the recurring usability issue that users of the system found it difficult to log details of a query on the system while interacting with the customer. So users tended to deal with the customer first, perhaps noting details with pen and paper, and entering them on the system later. One repercussion of this was that the enquiry might not be logged at all. Another issue caused by this

was the inability to provide the customer with a reference number for the logged query. Yet a third related issue was that users working at Reception wished to be provided with paper forms with the same layout as the software form in order to facilitate recording and transfer of the query.

The second usability issue identified in the upstream analysis as resulting from an ineffective contribution was the lack of support for recording the room number and location of a problem reported by a customer. This had been presented as a requirement in an early interview with users but had not been assimilated into any of the development artefacts through analysis, design and implementation. In use, Reception staff instituted a policy of adding location and room number information to the summary description field.

As described above, the downstream tracing analysis suggested that contributions UC5, UC7 and UC10 to the paper prototyping activity in sample CS7 were ineffective. In UC5, a user suggested that the customer's name is part of what needs to be recorded in a 'contact' field. In UC10, the user went on to note that the contact field includes data such as the customer's phone number and email. In UC7, the user suggested that a single contact field might not have been an appropriate design and that more than one related field might be necessary; for example, to relate a customer's name to a phone number. The analysis of chapter five suggested that these contributions were not assimilated into CG(O). The point was missed by the developers that the requirements for customer contact details had not adequately been thought through and the implementor was subsequently left unsupported to think through the software design of these features.

6.5.5 Unvalued contributions

The upstream tracing analysis identified this as a fourth source of usability issues associated with absent features of the software. Unvalued contributions are those which were effectively assimilated into the common ground of the development team participants but which were not subsequently reflected in the implemented software. Upstream tracing identified only one example of a usability issue associated with an unvalued contribution in the CI software and two examples in the CS software.

During the CI software evaluation, experienced *Windows* users were frustrated that the software did not use standard *Windows* accelerator keys (e.g. <ctrl>-<tab> and <alt>-<tab>) to facilitate fast switching amongst documents and applications. During cooperative paper prototyping in the CI project, a user had proposed that this feature was implemented. The chief developer accepted this proposal and promised that it would be implemented. This feature was not implemented in the evaluated software, due to time constraints on the implementation and a low priority given to this feature by the implementors.

The first example of a usability issue in the CS evaluation associated with unvalued contributions was the absence of facilities to provide overviews of logged enquiries for management purposes. One of the users who took part in the evaluation and who also fulfilled a management role wanted to be able to produce monthly reports from the logged information. The development environment used to implement the system provided basic reporting facilities which were found to be inadequate and largely unusable. A requirements specification produced very early in the project, based on interviews with users, included a requirement for reporting management statistics by customer department, by staff member, by type of query and so on. This feature did not appear in subsequent development artefacts and was not implemented in the evaluated software. Similar to the example of an absent feature in the CI software, this feature was not implemented due to time constraints on the implementation and a low priority given to this feature by the development team throughout the project.

The CS evaluation also revealed that part-time Help Desk staff in particular were not familiar with staff specialities and would have liked an on-line list to assist them in identifying appropriate referees for enquiries which they could not resolve. Assistance in identifying appropriate referees was stated as a requirement on several occasions throughout the development work and the requirements specification mentioned in the previous example above included a requirement for an on-line file of staff specialities to aid referee selection. This feature did not appear in subsequent development artefacts, including the implemented software.

By dint of the complete absence of implemented software support for recording Reception information, downstream tracing revealed many contributions within samples CS1, CS5 and CS7 which were unvalued, i.e. not reflected in the implemented software, simply because they contributed to the requirements or design of this part of the software. These numerous unvalued contributions did not raise specific usability issues in the evaluation *because* the Reception form was not implemented and, therefore, was not evaluated.

Downstream tracing also highlighted the second CS usability issue identified by the upstream trace as having its source in an unvalued contribution. Contribution UC3 in sample CS5 suggested that Help Desk staff should be able to identify appropriate referees for customer enquiries. No support for this task was implemented and the evaluation eventually showed that Help Desk staff were often those with least knowledge of staff specialities. The software did provide a facility for quick and easy selection of a referee (see Figure 6.x) but no guidance as to which might be suitable for a given type of enquiry.

6.5.6 Unrequested features

The upstream tracing analysis identified this as a fifth source of usability issues associated with absent features of the software. As with *deus ex machina* and technology features, contributions of these software features could not be traced

downstream from the cooperative development meetings since, by definition, they appeared without being introduced in such meetings.

Upstream tracing identified seven absences of software features which raised usability issues and which had not previously been requested, four in the CI and three in the CS projects. Section 6.5.1 has described the problem of specification gaps. When these were spotted and filled by the implementors, they often resulted in *deus ex machina* software features which raised usability issues. When they were not spotted, they remained as gaps in the software which were then revealed by the evaluation as 'unrequested features' which also raised usability issues.

For example, users of the CI software were frustrated at not being able to expand the width of a particular window. The paper prototyping work assumed that this window would be wide enough. The lack of scaled mapping between the paper prototype and software features such as screen and font sizes prevented the paper prototyping work from picking up this issue.

All four of the absent features identified by the CI software would have been difficult to identify before evaluating a running software prototype. As noted in section 6.5.1, there were navigational issues which became apparent when the software was used. In addition, new opportunities for task supporting features became apparent only when users with domain knowledge interacted with the software and appreciated the facilities which it did offer.

The level two usability issue raised by an absent feature in the CS software related to the lack of on-line help. This facility was not considered at all during the development process as it was agreed very early amongst the developers and lead user that it was an inessential addition which could be provided later. The level four and level five issues arising from features which were absent were raised by users who had not been involved in the development activities. For example, a user expressed a desire during the evaluation to be able to check whether or not a given colleague had any outstanding enquiries. These colleagues worked closely together and wanted to cover work in the other's absence. They were not involved in the development activities and this requirement was never previously mentioned.

6.5.7 False absence

The upstream tracing analysis from the CS software identified five usability issues around software features which the users wanted but believed were absent when in fact the features were available. The apparent absences were caused by a combination of poor design and lack of user training. Such features might in principle have been introduced to the software via standard flow, *deus ex machina* or technology contributions. In fact, the analysis found that the issue of a present software feature's apparent absence at each of levels two, three and four had its source in the implementation technology, while both such issues at level five had their source in standard flow contributions.

The CS software evaluation revealed that users were unaware of the facility to save customisation options for fonts and colours, the facilities provided by toolbar buttons (see Figure 6.x) and the facility to write macros. These three features were provided by the software development environment used for implementation. A combination of unintuitive presentation and lack of user training made these features generally unusable and, so far as these users were concerned, absent from the software system.

The other two usability issues of false absence of software features were traced to standard flow contributions. The first such usability issue was that users did not know where to record a customer's phone number and/or email address. A field in which to record these pieces of information had been designed into the paper prototype of the query form; see the description of the user-developer discussion of a 'contact' field in section 6.4.2.3 and Figure 6.7 above. This field was transformed into a 'Method of contact' field in the implemented software. As described in section 6.4.2.3, users had problems with this field, not realising its intended purpose. Several users complained that there was no field in which to record a customer's phone number or email address. Thus, despite this feature's having been designed into the software through user-developer discourse, assimilation into a development artefact (the paper prototype) and transformation into a feature of the implemented software, the users found this feature unusable to the point of believing it to be absent.

The second feature of the software which had been introduced by standard flow and of which users were unaware was the facility to forward enquiries to colleagues. While most users were content with the implemented means of forwarding enquiries, some users reported being unaware of this facility and using conventional email to forward enquiries. The two primary motivating requirements for the development project were a unified facility for recording customer enquiries and an integrated means of forwarding outstanding enquiries amongst users. That users were unaware of the presence of one of these primary features of the implemented software is a remarkable failing in a project which intended to maintain close working connections between users and developers in order to provide a system which included features which the users required and to maintain users' knowledge and acceptance of the design.

6.6 Conclusions

The work reported in this chapter addressed research question four: how valuable was user participation in the studied projects? Having previously analysed the effectiveness of user participation in terms of the assimilation of users' contributions into the artefacts of the development process, this research assessed the value of user participation in terms of the impact which users' contributions had on the usability of the resulting software. The chapter evaluated the usability of the software produced in the development projects studied and traced both forward

from development activity contributions to features of the software and backward from features of the software to contributions.

Having completed the work reported in this chapter, we find two software systems with high levels of usability but some notable areas of usability problems. This chapter has attempted to relate these specific usability issues to the nature of the development work, cooperative or otherwise, which led to them.

A distinction has been drawn in this chapter between prototype and implemented software. The implemented software was taken to be the version which was subjected to the usability evaluation, while any previous version was a prototype⁵. This distinction is essentially arbitrary, as it is with any software product which goes beyond a first release. A decision is taken, often for contractual or marketing reasons, to call a particular incarnation of the software something like version 1.0. All previous incarnations thereby become 'prototypes' and all subsequent versions 'upgrades'.

Whilst such a decision must inevitably be taken, it has ramifications for whether a contribution is considered as standard flow or as *deus ex machina*. Sweeney, Maguire and Shackel [1993, p.692] note that 'the time of evaluation ... dictates the representation of the product which is available for testing and hence the kinds of questions which may be posed in the evaluation'. In chapter four, a distinction was drawn between the software which is the object of a development activity (software design) and a prototype which is an external shared model of that object. That is, in the concept of CG(O), the software is O and the prototype forms part of CG. So, treating the evaluated version as 'the software' leads to features which appear in the software but not in previous development activities being considered as *deus ex machina*. Treating the evaluated version as a prototype and the evaluation as part of the ongoing cooperative development activities leads to some such features being considered as initial contributions to standard flow and their associated usability issues cease *ipso facto* to be usability issues with the software *product*.

It is, therefore, not surprising to find features of the eventual software product which were absent from the prototype. That is a point of using a prototype: to find out what needs to be added, removed or amended. In the same way, for example, there were contributions to the prototypes which had not been explicitly assimilated into the preceding designed task models. This is part of the iterative refinement process of improving earlier drafts of every external shared model.

Both of the level five standard flow issues in the CS project, concerning customer contact details and query reference numbers, highlight the difference between user presence at development meetings and active, effective user participation. Whilst the software feature which raised these issues had been developed through standard

⁵ The versions evaluated here were themselves prototypes since later versions were developed in both projects.

flow, the usability issues might have been avoided had users' contributions to analysis (in the case of the reference number) and design (in the case of the contact details) not been ignored by the developers. Two other level five issues in the CS project were traced directly to ineffective contributions. If CG(O) does provide the basis for the software designs, we should expect ineffective contributions not to be reflected in features of the implemented software since they did not become part of CG(O). In turn, the corollary of effective user contributions' enhancing software usability is that the absence of software features suggested by user contributions should decrease usability. Indeed, this is what we find with these level five issues.

Similarly, if the usability of the software is directly related to the degree of user-developer cooperation in the genesis, transformation and implementation of the features of that software, then features which were introduced to the software without such cooperation are likely to contribute to poor usability of the software. This is supported by the evidence that features introduced uncooperatively, *deus ex machina*, provided the major source of usability issues in the evaluated software.

The research began from a position that active user-developer cooperation in development work should promote the production of highly usable software. At first sight, usability issues which had their source in contributions which followed the standard flow from user-developer cooperation, through embodiment in development artefacts, to features of the software are the most damaging to this assumption. If the assumption that active user-developer cooperation promotes software usability is correct, we should expect few problematic usability issues to arise from standard flow contributions.

This was indeed the case for the CI project. We find only six usability issues out of a total reported of eighty-one traceable to standard flow contributions (see Figure 6.4). Three of these were quite trivial level one issues. More worryingly, one was at level four and two at level five. Detailed examination, however, reveals that one of the level five issues was only a potential problem suggested by the very success of the design in delivering a highly usable and powerful feature. In fact, the risk of undermining current work practices by the very success of the software had been discussed in earlier cooperative development meetings in the project and had been accepted. The level four issue suggests that user contributions to the development work, while to be encouraged, must be strictly controlled. Here we had a user making design proposals 'on the fly' on the basis of a brief look at a very early running prototype. These proposals were incorporated in subsequent versions of the software and were not well received by users in the evaluation. It seems, unsurprisingly, that unsupported, unstructured design contributions from users are as potentially harmful as those from developers.

This leaves only one, level five, usability issue arising from standard flow contributions. This issue was quite fundamental to the design of the software. The users' confusion over their model of the system seems to have its origins in the

failure of the development team to produce a coherent developers' model of the system. This may be traced back to the transition between task design and software design activities. The development team had produced a comprehensive task model of the designed tasks but were somewhat at a loss as to how to move forward from this model to a detailed design of the software to support these tasks. The tools which they had to support this transition, in the form of paper prototypes, were not wholly adequate, as described above.

For the CS project (see Figure 6.3), we again find six usability issues traced to standard flow contributions, but in this case from a total of only twenty-nine reported usability issues. The first level three standard flow usability issue with the CS software revealed a weakness from not involving a wider range of users in the project's cooperative development activities. Quite a lot of effort went into designing out redundancy across tasks performed by different roles in the systems produced by both projects. This must be balanced by a concern for the differences between such tasks as perceived by the users who fulfil the different roles.

This usability issue and one of the level five CS issues were exacerbated by the quick and dirty implementation of the software. This included the implementor's simply copying large chunks of code from one section of the software to another, thereby presenting a user with a software feature which was quite supportive in performing one task but made no sense at all in performing another. This was done to make the implementor's work easier without regard to the established user requirements and designs. These problems argue strongly that control must be maintained throughout a project to ensure that the results of user-developer cooperation in analysis and design are not lost in implementation.

Blomberg, Suchman and Trigg [1996] report a PD project which was intended to be a collaboration between researchers, developers and work practitioners (i.e. users). Due to constraints on resources leading to a lack of engagement of the developers with the project, the authors, as researchers, found themselves 'in the position of trying to maintain some kind of alignment between the work site and product development activities' [Blomberg, Suchman and Trigg, 1996, p.259]. Similarly, in the projects studied for this thesis, the author, whose presence was motivated by research interest, found himself in the position of PD champion.

In the CI project, with highly motivated and experienced professional developers, the author's withdrawal from contact with the project had no noticeable debilitating effects. However, in the CS project, with much less experienced developers and unresolved differences of design opinion, the project ran into difficulties after the author's departure, with the latter stages of software implementation abandoning both participatory practices and previously established requirements and designs. As noted above, implementation corners were cut by copying code from one application area to another without regard for the envisioned differences in use. Also, some users stated that in the several weeks between initial release of the

system and the evaluation reported in this chapter, they had comments and complaints to make about the system but with the author's departure they no longer knew where to direct their contributions. There may be merit, as Bødker [1996] suggests, in the PD champion's leaving quietly and unnoticed but he had better not turn the lights out when he goes.

The results of the work presented in this chapter support the view that active user participation in development work promotes software usability, with the important *caveats* that control must be maintained through to software delivery to maintain user participation, to carry user contributions forward through downstream development activities and artefacts and to discourage the subversion of participatory work by unilateral and unvalidated contributions, especially from developers.

Further work is also suggested in avoiding design underspecification in task models and prototypes. Such investigations could focus on the representational richness of these artefacts and the problem of transmitting the supporting common ground associated with a representation to those not involved in its construction.

The thesis has now addressed the four research questions set out in chapter two and tackled the motivating issues which underlie them. The next and final chapter concludes the thesis with some reflections on how these questions have been answered and what has been contributed by this research.

Chapter 7

Task based cooperative development: conclusions

In the preceding chapters, the thesis has examined a wide range of issues in the study and practice of cooperative software development. This chapter summarises the thesis, reflects on how far its central issues and research questions have been answered and makes suggestions for further work in these areas. Section 7.1 summarises the thesis, outlining its motivations and methods and providing a brief reminder of the concerns of the preceding chapters. Section 7.2 draws together lessons from the thesis in the areas of building theory, developing research methods and improving software development practice. Finally, in section 7.3, the chapter and the thesis conclude with some suggested directions for future work.

7.1 Thesis summary

The thesis has explored what happens when user and developer are brought together in software systems development and how this may contribute to developing usable systems. In investigating this area, the research applied and integrated a wide range of research methods and techniques to the analysis of extensive and heterogeneous data from two real world development projects.

Amongst other contributions, this research provided analyses of the nature of user-developer interaction in cooperative systems development and the relationship between user participation and software usability. Both practical and theoretical issues motivated this research. Practical issues included: a dearth of reported applications of participatory design and of task analysis in real world development projects; a lack of focus within PD approaches on *predesign* activities in system development; a lack of focus within TA approaches on active user participation in development activities; and a lack of integration of TA and PD methods, despite the popularity of each.

Theoretical issues included: a lack of fine grain analyses of user-developer interaction in cooperative development activities; a lack of analyses of the effectiveness of user participation in development activities; and a lack of analyses

of the impact on software usability of user participation in development. The decision to tackle these issues in the field raised other important practical, theoretical and epistemological issues around the difficulties inherent in studying real world development projects.

In addressing these issues, the thesis focused on four essential research questions in the context of two real world software development projects. The research questions were:

RQ1 What *is* user participation in software development, i.e. what are the processes and activities in which users and developers are involved and how do the participants, especially users, contribute to those processes and activities?

RQ2 Were the projects studied *participatory*, i.e. did users actively contribute to the interaction, rather than being passively present or simply a source of on-line data to be tapped by developers?

RQ3 Was user participation *effective* in the studied projects, i.e. were user contributions assimilated into the external and internal artefacts of the development process?

RQ4 How *valuable* was user participation in the studied projects, i.e. what relations were there between user contributions to development activities and features of the software product's usability?

In answering these questions, the thesis examined user-developer interaction in a short pilot study of cooperative task modelling, in the commercial development of a document analysis system for criminal intelligence agencies and in the in-house development of a fault and query logging system for a computing services department. The commercial project was analysed in depth over development work lasting three years and the in-house project over eight months.

A task based participatory approach to software systems development was proposed and applied in the two real world development projects. In this integrated approach, the respective strengths of TA and PD methods complemented each other's weaker aspects. The PD features encouraged active user participation in the development work while the TA features extended this participation upstream from software design activities to include work situation analysis, requirements analysis and work design.

This research demanded and delivered not only a practical approach to task based cooperative development but also the development of an analytical approach to studying the ensuing development activities. Hence, the thesis has made both practical contributions to real world software development and methodological contributions to computer science research. In a third strand of contributions, resulting from the application of the research method in the analysis of the software

development projects, the thesis has provided detailed analyses of the nature of user-developer interaction in cooperative development work, the effectiveness of user participation in embodying user contributions in the artefacts of the development process and the impact on software usability of user participation.

Chapter two of the thesis set the scene for this research. It included a review of the place of TA and PD in the still brief history of software development. It assessed current software development practice through surveying both the literature and practising developers. Based on this work it raised the issues and research questions outlined above.

Chapter three first provided a description of the task based cooperative development approach. It then presented an overview of the pilot study and the two real world software development projects in which the approach was applied. After outlining the data which was available from these projects, chapter three discussed the choice and development of the research approach which was adopted. This discussion covered the need for an observational approach to the analysis of user-developer interaction, the use of video as a means of collecting data for interaction analysis, the involvement of the author in the studied projects as a participant-observer and the development and refinement of the research approach over the course of the work.

Chapter four reported the work which addressed research question one. An inductive analysis of user-developer interaction in the studied projects was combined with a theoretical analysis drawing upon Clark's (e.g. [1996]) work on common ground. This work generated an account of user-developer interaction in terms of the joint construction of two distinct forms of common ground between user and developer: common ground about their present joint development activities and common ground about the objects of those joint activities.

The work on answering research question two continued to extend the application of the concept of common ground. Chapter five operationalised user participation in terms of contributions to the common ground developed through the user-developer discourse. The thesis then assessed user participation in the development activities in terms of those contributions.

In tackling research question three, chapter five went on to operationalise and to assess the effectiveness of user participation in terms of the assimilation of users' contributions into the artefacts of the development work. These artefacts included internal, cognitive artefacts or models which represent the participants' individual understandings on which their common ground is based and external artefacts such as task models and prototypes.

Finally, chapter six addressed research question four, operationalising the value of user participation in terms of the impact of contributions to the development activities on the usability of the software produced. This phase of the research

traced the assimilation of user contributions into the artefacts of the development process and, ultimately, into features of the delivered software.

The following section summarises the findings and lessons from the research reported in the preceding chapters from three perspectives: building theory, advancing research methodology and improving practice.

7.2 Lessons and limitations

7.2.1 Building theory

A research goal of this thesis was to derive an explanatory account of user-developer interaction which could provide the basis for a theory of user-developer cooperation in software development. In addressing the four research questions, the preceding chapters have gone a considerable way to achieving that goal. Chapter four developed an account of user-developer interaction in cooperative development in terms of their joint construction of shared understandings. These shared understandings were of their joint development activities and of the objects of those development activities: the current work situation, the requirements, the envisioned work situation and the envisioned software system.

A recurring pattern of interaction was identified between user and developer: an invitation to provide information, accompanied by an explicit reference to an external shared model (i.e. a task model or prototype), a central interaction sequence and termination of the sequence, with another explicit reference to the external shared model. The initiating invitation usually was to provide information on or verification of a participant's understanding. The central interaction sequence could be a simple provision of information or verification followed by acceptance and termination of the sequence. On the other hand, the central interaction sequence often consisted of a complex pattern of mutual verification, whether the initial invitation had sought information or verification.

The thesis argued that this pattern of interaction reflected the joint construction of common ground [Clark, 1996] by user and developer. Chapter four postulated the construction of two distinct forms of common ground by the participants in cooperative development work: common ground about the objects of the development activities, CG(O), and common ground about the immediate and present development situation, CG(P). The analysis suggested that participants found the construction of CG(O) difficult, leading to frequent, complex interaction patterns of checking their mutual understandings, as described above.

The participants appeared to have less trouble with the construction of CG(P), with many fewer and less complex interaction sequences devoted to checking their mutual understandings of elements of CG(P). The thesis further suggested that the participants in the cooperative development activities may have been using different

cognitive representations, CG-shared and CG-iterated [Clark, 1996], for CG(P) and CG(O) respectively.

Chapter five further extended the theory of common ground, refining the explanatory account of how participation in cooperative development activities develops and proceeds. This analysis distinguished between successful and effective contributions to the system development discourse. Clark and Schaefer's [1989] model of contributing to discourse describes successfully making a contribution as having it heard and understood as the contributor intended. In their view, the contribution has then been assimilated into the participants' common ground. The analysis in chapter five extends this model to argue that for a contribution to CG(O) in cooperative development to be effective, it must be successful *and in addition* its content must be assimilated into the artefacts of the development process, specifically the internal and external models. In this model, assimilation involves a further step of reconciling (or noting the differences between) the content of the new contribution and the contents of the participant's current understandings.

This model was then used in an assessment of the extent of user participation, in terms of user contributions to the development activities, and the effectiveness of user contributions, in terms of the proportion of contributions which were assimilated into the internal and external models of the development process.

The broad interaction analysis of the development meetings suggested that there were high levels of user participation throughout the projects. This view was supported by the results of the more tightly focused analysis of interaction samples reported in chapter five. From a total of 171 contributions noted across three samples, 78 were made by a user and 93 by a developer. The main patterns of contributions in the sample from a current work situation analysis meeting were unsurprising. The vast majority of contributions were to the declared purpose of the meeting. All the user contributions provided information, verification or representation of features of the current work situation. A large majority of developer contributions invited one of these.

More interesting patterns were apparent in the sample from a work situation design meeting. There was no longer a huge majority of contributions to the officially declared purpose of the meeting. Contributions were quite evenly spread across the development activities with a majority made to current work situation analysis. This suggests the 'inevitable intertwining' [Swartout and Balzer, 1982] of development activities from different phases in the traditional development cycle model. In the early meetings, when nothing but current work situation analysis has been tackled, a huge majority of contributions to that activity is noted. As the meetings are held later in the development project, as in the second sample, the participants are progressively enabled to pursue a much wider range of activities. Instead of just sticking to the declared activity of designing the work situation, they may also

contribute to software design, requirements analysis and current work situation analysis.

Interestingly, of the latter contributions to current work situation analysis, five were developer invitational contributions and nine were user provisional contributions, as should be expected, but eight were developer provisional contributions. There are two likely explanations for this result. First, as the project progressed, the developers gathered information from different users and eventually came to have what was in many cases a wider and more detailed picture of the current work situation than any individual user. What is then found throughout the development meetings is examples of developers informing users about aspects of the current work situation!

Secondly, one of several longer term patterns of interaction which emerged from the analysis was a 'meta-sequence' of: a sequence of design contributions, followed by a sequence of requirements analysis contributions (to justify the design need), followed by a sequence of work situation analysis contributions (to justify the requirement), followed by a return to the original design contributions (terminating the meta-sequence).

From the broader interaction analysis, it appears likely that the first explanation holds for the sample of the work design meeting, while the second effect influenced the contributions in the third, software design meeting, sample. The implications for practice of these results are taken up in section 7.2.3.

Having assessed the levels of contributions across the development activities, the analysis moved on to assess the proportions of these noted contributions which were assimilated into the artefacts of the development activities.

In addition to the analysis of contributions to CG(O) described above, the assessment of effectiveness included analysis of ten user contributions to CG(P) across the three samples. As described in section 5.3.2 of chapter five, a huge majority of user contributions was effectively assimilated into either developers' internal models or shared external models. However, assimilations into internal models, in turn, massively outnumbered assimilations into external models. Chapter five offered two explanations for this.

Complex interaction sequences frequently involved many contributions which were apparently assimilated by the recipient into his internal models. However, in many such cases, only one explicit assimilation was made into an external model, generally right at the end of the interaction sequence - the terminating reference to an external shared model described in chapters four and five.

The second explanation, which has more serious implications for development practice, is that the external shared models did not facilitate recording many contributions which should otherwise have been assimilated. This was usually the

case when the contribution was to a development activity which was not the declared activity of the meeting. As described in section 5.3.2 of chapter five, contributions to requirements analysis were frequently assimilated into task models - whose intended role was to support current or envisioned work situation analysis. However, this type of cross-assimilation was not always possible or convenient. As noted above, the further down the development path, the greater the mixture of contribution types that were offered. In software design sessions, a long way down the development path and hence including many different kinds of contributions, the external shared model was a paper prototype and was wholly unsuited to assimilating contributions to development activities other than software design. This may have resulted in the loss of contributions which otherwise should have been assimilated.

In the final phase of this research, the work reported in chapter six traced the relations between user participation and software usability in two directions, downstream from development work contributions to features of the software and upstream from features of the software to contributions.

A total of 81 usability issues were identified with the CI software. A total of 29 usability issues were identified with the CS software. The software features giving rise to these usability issues defined the scope for the upstream trace. The scope from which to perform downstream tracing was defined as the user's provisional contributions to CG(O) of software design, work design or requirements in the video samples from CS1, CS5 and CS7 which were analysed in detail in chapter five.

Upstream tracing from the usability evaluation results identified three sources of features which appeared in the interface and raised usability issues and four sources for the absence of features whose presence users felt should have enhanced usability.

One source of such absences was unvalued contributions, where a requirement or design proposal for the absent feature had been contributed during the development work but the feature was never implemented. A second source of absent features was that a contribution had been made to the development discourse but had been ineffective (i.e. not assimilated into the common ground of the development team). Thirdly, some software features were absent because they had never been requested or suggested during previous development work. Fourthly, some allegedly absent features were in fact present in all or in part. In the latter cases, the users had not received sufficient training to be aware of the features.

One source of features which were present in the software and which raised usability issues may be termed standard flow through the development activities. A participant makes a contribution to the development discourse which is effectively assimilated into an external shared model and subsequently becomes a feature of the implemented software. The second source of software features which raised

usability issues was the implementation technology. The third source of such features was *deus ex machina*: features which had not appeared in any of the cooperative development activities or in any previous external artefacts and which then suddenly appeared in the implemented software.

As noted in chapter six, if active user-developer cooperation promotes software usability, few usability problems should arise from standard flow contributions. This was found to be the case in the CI project. One of the few serious standard flow usability issues in this project suggests that user contributions to the development work, while to be encouraged, must be regulated and rooted in systematic development work. Design proposals made by a user on the basis of a brief look at a very early running prototype were incorporated in subsequent versions of the software and raised usability issues in the subsequent evaluation.

As a corollary of the assumption that user participation improves usability, features of the software introduced without such participation are likely to contribute to poor usability of the software. This is supported by the evidence that features introduced uncooperatively, *deus ex machina*, provided the major source of usability issues in both projects.

In providing answers to research questions one to four, the contributions of the thesis have included a theoretical account of user-developer interaction in cooperative development in terms of the nature, extent, effectiveness and value of user participation. As noted in chapter three, there is a widespread absence of theory in HCI or, at least, theory which has been demonstrably linked to practice either through induction or application. Dowell and Long have argued that this paucity of theory reflects HCI's status as a craft rather than a science [Dowell and Long, 1988; Dowell and Long, 1989; Long and Dowell, 1989]. In its continuing efforts to build theoretical and scientific bases, the increasingly diverse field of HCI continues to integrate research topics and methods from various traditions (see, e.g., [Monk, Nardi, Gilbert, Mantei and McCarthy, 1993]). The work of this thesis has contributed to the body of research and to the use and integration of research methods from several areas.

It is arguable whether there ever can be, in pure terms, a science of HCI (see, e.g., [Anderson, Heath, Luff and Moran, 1993; Barnard, 1991]), so the type of research approach applied in this work and the partial theoretical accounts and guidelines for practice which came out of it have a value of their own given the current state of the art and may be as much as we can provide for HCI and interactive software development.

The account presented in the thesis of user-developer interaction in cooperative development was primarily derived inductively from analysis of the user-developer interactions in the projects studied and extended existing theory from other disciplines. With the foundations of a theory of user-developer interaction laid, the

next steps in theory building should be in testing and developing the theory. Suggestions for possible next steps are taken up in section 7.3 below.

7.2.2 On research methodology

From its inception, this research was intended to be firmly grounded in real world software development practice. Two major difficulties in researching ongoing software development projects are the infeasibility of running controlled experiments and the frequent absence of established theory on which to base hypotheses. These features presented both problems for the research which the thesis sought to conduct and opportunities for the delivery of research methods and results. The cross-disciplinary nature of the research, contributing to computer science ideas and approaches from ethnography, psychology and linguistics, similarly offered challenges and opportunities. The scope and subject of the research necessitated the simultaneous pursuit of multiple research questions. The nature and range of the research questions, in turn, demanded different but related approaches to answering each. Hence, the thesis had to conduct and to integrate work drawing on a wide range of research traditions.

An important feature of the research was the author's involvement as a participant-observer in the software development projects studied. The author needed to be present to monitor and, when necessary, to encourage the use of the cooperative development approach. The author also had to gather data on the development work. This included both taking a video record of development activities and performing the more ethnographic observation which forms a necessary complement to the video record.

The author's presence as part of the development group made him familiar to the other participants. This in turn made recording and analysing their activities less intrusive. The action research approach of this work both acknowledged the role of the author as participant and allowed the theory and practice of cooperative development to evolve together during the projects.

The thesis relied in large part on video based interaction analysis. This raised several issues. Individual participants were often reluctant to be recorded even when recording had been agreed in principle for a project as a whole. This necessitated some gaps in the video record. However, analysis cannot in any case depend upon a continuous record of all the activities which take place in and around a software development project. Development activities may go on in all sorts of unofficial locations and at unofficial times. For example, participants may meet by chance in a corridor and discuss an issue. On one hand, then, the analysis must be flexible enough to work with a partial video record, supplementing it with insights and data gained from other sources. On the other hand, the analyst must be aware that samples of recorded interaction are taken from video records which are themselves generally unsystematic samples.

The video based work in the thesis was limited by the technology available to the researcher. Only one fixed position camera was available which meant making decisions at the beginning of a session as to what should be recorded. The primary research interest in the interaction of the participants demanded that almost all of the video record was taken with a fairly wide angle. Combined with the quality of recording, this made it very difficult to see on the tapes fine detail of, for example, the representations with which the participants were working. This at times necessitated analysing with a video remote control in one hand and a copy of the representation in the other.

The fixed camera position also meant that at times the wide angle was not wide enough, with participants moving out of shot. It is recommended that future video recording of development sessions should use at least two cameras, perhaps with one fixed on the work surface and one at a wider angle to capture the interaction. A dedicated camera operator should allow fuller and sharper recording but might make the participants uncomfortable. A major strength of the researcher's position in the projects studied here was that he was seen at least in part as a developer, as one of the team. This facilitated access to and interaction with the other participants which could not be so unobtrusive with a larger research team present.

Experienced interaction analysts tend not to transcribe large amounts of video data, preferring to transcribe only short sections of particular interest [Jordan and Henderson, 1995]. The experience of this research suggests that this is a sound strategy. This work expended considerable time and resources on attempting to produce lengthy transcripts of quite short stretches of video record before adopting the more selective strategy. Neither do transcripts provide an adequate transformation of the video record for initial analysis of the data. The video record itself is an incomplete representation of the source data; the transcript is still further removed.

Transcripts did come into their own when the analysis had moved beyond the identification of broader patterns in the interaction (chapter four) and was seeking to identify instances of specific interactions (chapter five). Here, the relative simplicity of the transcript, compared with the video record, and its concrete, static representation of the interaction facilitated the coding and counting of already specified elements of interaction. Again, the uses of video and transcript illustrate the integration of research techniques for related analytical work. The different media respectively supported the complementary macro and micro analyses.

In interaction analysis there remains a need for a better understanding of what are relevant and irrelevant phenomena. As described, for example, by Bamberger and Schön [1991], interaction analysis generally feels its way into a situation of interest with little or no preconception of what are relevant phenomena. Hence, the use of analytic foci as 'ways-into-a-tape' [Jordan and Henderson, 1995, p.57]. While this can be a valuable (indeed, often the only possible) approach when analysing

interactional situations for which there is little previous analysis or from which the analyst wishes to induce fresh analytic insights, it does have several disadvantages. It is immensely time consuming, involving continual reviewing of the interaction as a perspective develops and frequent analytic deadends. It is also hard to assess the internal validity of the accounts generated.

The contributions of this thesis include the framework introduced in chapter five for analysing participation in cooperative development activities in terms of provisional and invitational contributions by the participants of information, verification and representation. Supported by the concept of common ground as a unifying goal for the participants in development activities and the characteristic pattern of interaction identified in this research for the construction of that common ground (see chapter four), the framework may be used by other researchers in analysing user-developer interaction in software development activities.

The framework provides the analyst of user-developer cooperation with a means of rapidly and systematically identifying, and if desired quantifying, relevant phenomena. Thus, the framework offers a set of concrete analytic foci which may be used to facilitate the analysis of user-developer interaction. In turn, the application of this framework by other researchers should provide a check on the validity and reliability of the analysis presented in this thesis. Some further uses for this framework are suggested in section 7.3 below.

7.2.3 Improving practice

The ultimate aim of an applied discipline, such as HCI, must be the improvement of practice. The indicators that this was achieved in this work are strong. The computing services department of the CS project uses the system which was developed. The system developed in the CI project is successfully being sold around the world. More importantly to improving practice through industrial penetration of HCI, the software development company have now set up an HCI Group. The work of this group has already included usability evaluations of new and established products and major redesigns of at least one application interface.

Practical lessons on cooperative software development may be drawn from throughout the thesis. The survey work of chapter two revealed a picture of current practice amongst software developers while the remainder of the thesis suggested potential improvements to practice and provided an evaluation of the practical impact of task based cooperative development on the resulting software systems.

The survey suggested that, despite the popularity within the research community of participatory design and task analysis, these methods receive only lip service from software development practitioners. A major impediment to the uptake of systematic cooperative development is the reluctance of many at managerial level in both user and developer organisations to allow contact between users and developers.

A common fear in commercial projects, illustrated in the CI project, is that competitors may receive commercially sensitive information if anyone outside the development organisation is allowed access to the development work. This was overcome very effectively in the CI project by taking as representative user someone who could provide the required input to the cooperative development sessions and who could be trusted not to reveal commercial secrets. Interestingly, a similar tactic was used by the development organisation when the author was employed on a short term contract specifically to run the usability evaluations of the developed software.

In the CS study, there was enthusiasm for user participation in the project from the users' management and from the lead user charged with overseeing the project for the user organisation. Reasons expressed for the non-involvement of particular users ranged from apathy to lack of knowledge to never quite finding the time. However, there was steady involvement from a number of users who each had a strong personal interest in influencing the design of the software product and any associated work design.

While the focus of this research was on the nature of user-developer interaction *within* the cooperative development activities, i.e. after working contact has been set up between user and developer, a crucial issue in cooperative development remains setting up that working contact and creating the opportunity for the interaction. This work suggests that a key prerequisite for the initiation of cooperative development work is educating managers and developers about the benefits of cooperative development approaches. As suggested in chapter two, if managers and developers are to allow and to facilitate the use of cooperative methods and techniques, they must be informed of what the latter involve and must be convinced of their value. This thesis contributes both to informing about cooperative methods and techniques and to assessing the value of their use.

Another important lesson for practice is that to promote active user participation in system development work, one should ensure that the users are highly motivated to participate. User motivation may come from being employed by the development organisation, from wanting to make sure that their own work is not made more onerous in any redesign or from other factors which may be identified as motivating *the particular user* to active participation. Grudin [1991b, 1993] identifies several obstacles to user involvement, including gaining access to users and motivating developers. But having gained access in principle, user motivation, whatever its source, is essential. When user and developer are brought together to collaborate in development work, the developer is in the room because it is his job to be there. It is not (in general) the user's job to be there. In fact, the user's job is somewhere else and every moment spent working with the developer is a moment lost to the user's real work. Hence, user motivation is crucial.

The pilot study and the CI and CS projects provided insights into the use of task models and paper prototypes. The main problems revealed by the pilot study were in the poor support provided by the modelling tools for flowing, dynamic cooperative work. In the CI project, pencil and paper proved to be much more efficient media for task modelling than the coloured cards used in the pilot study. The electronic and wallchart versions of the task model in the CI project and the whiteboard version in the CS project provided several advantages and disadvantages, discussed in chapter three. Issues around the use of the external shared representations are taken up more fully in O'Neill, Johnson and Johnson [1997] and in O'Neill, Johnson and Johnson [under review].

A strength of the cooperative development approach used here is the tight binding of analysis and modelling through the cooperative sessions. However, it is worth noting again (see section 3.2.1 of chapter three) that the modelling difficulties in the pilot study may have been exacerbated by attempting to model from a 'cold start', with *no* prior analysis of the user's work situation. Hence, cooperative modelling should be preceded by the developer's becoming at least vaguely familiar with the target work situation. The developers' bootstrapping the cooperative modelling work (in the CS and CI projects) with at least a rough outline of some of the roles to be modelled greatly facilitated the users' initial involvement in the modelling work.

The main analyses of the thesis provided not only a theoretical account of development work but also recommendations for its practice, many of them driven by the theoretical insights. The thesis suggests that the coconstruction of common ground is a signifier of the success of a cooperative development project. Indeed, users across the projects found their increased understanding of their broader work situations to be a useful result *per se*. User and developer working together built increasingly strong personal common ground. This is a product of cooperative development not offered by other approaches. On the other hand, building strong personal common ground with developers in a cooperative development situation may contribute to users' losing touch with their own user community and becoming part of a different, software development team community.

The interaction analysis identified a recurring pattern of grounding interaction sequences in external shared models. Typically, interaction sequences were bounded by explicit references to an external shared model, with manipulation of the model marking the end of a sequence. However, the available external shared models did not always lend themselves to the assimilation or appending of contributions made, particularly within complex, recursive interaction sequences.

Common examples of this occurred when software design meetings, with software prototypes as the active external shared model, included contributions to requirements analysis and users' work situation analysis. Whilst such contributions could readily be assimilated into the participants' internal models and might

influence future versions of the software prototypes, they were not, indeed often could not be, assimilated into the prototype in use.

A related issue here is the lack of use of 'formal' requirements specifications in the work between user and developer. They were viewed as an unnecessary overhead in small group work. This often led to assimilation of requirements into the participants' internal models and *ad hoc* appending of requirements to task models.

It may be more effective to focus a meeting on the relevant form of model (task model for work analysis and design, prototype for software design) but also to have permanently visible, and accessible for changes, other forms of representation and structuring. For example, when prototyping it might be productive also to have available modifiable task models and requirements specifications in which to assimilate contributions to work situation analysis or design and requirements analysis.

Another lesson for software development practice is that even in 'cooperative' development projects a user may successfully make a contribution which nevertheless is not assimilated into the developers' understanding, the common ground of the participants or the external development artefacts. It may be worth investigating means of capturing these lost contributions, for example through reviewing video records of project meetings. Although, as Muller [1992] found, developers may have difficulty finding the time and motivation to pore over such video records.

The development approach adopted in the studied projects provided strong grounds for building CG(P), particularly personal CG(P) through the cooperative work in defining and refining the processes and artefacts together. Unfortunately, it provided quite poor grounds for building personal CG(O) since users gained little or no real experience of the developers' working world and *vice versa*. Communal common ground was, inevitably, not strong between developers and users and this too made for relatively weaker CG(O). This has two main implications for improving software development practice.

An implication for sound development practice from the theoretical analysis is the desirability of keeping external shared models (such as task models and prototypes) based on wide world communal common ground and on personal common ground rather than on professional common ground which does not exist between user and developer. There is, on the other hand, the problem with a representation which relies on personal common ground of disseminating the representation beyond the original participants.

There was considerable evidence from the projects for the fulfilment of the objective of maximising user cooperation while minimising user dependence in order to make the best use of potentially limited user access time. In both projects, development work had to press ahead when one or more of the users was unavailable. In every

case, the user's integration into the development process and familiarity with the analysis and design artefacts allowed him to return and to pick up the development work without difficulty. This may be explained in terms of the strength of personal common ground, particularly CG(P), which was built up during the periods of active collaboration with the users.

A further implication for practice is that we need not just the third of Friedman's [1989] strategies for dealing with user relations problems, that is creating arenas for user-developer interaction in development activities, but a combination of at least this strategy and the second strategy of allowing developers to cross the user-developer barrier. However, this second strategy needs to be more than recruiting developers who have an education vaguely relevant to the users' work situation. The developers must actively participate in that work situation in order to construct common ground with the users.

A principle of the development approach adopted in the projects is that development is, and should remain, developer-led. Software development is the job of the software developer, not the user. It is unrealistic to expect users to have the skills, time, motivation and authority to take on the role of developer. This approach promotes cooperation with the developer, not supplanting the developer (i.e. not Friedman's [1989] first strategy). There was considerable evidence from this study that attempting to place development in the hands of users could not succeed. The user, although eager and cooperative, retains a confidence in the developer as having requisite technical knowledge and modelling skills that the user cannot be expected to acquire in such depth.

Chapter six found that features of the software which were introduced *deus ex machina* accounted for the majority of poor usability issues. In both projects, design underspecification in the paper prototypes tended to lead to *deus ex machina* features which raised lower level usability issues. Specifically, specification gaps due to the limitations of paper prototypes in representing dynamic behaviour resulted in navigational features of the software - how to move from one task to another - being underspecified. Specification gaps in the paper prototypes led to implementors making poorly supported software design decisions which often led to lower level usability issues.

An implication here is that paper prototyping must be complemented by software prototyping. Initial paper prototyping can provide very rapid user input on the emerging software design. Subsequent software prototyping can be aimed both at refining features of the software which were represented in the paper prototype and at evaluating features which were not. In fact, this procedure was followed in both the CI and CS projects. The software versions which were evaluated as part of these studies revealed the specification gaps noted above and were themselves refined to resolve these usability issues in later versions.

Again in both projects, design underspecification in the envisioned task model tended to lead to implementors' making poorly supported task design decisions, producing deus ex machina features which raised usability issues at the higher levels of task disruption. Two reasons for specification gaps in task models were identified. First, there were areas where those constructing the task model explicitly agreed that there was a gap which they could not fill. That left the implementors to do detailed design of those tasks on the fly. This in turn led to features of the software which raised most of the higher level usability issues in the evaluation. A potential lesson there is that the balance might be shifted to doing more detailed task design *before* embarking upon software design.

The second reason for specification gaps in task models was the very strength of common ground built within the teams constructing the task models. As described in previous chapters, CG(O) established within the development team consisted in participants' internal models and shared external models. Thus, part of this common ground (i.e. the internal models) became unavailable when software implementation was not performed by the same people, revealing gaps in the explicit external representations. Also, CG(P) established within the cooperative development group could allow simple symbols in the external representation to carry a lot of meaning which was not conveyed outside the cooperative task modelling situation.

Specification gaps bridged by personal common ground constructed by the cooperative development participants were more insidious than explicitly recognised gaps in that the existence of the former kind of gaps often did not become apparent until software implementation was attempted, again leaving the implementors to attempt quick and dirty task design. Methods of formal specification (for example, Z [Spivey, 1992]) attempt to produce complete specifications but have very limited application. Their use depends upon a professional common ground which, as noted above, is absent in user-developer interaction. Proponents of formalisms such as Z also concede that they do not present complete specifications when, for example, they insist on each section of formal notation being accompanied by an explanation in natural language: another specification gap which relies for its bridging on common ground.

Another potential solution is to involve the implementors as participants in the earlier development activities, so making them partners in the personal common ground which is constructed therein. This, however, is not an approach which could be followed in many software development situations, especially very large projects with multiple development teams each with several members. The most effective and efficient means of resolving specification and implementation gaps due to gaps in common ground is likely to remain rapid iteration around the design, evaluate, redesign cycle.

7.3 Future work

The research reported here has contributed to our understanding of cooperative development, described the application of one form of such development in real world projects and assessed the effects of cooperative development work on the development artefacts and the software product.

The studies enjoyed wide coverage, examining the development of two very different software systems in two very different environments. Unusually for studies of real world development projects, this work was able to apply the same development approach across both projects.

The analysis, however, has some limitations. The groups examined were small and used a development method with considerable flexibility. The projects were 'green field' developments in which there was considerable freedom in envisioning software designs. There are of course many other kinds of software system intended for many different kinds of user and built using many different development approaches.

It is probable that the findings of this study apply to other development projects with a cooperative approach and small group sizes. It is possible that they may apply, perhaps less strongly, to larger development teams working within more rigidly structured development methods. They clearly apply to the development of new software systems but may have less application to projects aimed at maintenance of existing software. In maintenance projects, there is typically less freedom to produce innovative software designs. However, in maintenance development similar issues arise of revealing and integrating multiple understandings and of designing to meet users' requirements.

Comparative studies of diverse development projects should illuminate these questions and contribute to the establishment of a corpus of real world software development project studies. However, longitudinal studies of software development projects are notoriously difficult to set up and to run.

Even given the small group size, the account of interaction within the groups presented by this analysis was almost wholly in terms of a single participant's joint construction of common ground with a single other participant. Chapter four briefly alluded to the more complicated situation of multiple participants simultaneously attempting to achieve overall common ground. There is much scope for investigating whether or not people actually do attempt this feat and, if so, how. Previous work on the role of overhearers (e.g. [Clark and Schaefer, 1987b; Schober and Clark, 1989]) may be usefully applied here.

The framework used in chapter five to analyse contributions to the development discourse provides the analyst of user-developer cooperation with a means of rapidly and systematically identifying, and if desired quantifying, relevant

phenomena. As noted in section 7.2.2, the application of this framework by other researchers should provide a check on the validity and reliability of the analysis presented in this thesis.

Furthermore, the framework may also be used in conjunction with coding schemes for identifying and/or counting contributions other than that based on the four types of development activity used here. Thus, the framework may be used to support the analysis of wider interactional situations where the construction of common ground is important.

It might also prove fruitful to use the framework in analysing interaction around other forms of external model such as, for example, formal notations. The patterns of interaction which such an analysis should reveal might then be compared for the uses of various forms of representation. In turn, this could provide insights into the utility of different representational forms in supporting the development of shared understandings between different types of participant.

This research began with an exploratory approach to theory building. Having developed a theory of user-developer interaction in the cooperative setting in the course of this thesis, further work might generate a set of predictions based on this theory and design and run experimental tests of these predictions. One notion potentially amenable to experimental testing is that the construction of CG(O) is more difficult than the construction of CG(P). In order to test this, the investigator must operationalise 'difficult' to some experimentally measurable variables. This work and the design of a suitable experiment are likely to prove challenging.

Experimental testing might also be applied to the suggestion that the participants in the cooperative development activities used primarily CG-shared as a cognitive representation in the construction of CG(P) and primarily CG-iterated as a cognitive representation in the construction of CG(O). This may be made amenable to experimentation by, for example, the prediction that those working with an external representation, such as a task model, will use CG-shared for (in Palmer's [1978] terms) the representing world and CG-iterated for the represented world. Again, however, designing and running an experiment to test this is not a trivial exercise.

To answer the theoretical question of which cognitive representation is used, we need an experimental design in which we can oblige the participants to perform tasks which we predict can only be performed with one or the other representation. The design of such an experiment is a massive undertaking, beyond the scope of the present work.

Much of the analysis in the thesis has built upon theoretical work on common ground from the field of psycholinguistics. There are of course many other fields which may offer productive theoretical insights into the human activity of software systems development which is the subject of this research. Design theory, systems analysis, psychology, anthropology and many other disciplines offer additional

perspectives. Further research may provide theoretical accounts of user-developer interaction which complement, confirm, contradict but ultimately add to the results of this research.

References

Ackoff, R.L. (1974). *Redesigning the future*, New York, John Wiley and Sons.

Agresti, W.W. (1986). The conventional software life-cycle model: its evolution and assumptions. In W.W. Agresti (ed) *New paradigms for software development*, Washington DC, IEEE Computer Society Press, 2-5.

Anderson, R.J. (1994). Representations and requirements: the value of ethnography in system design. *Human-Computer Interaction* 9, 151-82.

Anderson, R.J., Heath, C.C., Luff, P. and Moran, T.P. (1993). The social and the cognitive in Human-Computer Interaction. *International Journal of Man-Machine Studies* 38, 999-1016.

Atkinson, J.M. and Heritage, J. (eds) (1984). *The structures of social action*, Cambridge, Cambridge University Press.

Bach, K. and Harnish, R.M. (1979). *Linguistic communications and speech acts*, Cambridge, MA, The MIT Press.

Bamberger, J. and Schön, D.A. (1991) Learning as reflective conversation with materials. In F. Steier (ed) *Research and reflexivity*, London, Sage, 186-209.

Bannon, L.J. and Bødker, S. (1991). Beyond the interface: encountering artifacts in use. In J.M. Carroll (ed) *Designing interaction: psychology at the human-computer interface*, Cambridge, Cambridge University Press, 227-53.

Barnard, P. (1991). Bridging between basic theories and the artifacts of Human-Computer Interaction. In J.M. Carroll (ed) *Designing interaction: psychology at the human computer interface*, Cambridge, Cambridge University Press, 103-27.

Baroudi, J.J., Olson, M.H. and Ives, B. (1986). An empirical study of the impact of user involvement on system usage and information satisfaction. *Communications of the ACM* 29 (3), 232-38.

Bauer, F.L. (ed.) (1977). *Software engineering: an advanced course*, Springer-Verlag.

Beck, A. (1993). User participation in systems design - results of a field study. In M.J. Smith and G. Salvendy (eds) *Human-Computer Interaction: applications and case studies*, Volume 19A, Amsterdam, Elsevier North-Holland, 534-39.

Bellotti, V. (1988). Implications of current design practice for the use of HCI techniques. In D.M. Jones and R. Winder (eds) *People and Computers IV*, Cambridge, Cambridge University Press, 13-34.

Bellotti, V. (1990). Applicability of HCI techniques to systems interface design. PhD thesis, Queen Mary and Westfield College, University of London.

Benyon, D. (1992). The role of task analysis in systems design. *Interacting with Computers* 4 (1), 102-23.

Blomberg, J., Suchman, L. and Trigg, R.H. (1996). Reflections on a work-oriented design project. *Human-Computer Interaction* 11, 237-65.

Bloomer, S., Croft, R. and Wright, L. (1997). Collaborative design workshops: a case study. *Interactions*, January and February 1997, 31-39.

Bødker, S. (1991). *Through the interface: a human activity approach to user interface design*, Hillsdale, N.J., Lawrence Erlbaum.

Bødker, S. (1996). Creating conditions for participation: conflicts and resources in systems development. *Human-Computer Interaction* 11, 215-36.

Bødker, S., Ehn, P., Kammersgaard, J., Kyng, M. and Sundblad, Y. (1987). A UTOPIAN experience: on design of powerful computer-based tools for skilled graphic workers. In G. Bjerknes, P. Ehn and M. Kyng (eds) *Computers and democracy: a Scandinavian challenge*, Aldershot, Avebury, 251-78.

Bødker, S., Ehn, P., Knudsen, J.L., Kyng, M. and Madsen, K.H. (1988). Computer support for cooperative design. *Proceedings of the CSCW'88 conference on computer supported cooperative work*, New York, ACM Press, 377-94.

Bødker, S. and Grønbaek, K. (1990). Cooperative prototyping: users and designers in mutual activity. *International Journal of Man-Machine Studies* 91 (34), 453-78.

Bødker, S. and Grønbaek, K. (1991). Design in action: from prototyping by demonstration to cooperative prototyping. In J. Greenbaum and M. Kyng (eds) *Design at work: cooperative design of computer systems*, Hillsdale, N.J., Lawrence Erlbaum, 197-218.

Bødker, S., Grønbaek, K. and Kyng, M. (1993). Cooperative design: techniques and experiences from the Scandinavian scene. In D. Schuler and A. Namioka (eds)

- Participatory design: principles and practices, Hillsdale, N.J., Lawrence Erlbaum, 157-75.
- Boehm, B.W. (1973). Software and its impact: a quantitative assessment. *Datamation* 19, 48-59.
- Bowers, J., Pycock, J. and O'Brien, J. (1996). Talk and embodiment in collaborative virtual environments. *Proceedings of CHI'96 Conference on Human Factors in Computing Systems, Conference Proceedings, New York, ACM Press, 58-65.*
- Braa, K. and Vigden, R. (1995). Action case: exploring the middle kingdom in information system research methods. *Proceedings of Third Decennial Conference Computers in Context: Joining Forces in Design, Aarhus, Denmark, August 1995, 50-59.*
- Breuker, J. (ed) (1990). EUROHELP: developing intelligent help systems. ESPRIT project P280, Final Report, University of Leeds.
- Brooke, J., Bevan, N., Brigham, F., Harker, S. and Youmans, D. (1990). Usability statements and standardisation - work in progress in ISO. In D. Diaper *et al* (eds) *Human-Computer Interaction - INTERACT'90*, Amsterdam, Elsevier North-Holland, 357-61.
- Brooks, P. (1994). Adding value to usability testing. In J. Nielsen and R.L. Mack (eds) *Usability inspection methods*, New York, John Wiley and Sons, 255-271.
- Brooks, R. (1990). The contribution of practitioner case studies to Human-Computer Interaction. *Interacting with Computers* 2 (1), 3-7.
- Bucciarelli, L.L. (1988). An ethnographic perspective on engineering design. *Design Studies* 9 (3), 159-68.
- Canning, R.G. (1956). *Electronic data processing for business and industry*, New York, John Wiley and Sons.
- Card, S.K., Moran, T.P. and Newell, A. (1983). *The psychology of Human-Computer Interaction*, Hillsdale, N.J., Lawrence Erlbaum.
- Carey, T.T. and Mason, R.E.A. (1983). Information system prototyping: techniques, tools, and methodologies. *INFOR - The Canadian Journal of Operational Research and Information Processing* 21 (3), 177-91.
- Carroll, J.M. (1996). Encountering others: reciprocal openings in participatory design and user-centred design. *Human Computer Interaction* 11, 285-90.

Carroll, J.M. and Campbell, R.L. (1989). Artifacts as psychological theories: the case of Human-Computer Interaction. *Behaviour and Information Technology* 8 (4), 247-55.

Carroll, J.M., Kellogg, W.A. and Rosson, M.B. (1991). The task-artifact cycle. In J.M. Carroll (ed) *Designing interaction: psychology at the human-computer interface*, Cambridge, Cambridge University Press, 74-102.

Carroll, J.M. and Rosson, M.B. (1992). Getting around the task-artifact cycle: how to make claims and design by scenario. *ACM Transactions on Information Systems* 10 (2), 181-212.

Checkland, P.B. (1981). *Systems thinking, systems practice*, Chichester, John Wiley and Sons.

Checkland, P. and Scholes, J. (1990). *Soft Systems Methodology in action*, Chichester, John Wiley and Sons.

Chin, G., Rosson, M.B. and Carroll, J.M. (1997). Participatory analysis: shared development of requirements from scenarios. *Proceedings of CHI'97 Conference on Human Factors in Computing Systems*, Conference Proceedings, New York, ACM Press, 162-69.

Christel, M.G. and Kang, K.C. (1992). Issues in requirements elicitation. Technical Report (CMU/SEI-92-TR-12), Pittsburgh, Carnegie Mellon University.

Clark, H.H. (1992). *Arenas of language use*, Chicago, University of Chicago Press.

Clark, H.H. (1996). *Using language*, Cambridge, Cambridge University Press.

Clark, H.H. and Brennan, S.E. (1991). Grounding in communication. In L.B. Resnick, J.M. Levine and S.D. Teasley (eds) *Perspectives on socially shared cognition*, Washington D.C., APA Books, 127-49.

Clark, H.H. and Carlson, T.B. (1982). Speech acts and hearers' beliefs. In N.V. Smith (ed) *Mutual knowledge*, London, Academic Press, 1-37.

Clark, H.H. and Marshall, C.R. (1981). Definite reference and mutual knowledge. In A.K. Joshi, B.L. Webber and I.A. Sag (eds) *Elements of discourse understanding*, Cambridge, Cambridge University Press, 10-63.

Clark, H.H. and Schaefer, E.F. (1987a). Collaborating on contributions to conversations. *Language and Cognitive Processes* 2 (1), 19-41.

Clark, H.H. and Schaefer, E.F. (1987b). Concealing one's meaning from overhearers. *Journal of Memory and Language* 26, 209-225.

- Clark, H.H. and Schaefer, E.F. (1989). Contributing to discourse. *Cognitive Science* 13, 259-94. Reprinted in [Clark, 1992].
- Clark, H.H. and Wilkes-Gibbs, D. (1986). Referring as a collaborative process. *Cognition* 22, 1-39. Reprinted in [Clark, 1992].
- Dagwell, R. and Weber, R. (1983). System designers' user models: a comparative study and methodological critique. *Communications of the ACM* 26 (11) 987-97.
- Diaper, D. (1989). *Task analysis for Human-Computer Interaction*, Chichester, Ellis Horwood.
- Dixon, J.R. (1987). On research methodology towards a scientific theory of engineering design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 1 (3), 145-57.
- Dowell, J. and Long, J.B. (1988). Towards a paradigm for Human Computer Interaction engineering. In E.D. Megaw (ed) *Contemporary ergonomics 1988: Proceedings of the Ergonomics Society's 1988 Annual Conference*, London, Taylor and Francis, 45-50.
- Dowell, J. and Long, J. (1989). Towards a conception for an engineering discipline of human factors. *Ergonomics* 32 (11) 1513-35.
- Downs, E., Clare, P. and Coe, I. (1988). *Structured Systems Analysis and Design Method: application and context*, New York, Prentice Hall.
- Ehn, P. (1989). *Work-oriented design of computer artifacts*, Hillsdale, N.J., Lawrence Erlbaum.
- Ehn, P. and Kyng, M. (1987). The collective resource approach to systems design. In G. Bjerknes, P. Ehn and M. Kyng (eds) *Computers and democracy: a Scandinavian challenge*, Aldershot, Avebury, 17-58.
- Ehn, P. and Kyng, M. (1991). Cardboard computers: mocking it up or hands-on the future. In J. Greenbaum and M. Kyng (eds) *Design at work: cooperative design of computer systems*, Hillsdale, N.J., Lawrence Erlbaum, 169-95.
- Ericsson, K.A. and Simon, H.A. (1993). *Protocol analysis - verbal reports as data*, Cambridge, MA, The MIT Press.
- Floyd, C. (1987). Outline of a paradigm change in software engineering. In G. Bjerknes, P. Ehn and M. Kyng (eds) *Computers and democracy: a Scandinavian challenge*, Aldershot, Avebury, 191-210.

Floyd, C., Mehl, W., Reisin, F., Schmidt, G. and Wolf, G. (1989). Out of Scandinavia: alternative approaches to software design and system development. *Human-Computer Interaction* 4, 253-350.

Fowler, C., Macaulay, L., Castell, A. and Hutt, A. (1989). An evaluation of the usability of a human factors based requirements capture methodology. In A. Sutcliffe and L. Macaulay (eds) *People and Computers V: Proceedings of the Fifth Conference of the British Computer Society Human-Computer Interaction Specialist Group*, Cambridge, Cambridge University Press, 359-71.

Friedman, A. (1989). *Computer systems development: history, organization and implementation*, Chichester, John Wiley and Sons.

Galliers, R. (1992). *Information systems research: issues, methods and practical guidelines*, Oxford, Blackwell Scientific.

Gärtner, J. and Wagner, I. (1996). Mapping actors and agendas: political frameworks of systems design and participation. *Human-Computer Interaction* 11, 187-214.

Goguen, J. and Linde, C. (1993). Techniques for requirements elicitation. *Proceedings of RE'93, IEEE International Symposium on Requirements Engineering*.

Gotel, O.C.Z. and Finkelstein, A.C.W. (1994). An analysis of the requirements traceability problem. *Proceedings of the first international conference on requirements engineering (ICRE'94)*, Colorado Springs, April 18-22 1994, IEEE Computer Society Press, 94-101.

Gould, J.D. and Lewis, C. (1985). Designing for usability: key principles and what designers think. *Communications of the ACM* 28 (3), 300-311.

Greenbaum, J. (1993a). A design of one's own: towards participatory design in the United States. In D. Schuler and A. Namioka (eds) *Participatory design: principles and practices*, Hillsdale, N.J., Lawrence Erlbaum, 27-37.

Greenbaum, J. (1993b). PD: a personal statement. *Communications of the ACM* 36 (4), 47-48.

Greenbaum, J. and Madsen, K.H. (1993). Small changes: starting a participatory design process by giving participants a voice. In D. Schuler and A. Namioka (eds) *Participatory design: principles and practices*, Hillsdale, N.J., Lawrence Erlbaum, 289-98.

Greenbaum, J. and Kyng, M. (1991). *Design at work: cooperative design of computer systems*, Hillsdale, N.J., Lawrence Erlbaum.

- Greenberger, M. (ed) (1962). *Computers and the world of the future*, Cambridge, MA, The MIT Press.
- Greenwald, A.G., Pratkanis, A.R., Leippe, M.R. and Baumgardner, M.H. (1986). Under what conditions does research obstruct theory progress? *Psychological Review* 93 (2), 216-29.
- Grønbaek, K., Grudin, J., Bødker, S. and Bannon, L. (1993). Achieving cooperative design: shifting from a product to a process focus. In D. Schuler and A. Namioka (eds) *Participatory design: principles and practices*, Hillsdale, N.J., Lawrence Erlbaum, 79-98.
- Grønbaek, K., Kyng, M. and Mogensen, P. (1993). CSCW challenges: cooperative design in engineering projects. *Communications of the ACM* 36 (4), 67-77.
- Grønbaek, K. and Mogensen, P. (1994). Specific cooperative analysis and design in general hypermedia development. *PDC'94: proceedings of the participatory design conference*, Chapel Hill, 27-8 October, 1994, CPSR Publications, 159-71.
- Grudin, J. (1990). The computer reaches out: the historical continuity of interface design. *Proceedings of CHI'90 Conference on Human Factors in Computing Systems*, New York, ACM Press, 261-68.
- Grudin, J. (1991a). Interactive systems: bridging the gaps between developers and users. *IEEE Computer* 24 (4), 59-69.
- Grudin, J. (1991b). Obstacles to user involvement in software product development, with implications for CSCW. *International Journal of Man-Machine Studies* 34, 435-52.
- Grudin, J. (1993). Obstacles to participatory design in large product development organizations. In D. Schuler and A. Namioka (eds) *Participatory design: principles and practices*, Hillsdale, N.J., Lawrence Erlbaum, 99-119.
- Harder, P. and Kock, C. (1976). *The theory of presupposition failure*, Copenhagen, Akademisk Forlag.
- Harman, G. (1977). Review of *J. Bennett (1976) Linguistic Behaviour*, Cambridge, Cambridge University Press. *Language* 53 (2), 417-24.
- Hawk, S.R. (1993). The effects of user involvement: some personality determinants. *International Journal of Man-Machine Studies* 38, 839-55.
- Heath, C. and Luff, P. (1991). Collaborative activity and technological design: task coordination in London Underground control rooms. In L. Bannon, M.

Robins and K. Schmidt (eds) Proceedings of the 2nd European conference on CSCW, Amsterdam, Kluwer Academic Publishers, 65-80.

Henderson, A. and Kyng, M. (1991). There's no place like home: continuing design in use. In J. Greenbaum and M. Kyng (eds) Design at work: cooperative design of computer systems, Hillsdale, N.J., Lawrence Erlbaum, 219-40.

Herbsleb, J.D. and Kuwana, E. (1993). Preserving knowledge in design projects: what designers need to know. Proceedings of INTERCHI'93, New York, ACM Press, 7-14.

Holtzblatt, K. and Beyer, H. (1993). Making customer-centred design work for teams. Communications of the ACM 36 (10), 93-103.

Holtzblatt, K. and Jones, S. (1993). Contextual inquiry: a participatory technique for system design. In D. Schuler and A. Namioka (eds) Participatory design: principles and practices, Hillsdale, N.J., Lawrence Erlbaum, 177-210.

Hornby, P. and Clegg, C. (1992). User participation in context: a case study in a UK bank. Behaviour and Information Technology 11 (5), 293-307.

Hutt, A.T.F., and Flower, F. (1990). The development of marketing to design: the incorporation of human factors into product specification and design. ICL Technical Journal, November 1990, 253-69.

Hutt, A.T.F., Donnelly, N., Macaulay, L., Fowler, C. and Twigger, D. (1987). Describing a product opportunity: a method of understanding the users' environment. In D. Diaper and R. Winder (eds) People and Computers III, Cambridge, Cambridge University Press, 61-74.

ISO 9241-11 (1998). Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: guidance on usability. ISO Technical Committee TC 159/SC4.

Jackson, M.A. (1983). System development, New York, Prentice-Hall.

Johnson, H. and Johnson, P. (1989). Integrating task analysis into system design: surveying designers' needs. Ergonomics 32 (11), 1451-67.

Johnson, H. and Johnson, P. (1990). Designers-identified requirements for tools to support task analyses. In D. Diaper *et al* (eds) Human-Computer Interaction - INTERACT'90, Amsterdam, Elsevier North-Holland, 259-64.

Johnson, H. and Johnson, P. (1991). Task knowledge structures: psychological basis and integration into system design. Acta Psychologica 78, 3-26.

- Johnson, P. (1992). *Human-Computer Interaction: psychology, task analysis and software engineering*, Maidenhead, McGraw Hill.
- Jordan, B. and Henderson, A. (1995). Interaction analysis: foundations and practice. *The Journal of the Learning Sciences* 4 (1), 39-103.
- Kaptelinin, V. (1996). Computer-mediated activity: functional organs in social and developmental contexts. In B.A. Nardi (ed) *Context and consciousness: Activity Theory and Human-Computer Interaction*, Cambridge, MA, The MIT Press, 45-68.
- Karat, C. (1994). A comparison of user interface evaluation methods. In J. Nielsen and R.L. Mack (eds) *Usability inspection methods*, New York, John Wiley and Sons, 203-33.
- Karat, C., Campbell, R. and Fiegel, T. (1992). Comparison of empirical testing and walkthrough methods in user interface evaluation. *Proceedings of CHI'92 Conference on Human Factors in Computing Systems*, New York, ACM Press, 397-404.
- Karat, J. (1988). Software evaluation methodologies. In M. Helander (ed) *Handbook of Human-Computer Interaction*, Amsterdam, Elsevier North-Holland, 891-903.
- Keil, M. and Carmel, E. (1995). Customer-development links in software development. *Communications of the ACM* 38 (5), 33-44.
- Kensing, F. and Munk-Madsen, A. (1993). PD: structure in the toolbox. *Communications of the ACM* 36 (4), 78-85.
- Kieras, D.E. and Polson, P.G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, 22, 365-94.
- Kjær, A. and Madsen, K.H. (1995). Participatory analysis of flexibility. *Communications of the ACM* 38 (5), 53-60.
- Kuhn, M. (1962). *The structure of scientific revolutions*, Chicago, University of Chicago Press.
- Kuutti, K. (1996). Activity Theory as a potential framework for Human-Computer Interaction research. In B.A. Nardi (ed) *Context and consciousness: Activity Theory and Human-Computer Interaction*, Cambridge, MA, The MIT Press, 17-44.
- Kyng, M. (1991). Designing for cooperation: cooperating in design. *Communications of the ACM* 34 (12), 65-73.

Kyng, M. (1995). Creating contexts for design. In J.M. Carroll (ed) Scenario-based design: envisioning work and technology in system development, New York, John Wiley and Sons, 85-108.

Laden, H.N. and Gildersleeve, T.R. (1963). System design for computer applications. New York. John Wiley and Sons.

Lewis, C., Polson, P., Wharton, C. and Rieman, J. (1990). Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. Proceedings of CHI'90 Conference on Human Factors in Computing Systems, New York, ACM Press, 235-42.

Lewis, D.K. (1969). Convention: a philosophical study, Cambridge, MA, Harvard University Press.

Lim, K.Y. and Long, J.B. (1994). MUSE: a structured human factors method for usability engineering, Cambridge, Cambridge University Press.

Long, J. and Dowell, J. (1989). Conceptions of the discipline of HCI: craft, applied science, and engineering. In A. Sutcliffe and L. Macauley (eds) People and Computers V: Proceedings of the Fifth Conference of the British Computer Society Human-Computer Interaction Specialist Group, Cambridge, Cambridge University Press, 9-31.

Luff, P., Jirotko, M., Heath, C. and Greatbach, D. (1993). Tasks and social interaction: the relevance of naturalistic analyses of conduct for requirements engineering. Proceedings of RE'93, IEEE International Symposium on Requirements Engineering, 4-6 January, San Diego, CA.

Madsen, K.H. and Aiken, P.H. (1993). Experiences using cooperative interactive storyboard prototyping. Communications of the ACM 36 (4), 57-77.

Mason, J. (1996). Qualitative researching, London, Sage.

Mayo, E. (1948). Hawthorne and the Western Electric Company: the social problems of an industrial civilisation, London, Routledge and Kegan-Paul.

Meister, D. (1987). A cognitive theory of design and requirements for a behavioural design aid. In W.B. Rouse and K. Boff (eds) System design: behavioural perspectives on design tools and organisations, Amsterdam, Elsevier North-Holland, 229-44.

Meister, D. and Farr, D.E. (1967). The utilization of human factors information by designers. Human Factors 9 (1), 71-87.

Minneman, S.L. (1991). The social construction of a technical reality: empirical studies of group engineering design practice, PhD thesis, Stanford University.

Published as Xerox Technical Report SSL-91-22, Palo Alto, Xerox Corporation Palo Alto Research Center.

Mogensen, P. and Trigg, R. (1992). Artifacts as triggers for participatory analysis. In Kuhn, S., Muller, M. and Meskill, J. (eds) Proceedings of the Participatory Design Conference, CPSR, Palo Alto, CA.

Monk, A. and Dix, A. (1987). Refining early design decisions with a black box model. In D. Diaper and R. Winder (eds) People and Computers III: Proceedings of HCI'87, Cambridge, Cambridge University Press, 147-58.

Monk, A., Nardi, B., Gilbert, N., Mantei, M. and McCarthy, J. (1993). Mixing oil and water? Ethnography versus experimental psychology in the study of computer-mediated communication. Proceedings of INTERCHI'93, New York, ACM Press, 3-14.

Monk, A., Wright, P., Haber, J. and Davenport, L. (1993). Improving your human-computer interface, New York, Prentice-Hall.

Mosier, J.N. and Smith, S.L. (1986). Application of guidelines for designing user interface software. Behaviour and Information Technology, 5 (1), 39-46.

Muller, M.J. (1991a). Participatory design in Britain and North America: responses to the "Scandinavian challenge". Proceedings of CHI'91 Conference on Human Factors in Computing Systems, New York, ACM Press, 225-31.

Muller, M.J. (1991b). PICTIVE - an exploration in participatory design. Proceedings of CHI'91 Conference on Human Factors in Computing Systems, New York, ACM Press, 225-31.

Muller, M.J. (1992). Retrospective on a year on participatory design using PICTIVE technique. Proceedings of CHI'92 Conference on Human Factors in Computing Systems, New York, ACM Press, 455-62.

Muller, M.J. (1993). PICTIVE: democratizing the dynamics of the design session. In D. Schuler and A. Namioka (eds) Participatory design: principles and practices, Hillsdale, N.J., Lawrence Erlbaum, 211-37.

Muller, M.J., Smith, J.G., Goldberg, H. and Shoher, J.Z. (1991). Privacy, anonymity and interpersonal competition issues identified during participatory design of project management groupware. SIGCHI Bulletin 23 (1), 82-7.

Muller, M.J., Tudor, L.G., Wildman, D.M., White, E.A., Root, R.W., Dayton, T., Carr, B., Diekmann, B. and Dykstra-Erickson, E. (1995). Bifocal tools for scenarios and representations in participatory activities with users. In J.M. Carroll (ed) Scenario based design: envisioning work and technology in system development, New York, John Wiley and Sons, 135-62.

- Muller, M.J., Wildman, D.M. and White, E. (1993). Taxonomy of PD practices: a brief practitioners guide. *Communications of the ACM* 36 (4), 26-28.
- Mumford, E. (1987). Sociotechnical systems design: evolving theory and practice. In G. Bjerknes, P. Ehn and M. Kyng (eds) *Computers and democracy: a Scandinavian challenge*, Aldershot, Avebury.
- Mumford, E. (1993). The participation of users in systems design: an account of the origin, evolution, and use of the ETHICS method. In D. Schuler and A. Namioka (eds) *Participatory design: principles and practices*, Hillsdale, N.J., Lawrence Erlbaum, 257-70.
- Mumford, E., Land, F. and Hawgood, J. (1978). A participative approach to the design of computer systems. *Impact of Science on Society* 28 (3), 235-51.
- Myers, B.A. and Rosson, M.B. (1992). Survey on user interface programming. *Proceedings of CHI'92 Conference on Human Factors in Computing Systems*, New York, ACM Press, 195-202.
- Nardi, B. (1996a) *Context and consciousness: Activity Theory and Human-Computer Interaction*, Cambridge, MA, The MIT Press.
- Nardi, B. (1996b). Studying context: a comparison of Activity Theory, situated action models, and distributed cognition. In B.A. Nardi (ed) *Context and consciousness: Activity Theory and Human-Computer Interaction*, Cambridge, MA, The MIT Press, 69-102.
- Naughton, J., (1984). *Complexity, management and change: applying a systems approach*, Block IV, Milton Keynes, The Open University Press.
- Nielsen, J. (1990). Paper versus computer implementations as mockup scenarios for heuristic evaluation. In D. Diaper *et al* (eds) *Human-Computer Interaction - INTERACT'90*, Amsterdam, Elsevier North-Holland, 315-20.
- Nielsen, J. (1994). Heuristic evaluation. In J. Nielsen and R.L. Mack (eds) *Usability inspection methods*, New York, John Wiley and Sons, 25-62.
- Nielsen, J. (1995). Scenarios in discount usability engineering. In J.M. Carroll (ed) *Scenario-based design: envisioning work and technology in system development*, New York, John Wiley and Sons, 59-84.
- Nielsen, J. and Phillips, V.L. (1993). Estimating the relative usability of two interfaces: heuristic, formal, and empirical methods compared. *Proceedings of INTERCHI'93*, New York, ACM Press, 214-21.
- Norman, D.A. and Draper, S.W. (1986). *User-centered system design*, Hillsdale, N.J., Lawrence Erlbaum.

- Noyes, J.M., Starr, A.F. and Frankish, C.R. (1996). User involvement in the early stages of development of an aircraft warning system. *Behaviour and Information Technology* 15 (2), 67-75.
- Nygaard, K. and Bergo, O.T. (1975). Trade unions: new users of research. *Personnel Review* 4, 2.
- Olson, G.M., Herbsleb, J.D. and Reuter, H.H. (1994). Characterizing the sequential structure of interactive behaviours through statistical and grammatical techniques. *Human-Computer Interaction* 9, 427-72.
- Olson, J.S. and Olson, G.M. (1990). The growth of cognitive modeling since GOMS. *Human Computer Interaction* 5, 221-65.
- Olson, G.M., Olson, J.S., Carter, M.R. and Storrøsten, M. (1992). Small group design meetings: an analysis of collaboration. *Human-Computer Interaction* 7, 347-74.
- O'Neill, E.J. (1995). CUSTARD: a participatory approach to usability requirements synthesis for software design. Doctoral Consortium, RE'95 Second IEEE international symposium on requirements engineering, York, March 27-9 1995.
- O'Neill, E.J. (1996). Task model support for cooperative analysis. Proceedings of CHI'96 Conference on Human Factors in Computing Systems, Conference Companion, New York, ACM Press, 259-60.
- O'Neill, E.J., Johnson, P. and Coulouris, G. (1995) CUSTARD: a participatory approach to task analysis, software requirements synthesis and design. Poster presentation, Computers in Context: Joining Forces in Design, Third Decennial Conference, Aarhus, Denmark.
- O'Neill, E.J., Johnson, P. and Johnson, H. (1997). Representations in cooperative software development: an initial framework. Proceedings of International Workshop on Representations in Interactive Software Development, Queen Mary and Westfield College, University of London.
- O'Neill, E.J., Johnson, P. and Johnson, H. (under review). Representations in cooperative analysis and design for system development. Submitted to *Human-Computer Interaction*.
- Oreström, B. (1983). *Turn-taking in English conversation*, Lund, Gleerup.
- Palmer, S.E. (1978). Fundamental aspects of cognitive representation. In E. Rosch and B.B. Lloyd (eds) *Cognition and categorization*, Hillsdale, N.J., Lawrence Erlbaum, 259-303.

- Polson, P.G., Lewis, C., Rieman, J. and Wharton, C. (1992). Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies* 36, 741-73.
- Reisner, P. (1981). Formal grammars and human factors design of an interactive graphics system. *IEEE Transactions on Software Engineering*, 5, 229-40.
- Rettig, M. (1994). Prototyping for tiny fingers. *Communications of the ACM* 37 (4), 21-7.
- Rosson, M.B., Maass, S. and Kellogg, W.A. (1988). The designer as user: building requirements for design tools from design practice. *Communications of the ACM* 31 (11), 1288-98.
- Royce, W. (1970). Managing the development of large software systems. In "IEEE WESCON", August 1970, 1-9 and reprinted in "Ninth IEEE international conference on software engineering", Washington D.C. Computer Society Press of the IEEE 1987, 328-38.
- Rudd, J., Stern, K. and Isensee, S. (1996). Low vs high-fidelity prototyping debate. *Interactions*, January 1996, 76-85.
- Sanderson, P.M. and Fisher, C. (1994). Exploratory sequential data analysis: foundations. *Human-Computer Interaction* 9, 251-317.
- Schegloff, E.A. and Sacks, H. (1973). Opening up closings. *Semiotica* 8, 289-327.
- Schegloff, E.A., Jefferson, G. and Sacks, H. (1977). The preference for self-correction in the organization of repair in conversation. *Language* 53 (2), 361-82.
- Schiffer, S.R. (1972). *Meaning*, Oxford, Oxford University Press.
- Schober, M.F. and Clark, H.H. (1989). Understanding by addressees and overhearers. *Cognitive Psychology* 21, 211-32.
- Schön, D.A. (1987). *Educating the reflective practitioner*, San Francisco, CA, Jossey-Bass.
- Shackel, B. (1990). Human factors and usability. In J. Preece and L. Keller (eds) *Human-Computer Interaction*, New York, Prentice-Hall, 27-41.
- Shackel, B. (1991). Usability-context, framework, definition, design and evaluation. In B. Shackel and S. Richardson (eds) *Human factors for information usability*, Cambridge, Cambridge University Press, 21-37.

Shneiderman, B. (1992). *Designing the user interface*, Reading, MA, Addison-Wesley Publishing.

Smith, N.V. (ed) (1982). *Mutual Knowledge*, London, Academic Press.

Smith, M.J. and O'Neill, E.J. (1996). *Beyond task analysis: exploiting task models in application implementation*. Proceedings of CHI'96 Human Factors in Computing, Conference Companion, New York, ACM Press, 263-64.

Smith, S.L. (1986). Standards versus guidelines for designing interface software. *Behaviour and Information Technology* 5 (1), 47-61.

Sperber, D. and Wilson, D. (1982). Mutual knowledge and relevance in theories of comprehension. In N.V. Smith (ed) *Mutual knowledge*, London, Academic Press, 61-87.

Sperber, D. and Wilson, D. (1987). *Précis of relevance: communication and cognition*. *Behavioural and Brain Sciences* 10, 697-754.

Spivey, J.M. (1992). *The Z notation: a reference manual*, New York, Prentice-Hall.

Spradley, J.P. (1980). *Participant observation*, New York, Holt, Rinehart and Winston.

Staggers, N. and Norcio, A.F. (1993). Mental models: concepts for Human-Computer Interaction research. *International Journal of Man-Machine Studies* 38, 587-605.

Stalnaker, R.C. (1978). Assertions. In P. Cole (ed) *Syntax and Semantics 9: pragmatics*, New York, Academic Press, 315-32.

Suchman, L.A. and Trigg, R.H. (1991). Understanding practice: video as a medium for reflection and design. In J. Greenbaum and M. Kyng (eds) *Design at work: cooperative design of computer systems*, Hillsdale, N.J., Lawrence Erlbaum, 65-89.

Swartout, W. and Balzer, R. (1982). On the inevitable intertwining of specification and implementation. *Communications of the ACM* 25 (7), 438-40.

Sweeney, M., Maguire, M. and Shackel, B. (1993). Evaluating user-computer interaction: a framework. *International Journal of Man-Machine Studies* 38, 689-711.

Tang, J.C. (1989). *Listing, drawing and gesturing in design: a study of the use of shared workspaces by design teams*. PhD thesis, Stanford University. Published

as Xerox Technical Report SSL-89-3, Palo Alto, Xerox Corporation Palo Alto Research Center.

Tang, J.C. (1991). Findings from observational studies of collaborative work. *International Journal of Man-Machine Studies* 34, 143-60.

Terrins-Rudge, D. and Jørgensen, A.H. (1993). Supporting the designers: reaching the users. In P.F. Byerley, P.J. Barnard and J. May (eds) *Computers, communication and usability: design issues, research and methods for integrated services*, Amsterdam, Elsevier North-Holland, 87-98.

Tetzlaff, L. and Mack, R.L. (1991). Discussion: perspectives on methodology in HCI research and practice. In J.M. Carroll (ed) *Designing interaction: psychology at the human-computer interface*, Cambridge, Cambridge University Press, 286-314.

Thoresen, K. (1993). Principles in practice: two cases of situated participatory design. In D. Schuler and A. Namioka (eds) *Participatory design: principles and practices*, Hillsdale, N.J., Lawrence Erlbaum, 271-87.

Tudor, L.G., Muller, M.J., Dayton, T. and Root, R.W. (1993). A participatory design technique for high-level task analysis, critique and redesign: the CARD method. *Proceedings of the Human Factors and Ergonomics Society 1993 Meeting*, Seattle, WA, October 1993.

Virzi, R.A., Sokolov, J.L. and Karis, D. (1996) Usability problem identification using both low and high fidelity prototypes. *Proceedings of CHI'96 Conference on Human Factors in Computing Systems*, Conference Proceedings, New York, ACM Press, 236-43.

Walz, D.B., Elam, J.J. and Curtis, B. (1993). Inside a software design team: knowledge acquisition, sharing and integration. *Communications of the ACM* 36 (10), 63-77.

Whitefield, A. (1990). Human-Computer Interaction models and their roles in the design of interactive systems. In P. Falzon (ed) *Cognitive ergonomics: understanding, learning and designing Human-Computer Interaction*, London, Academic Press, 7-25.

Winograd, T. and Flores, F. (1987). *Understanding computers and cognition: a new foundation for design*, Norwood, N.J., Ablex Publishing.

Wong, E.Y.W. and Tate, G. (1994). A study of user participation in information systems development. *Journal of Information Technology* 9, 51-60.

Wright, P. and Monk, A.F. (1989). Evaluation for design. In A.Sutcliffe and L. Macaulay (eds) *People in Computers V: Proceedings of the Fifth Conference of the*

British Computer Society Human-Computer Interaction Specialist Group,
Cambridge, Cambridge University Press, 345-58.

Wright, P. and Monk, A.F. (1991). A cost-effective evaluation method for use by
designers. *International Journal of Man-Machine Studies* 35, 891-912.

Appendix 1

The following is a partial and not particularly accurate transcription from the audio track of video recording CI1 (see chapter three). It is provided here *for illustrative purposes only*. It provides a flavour of the interaction which went on in a CI project task modelling session. It is not a basis for conducting an interaction analysis. For that, as discussed in the thesis, the analyst requires considerably more than even a 'perfect' transcript could provide.

The session starts with the user, U, and two developers, D1 and D2, sitting on chairs in the space between a long desk and an office wall. A first cut at TM1 is represented on a chart on the wall.

D1: You know, if you're going to genuinely reduce the manpower involved in these things, I think there has to be some change somewhere along the line and it'd be nice to start identifying where that might arise.

U: Are we saying that although we start off with the structure as laid out in the uh, HADS and MIRAS

D1: MIRSAP.

U: MIRSAP, we are now more happy than before to go straight off those, if we see benefits.

D1: No. I mean, with provisos. My view is that you should be able to sit a current HOLMES user in front of the new system, and he or she should feel perfectly familiar with it, and the new system should support the current working practices.

U: Right.

D1: Having said that, you would like to recognise and build in opportunities for change

U: Yeah.

D1: such that things can shift between one role and the other.

U: Yeah.

D1: Now roles can combine, so it is going to be a bit of juggling in terms of the new design to ensure that. What you don't want is to build, build it rigidly to the current working practices such there is no opportunity to change, or the opposite is build it so radically new that it will be difficult to transfer to because there will be a big barrier in using it.

U: Yeah.

D1: It is striking this balance between the two, isn't it? The first thing is to make sure that we understand what the current system's about.

U: Yeah.

D1: What we've done is produce this chart, and the people icons represent the different roles.

U: Right.

D1: So from the left there you've got: an enquiry officer who seemingly does very little, the telephonist, the exhibits officer, the receiver that one is and going across to the action indexer and so on. The red and blue lines represent the movement of objects around the system.

U: Let's just cover the lines again. The red and blue, right? What's the difference between red and blue then?

D1: The red ones are inputs to a particular process

U: Right.

D1: and the blue ones are outputs and we see that there's some kind of loops.

U: The secret is to get four lines around the object without

D1: We've abstracted things out of the business. You'd be talking about paper copiers and photocopiers and god knows what else, so we've abstracted that out, and assumed that it somehow gets dealt with, and focused on the essence of the process rather than sort of the nitty gritty detail. I mean for example it is an interesting question this business about electronic versus paper copies, right? This is a topic that we've covered and how that gets handled. You could go to one extreme of making everything electronic so everything was viewed on the screen. In many respects it would be quite easy to implement that, but I dare say you would have quite a lot of resistance in terms of working practices. So there's an example where you want the system to support either, so you can print them off or you can read them on the screen.

U: Right.

D1: I thought that the easiest way to start, to get familiar with the model and also to check that the overall structure is right, is to follow a few objects through it

U: Right.

D1: seeing what would happen in terms of

U: Well, the most important thing to follow would be a statement followed by an action.

D1: Do that then?

D2: Yeah.

U: Right.

D2: Uh where do we start?

U: Well, initially, this guy would probably get

U: Well the most important thing to follow would be a statement followed by an action.

D2: Do that then?

D1: Yeah.

U: Right.

D1: Uh where do we start?

U: Well, initially, before the incident room was probably set up, this guy would probably get a statement.

D1: Yeah.

U: So you can actually start - here, OK?

D1: Yeah.

U: Where he receives a hand-written statement and the first one he ever gets he ever gets will probably be without an action. Normally it would be accompanied by an action.

D1: In that sense, we've actually neglected that bootstrapping. We're working from the start from actions.

U: OK.

D2: Did we not deliberately ignore that?

D1: What we've got is that the enquiry officer delivers a return action, the return action includes some documents.

U: Statement, PDF.

D1: This goes to the receiver, and he does what he has to do with it, and you find that you get a completed action with the documents. So it transforms from a returned action into a completed action. And that'll be marked up with urgent actions and so on. It then goes to this guy, the Indexer stroke Action Writer, who marks it up, and you'll see that he's got a resulted action. And then it gets separated in terms of you get a registered statement manuscript and report manuscript.

U: So what we're saying here then that the action, the complete action and documents, that includes statements?

D1: Yes.

U: Right.

D1: We haven't separated them yet, the bundles are all at one end. But what you are saying is that when you start the incident up, some things will come in without an associated action

U: Some There will be some statements or documents taken before that probably is running.

D2: Is that something we need to model separately?

U: No, no. They handle that quite well now.

D2: In terms of we've got here, do we need explicitly to have the bootstrapping process somewhere here?

U: What do you mean by bootstrapping?

D1: It's getting yourself started really. To get the systems going, you need sort of

U: No, cos I mean, all you do is speed the documents to this guy, he gets them registered and numbered. OK?

D2: Hmmm.

U: These, the indexer

D2: That's the indexer there.

U: The indexer indexes the statements.

D1: Yeah.

U: Right, there's no actions to cross reference to them. There's no numbers on these documents. You start indexing at this stage.

D1: Yeah.

U: But as actions are raised that cover those statements

D1: They retrospectively

U: their research will cover those actions.

D1: Do they retrospectively update those actions?

U: Yeah, well yeah because they made statements. Right? At some stage somebody could say raise an action to take a statement from one of these people who made a statement before actions were being raised, OK?

D1: Mmm.

D2: Mmm.

U: D'you follow that?

D1: Yeah.

D2: Yeah.

U: But, on the document they are raising this action from, they will discover that a statement has already been taken, their, now look, research should discover that a statement has already been taken, so they are merely cross-referencing it.

D2: But they would never actually fill in an action form for that first original statement retrospectively?

U: They would have two choices. Yes, yes, they can do. They can either do an action which is so easy and cross reference it to the statement, or, in the statement they are now indexing, where they would normally raise this action to take the further statement but the statement's already been taken, they can just put the document cross-reference in. So there are actually two ways an indexer could tackle that. He can stop himself raising an action, and just put the document reference into the portion of the statement that it refers to, or they can raise an action and just cross-reference that action.

D1: I've put

U: And that action would then go straight to be completed, filed, everything, finished.

D1: I've put a little note here about early statements because it's something that persists

U: Early documents. It could be anything. Some people have a sweet set up, where everything starts up as it would in a classroom situation probably. But in here you could have a lot of telephone messages, a hundred maybe, before you actually get your set up running.

D1: If, for example, you had a system whereby you could actually generate an action which corresponds to getting the statement as a kind of way of getting you going, would that be useful?

U: Just go over that again.

D1: So you get a statement coming into the system without an early action, especially when you are starting off the incident. If there was a facility whereby you said let's make an action, automatically, a special facility.

U: No. No need. As long as that goes through the indexing process and the stuff out of those statements is indexed and any reference to those people or objects in those statements should be picked up later on through research and cross-referenced accordingly. Although we've mentioned these, it's not a problem to them as such, but there are quite a sizeable chunk of documents can appear.

D2: And throughout the enquiry, presumably, it's not just at the start? Presumably at any time something might turn up without an action.

U: Yes.

D1: You have incoming other documents, and you can have incoming telexes and stuff like that.

U: Messages come in all the time without action.

D2: And they are just put through in the same way?

U: Yes. Yeah. There's two routes for messages, but yeah.

D1: Tell you what, let's go back to just following the statements. The normal way that a statement comes in is as the result of an enquiry, and attached to a returned action. This goes to the receiver

U: Yes.

D1: He chooses the next document and then he does whatever he has to do to receive it.

U: Right. It goes to the receiver. If there is any property or exhibit brought in with the action, it would go there, give him the exhibit, get an exhibit reference number on his action form, and then go here and here.

D1: So are you saying that actually the returned action and the exhibit goes through that path?

U: Yes.

D1: What we've got is an exhibit going there, separate ways from the action, and you're saying that the two are kept together.

U: This officer will take his bundle of stuff and go here, because if you look on the action form it asks for an exhibit reference number, and this is the guy that can give him the exhibit reference number. And he should also give him a statement of recovering that exhibit.

D1: Right. So what we've got here is that the

U: Now, exhibits are a bit of a dodgy area. We've started hitting some snags already. Some of the exhibits will be stored in the incident room, so they merely go there, they're referenced by him in his exhibits book.

D1: Mmm.

U: OK? Put a number on it and then come back here and go through this system. Now, some of them will be documents that can be happily kept in the incident room. But if they are forensic items, they will stay there and he will look after the forensics. The easiest way to put it is that he will look after the forensics stuff, but any other stuff like document exhibits will go through this channel. It must go there first for numbering, and he should, in practice he should give him a statement of recovery, because again if. Now you won't find this in MIRSAP you see. It's the praxis thing that's important.

D1: A what thing?

U: It's the practical application that's important. You might not find it in your MIRSAP or whatever you call it. If and I'll tell you why. This guy is going to end up with thousands of exhibits, if not tens of thousands, depending on the job. Yeah? At the end of the enquiry, somebody has to find out and get all these statements to find continuity. Even if it's only two hundred exhibits out of these thousands, he's got to get two hundred statements, possibly. It's also difficult for the officer because he has got to backtrack all through the database and his pocket-books to say, well, I found the cigarette packet at nine o'clock at so and so location, and I found the matches at so and so location ten minutes later, y'know. Now, it's all done and dusted at this stage. It's all right for the guy that has done it, but there's no chasing. It saves an awful lot of time and resources.

D1: If there's not an exhibit, it goes through that route. If there is an exhibit, then the exhibit and the returned action

U: Yeah.

D1: goes through that route

U: Yeah.

D1: with an exhibit number

U: Yeah, the exhibits book number.

D1: is generated there, a recovery statement made.

U: The exhibit book number here, *his* exhibit book number, which is a manual book

D1: Yeah.

U: could be different to anything that is registered over here.

D1: Yeah, OK.

U: OK.

D1: OK. So that number needs to be incorporated and recorded into the system.

D2: If the exhibit is a document, is it the document that gets passed on or a copy of it?

U: Right, if it was for finger-printing or any forensic data then it would have to stay here, and if you can take a copy of it without contaminating it or anything else, it can carry on and flow through.

D2: But the original would never be sent through?

U: The original document would go through, yes. A statement can be an exhibit. A list of vehicles in a street could be an exhibit, but that does not need to stay with this guy.

D1: But when it comes to here, you've got this choice of either going to the exhibits officer or not, and you say that even a statement can be an exhibit, how do you know whether it's going to be an exhibit or not?

U: You don't. You don't know. You don't know how anything's going to be used in an incident. You set off, on most occasions, with a body and a scene. All right? And basically you saturate both to get intelligence about the body, because that's all you've got. You gather intelligence about the two things that you've got: you've got a scene and a body. Now what you take out of that the deceased's antecedent history might not be relevant at all, but it might be very relevant when you associate who has done it. But if you've gathered a load of antecedent history of associates and it turns out to be a complete stranger that has done it, then all that stuff you took was irrelevant, no use, but you don't know that at the time. If you knew at the time then the investigation would stop here because you would go straight to the person that has done it.

D1: Right.

U: Yeah.

D1: But, I mean, not everything that the enquiry officer brings in will go to the exhibits officer.

U: No, that's right.

D1: So.

U: This guy, the enquiry officer won't bring in that many exhibits because this chap, there could be two of these guys. There could be scenes of crimes guys. There will be a number of officers, maybe two or three officers who will bring in the bulk of exhibits.

D1: Yeah.

U: There might be a search team, and they will find exhibits or likely exhibits. The exhibits officer is going to come across exhibits or likely exhibits. And if they use different scenes of crimes officers to exhibits officers, they are going to find exhibits or likely exhibits. It's becoming more and more important that a qualified scenes of crimes officer is the evidence gatherer, the exhibits gatherer at the scene, because the protection of that stuff is highly important.

D1: So let's just finish off this then. It goes through this route.

U: Right.

D1: The action and all the documents and exhibits go here.

U: Yeah.

D1: The officer will fill out the statement of recovery

U: Yeah.

D1: an exhibit number will be written on the various things

U: Yeah.

D1: Right. And then the whole bundle comes back through this route

U: Yeah.

D1: as if it came in ordinarily

U: including the newly written statement

D1: including this statement of recovery. OK?

U: Yeah.

D1: OK? So then we've exhausted that bit.

U: Yes. Quite happy with that.

D1: The return action with all the statements and reports and whatever then goes through here. Most of these tasks are organising, he selects a document to be processed. Then he does the processing and then he forwards it.

U: Right.

D1: So we can come to what will happen later, right?

U: Right.

D1: So he has got a returned action with

U: Documents.

D1: documents. One thing we should check, actually, is that we've got the right kind of term, and we should try and be careful about the names of these things

U: Returned action

D1: Returned action, then you've got a completed action

U: Yeah, he completes it.

D1: and then here it goes through the resulted action.

U: Yeah. Yes. Of course, in here, indexer writer, indexer action writer, in here, that's where the registration's embodied as well.

D1: Yeah.

U: Yeah.

D1: Yeah, we've got different variants for that.

U: Yeah, OK, right.

D1: So, one of the questions I always feel unclear about is that, uh, one of the things that this guy does is checks that a, uh

U: Action has been completed correctly.

D1: Yeah. It's down here so you receive a returned action. Right, you assess the returned action. You read it and the associated documents. You perform basic checks.

U: Right.

D1: Is the action endorsed properly? Are all the documents returned that went out with it?

U: Yeah.

D1: You might debrief the officer, so

U: Well, you might and you might not. It's not essential.

D1: Question mark against that. And then you say, is the returned action completed satisfactorily?

U: Assess returned action. Perform basic check. OK. Yeah, it's also reading to see if there's anything urgent in the manuscript documents that've been returned that needs doing urgently.

D1: Yeah. Yeah. We've got that one.

D1: Further actions

U: He might accept further actions, the receiver. Only if they are very urgent.

D1: can distinguish between If the returned action is completed satisfactorily, there is a no branch and a yes branch. If it's a no then you endorse the action log with no. And the reason why is because you say the action has not been completed satisfactorily. I was a bit confused about what really happens here. It gets forwarded through the system, the non-completed action, and I believe the action.

U: No. If he says the action is not complete, this guy, it should go back to him.

D1: Directly?

U: Yes. That's the door, he should not allow it in. Takes a statement from Joe Bloggs¹, and then he goes and takes one from Eamonn, go back and take it

¹ Where names of the actual participants were used, they have been changed - except for the author's.

from Joe Bloggs. Or give me a good reason that is acceptable to put through the system.

D1: I've changed it and the reason we got confused is there seems to be two circumstances. There are some uncompleted actions, for example the guy is on holiday.

U: That goes to the action allocator.

D1: I got the two muddled. Here I've said that he has accepted the uncompleted action, and then an uncompleted action comes through the system, so he says I've accepted this uncompleted action. It comes through here to the action allocator, and there is a thing here about what you do with uncompleted actions, accept it, or you return it to the enquiry team, or you make it pending and you get the approval for the connection.

U: If the guy went on holiday, he does not need approval, he would just

D2: For referrals but not for pending?

U: Not for pending, no. It has got to be pending for a reason that you can't do it. The SIO can't make the guy come back from holiday, he can't make him come out of his hospital bed.

D1: So that is only for referrals seeking out the SIO's approval.

U: Yes. Take a statement from a guy that turns out to be on holiday, obviously falls into the parameters of those actions that need doing, completing.

D1: I suppose the opposite is true. If the guy needs the statement, the action then is to go off to the Bahamas and get the statement, and that would need approval.

U: If it was urgent enough to be done, and you can't wait for him to come back from holiday, then you should need approval before you go to the Bahamas. And also check to see if anybody else wants seeing while you are there. Are you happy with that?

D2: Yes.

D1: So the things we've missed off, then, it could appear that there are two types of uncompleted action. One goes straight back to the officer

U: An action is an instruction to do something. If it has not been done and there is no good reason why it has not been done, it should go back to the officer. He is virtually the first man in the quality control.

D2: If there is a good reason why it has not been done then it gets passed to the action allocator?

U: If there is a good reason it has not been done, at this moment in time (i.e. holiday, sickness), there will be an assessment written on there by the officer saying this guy is in hospital, we are expected out in forty minutes' time. They can go to the action allocator for a pending, a bringup date on the action, and the system will bring it up on the and reallocate it to the original officer.

D2: Is the bringup date appended automatically?

U: It is tied to. But the system will bring it up automatically like a batch program

D2: And it's always done? They don't just stick it in without a bringup date?

U: It would be unwise to stick it in without a bringup date. You can always look at the full list of pended actions anyway, just by sticking a p in and getting all of the pendings up. If you put a bringup date in, you can forget about it. You just know it will arrive when wanted.

D1: What I've got is rejected on the complete action, but coming back to the action allocator, so really this ought to come back from the receiver, shouldn't it?

U: Yes.

D1: Like that.

U: This guy should not be sending any actions back. He is purely an allocator and manages.

D1: That is good. You've made the diagram look tidy. These huge loop-back things are rather hard to represent.

U: Is that the Victoria line or

D2: So the other big loop will have to stay, so?

D1: So that link there is actually replaced by that one, from the receiver?

U: Yes. It would be better if I agreed to it all, and then I would not have to go through all this again.

D1: is pretty good, but it's not quite widely/precisely what you want.

U: It could be a little better.

D1: OK. So what we've got then is: we've got the rejected action, and we've got a route for uncompleted actions to go through to get pended, if necessary, and brought up at a later date. When it has been pended, when it's brought up to get reallocated so it will come back to an enquiry officer as an allocated action

U: It should go back to the same guy.

U: We are back to the allocation plan. You should have a team/teams dealing with anything to do with the victim, a team/teams dealing with anything to do with the scene, a team/teams to do the suspect, etc. They become experts in their own domain.

D2: The action allocator will keep a record of who pended it and give it straight back to them?

U: That is right.

D1: So when you pend it here, you actually keep the original allocation, and then it gets .

U: Yes. And it is all date-time stamped automatically. Although all these guys all become, or should become, experts in their own domain, they have briefings probably every day, perhaps twice a day, so that they are not isolated in their domains

D1: Actually the briefings was one of the things I wanted to ask you about. Whose responsibility is it to do the briefings ?

U: The responsibility has got to come to the SIO. The overall briefings for the team on the enquiry fall on the SIO. The office manager can have an incident-room briefing without the SIO, he can just brief his team when he needs to. An officer in charge of the outside teams can brief just the outside teams. It may be that the SIO wants to address these individual teams on certain things in isolation from other teams. There is no need for everybody on the whole enquiry to be present at a briefing to the house to house team, nine times out of ten. You may find an exception, but I would find it unusual to have everybody standing there just to listen to a house to house team being briefed. They are going to go down a street, knock on every door and get the details. Why have detectives, who are specialising in other areas sitting listening to them. They might want to listen to the outcome of that house to house enquiry, but not to the actual start-up.

D1: So what have we done so far then? We've got we've got to the point where actions uh statements are rejected or actions are rejected. And we've now got a completed action and documents.

U: Yeah. Yeah.

D1: So these would get registered stroke indexed depending on which document you're talking about.

U: Yeah.

D1: It's fairly noticeable that certain things get processed instantly, but certain things get processed only when there has been time: they get registered the basic details get registered straight away.

U: Yeah.

D1: Um so what d'you get? You get a completed we were following the course of a statement through weren't we?

U: Yeah.

D1: So that's attached to a completed action.

U: Yeah.

D1: Um it comes through here, you select it for indexing, um there's you register, index and action a document, right, and there's different variants that you do something for a message, you do something for a telex

U: Yeah. Yeah.

D1: you do something for another document, you do something for a completed action

U: Yeah

D1: and you do things for standalone documents that come back through the system.

U: Yeah.

D1: Mm?

U: Yeah.

D1: Uh. He records nominals and vehicles for CRO and PNC checks <pause> and a list is maintained somewhere. <long pause>

U: I'm not sure what you're getting at there.

D1: For every every nominal that comes through the system

U: Has it been PNC checked?

D1: Yeah.

U: No. The system will generate a list of people, every twenty-four hours, with details for a PNC check.

D1: Yeah.

U: First name, first names, last name, date of birth, sex.

J: Sorry to interrupt but sandwiches.

D1: D'you want some sandwiches.

U: Yeah. Do you?

D1: I brought some. We've both brought some.

U: Can we stop for two minutes till I get a sandwich.

D1: I think we'll let you.

<laughter>

U: I'll just finish that off then. Where are we?

D2: PNC.

D1: PNC.

U: PNC. But, and this a fall-down of the system, it only picks up the people who had documents registered to them. OK? So if these people register a statement from me, and I mention you in my statement, because the statement is registered to me I will appear on the next day's list for a PNC check.

D1: Mm mm.

U: But because you've got no documents to you, registered to you, you'll just purely be linked to my document, you will not appear on that list.

D1: So what is the real, underlying requirement, that you check everybody?

U: Everybody.

D1: OK so what we have here then is that you record nominals and vehicles as well, is it? Every every every nominal and vehicle that gets

U: Well all nominals are PNC checked. Vehicles if you want vehicles doing that's a different task.

D1: OK. So let's deal with them one at a time.

U: Right.

D1: Nominals.

U: Nominals.

D1: Every nominal that's registered on the system

U: That is No, that is registered to a document. <pause>

D2: Authors only?

U: Authors only, exactly. Authors only.

D1: Well. Sorry. I thought a moment ago you said that you

U: Registered. No you you're talking about registered on the system. Your registered on the system is just putting your name and address on the nominal index.

D1: I'm asking a question. Right? What what would you what is really required? What what would you, what is really required? Just those people who've had documents registered to them

U: Everybody.

D1: Or everybody.

U: Everybody.

D1: So

U: Every new entry on the system should be PNCed.

D1: So that every nominal that you enter in the system

U: At the moment

D1: is PNCed.

U: Yeah, at the moment those people don't, who are not authors of documents, right, are not getting picked up

D2: So

U: or are being difficult to pick up.

D1: Mm.

U: Now eventually most do end up having some kind form of document registered to them.

D1: Mm.

U: but not not all.

D2: Would it be better if you could pick up everybody at that stage?

U: Yes. Yeah. Because let's be fair if I mention Joe Bloggs in my statement, I get registered here. Right? And we'll say it's a it's a uh a very type, the MO is really unusual, duh-duh duh-duh, y'know, a one off. Joe Bloggs doesn't, has

a conviction for this type of offence, the exact MO, y'know everything every minute detail, and we don't pick it up for five days or a week possibly, because he's got to go right through this system actioned right round here. Now if you pick it up here, you could save a lot of time. It would certainly it would certainly alter the type of action that would be raised.

D1: Yeah.

U: It might make say it might make them raise a high priority action, saying Hey let's get this guy seen and eliminated, or put into the or y'know out of the frame very quickly.

D1: Mm. So. There's a change there isn't there? Uh. All nominals

D2: Mm.

D1: not just nominals registered to documents.

D2: And vehicles.

D1: And vehicles. We we need to ask him about that.

D2: Mm. We're getting there.

D1: It's going all right then. He seems to be following the diagram quite readily. The strength of just doing a simple decomposition, is that you don't have all sorts of fancy arrows and controls and things like that which is

D2: confusing. I think that he is taking it for granted that we've got it right.

D1: off doing is following the flow of documents to check that that is right, and various useful things are coming out of it. That is what we should do is actually . Perhaps what we should do is go through the model from left to right and check it, because otherwise we are going to miss things. Do it role by role. When necessary we can follow the flow of documents through to check it is right, change tactics. What I want to check is that as objects that we've mentioned follow round, so that we can then decide that objects have got to be supported.

D2: Nothing new has come up yet, has it?

D1: Details, you might say that they are attributes, like bringup dates, keep the allocation on action. His opinion is that actions are pending. You might as well store the action ready-made, with a bringup date associated with it, then the same action can be injected into the system. the whole area

D2: It covers the whole thing.

D1: In fact we might be able to draw it up here and show things going through. It's quite a difficulty with the charts actually. It's very difficult to show the flows. These big feedback loops I had to rearrange in order to fit it on.

D1: What you could do with this is a multi-part link. I took the comments at heart

D1: What we are actually doing is ignoring the detail, and it's some of the detail that we would like to check. If you don't mind, we will change tack a bit and just go through role by role, working out all of the detail. Where necessary we will

follow through with the object flow just to check that we've got it right. So, shall we just start from left to right and move across?

U: Yes.

D1: The joke I made earlier, that the enquiry officer does not do very much

U: Shall we leave him to last. The enquiry officer is normally a detective. They normally work in twos, and they deal with actions.

D1: In terms of the model here, really

U: We are not too concerned with him really. We are more concerned to administer the enquiry for him.

D1: I am merely explaining why there is not much detail there.

U: How he carries out the tasks that this gives him, is really down to his expertise.

D1: The task here is to perform an enquiry, which generates a return action, and we discussed this earlier. And he gets allocated an action.

U: These guys try to and he can get messages himself. He can people to say things take a statement from. He can put that in as a message. He might have information himself he wants to put on as a message.

D1: He is supposed to record outgoing calls

U: He should record all outgoing calls. He might want to call another force and say can you tell me about Joe Bloggs?, can you quickly establish if a vehicle is in a certain location?, any kind of thing. He could want to make a phone call regarding an action that he is doing.

D2: Wasn't the clearest the message passed that she gives him

U: There are two areas here. It's a little bit confusing here the procedure regarding the telephonist. There are two ways that can be done. She receives a telephone call, and she writes it manually onto a message pad. That message pad can then go directly to the receiver, who can mark up the manuscript one for indexing. It would then get put on the system and registered, and at registration here it would get everything done to it. Or

D1: Does it ever get typed, by a typist?

U: Yes, at this stage.

D1: The indexer types it?

U: They are normally only two or three lines long. The other method is that straight from her it gets registered and typed, then to him, to the receiver, to be marked up and actions raised.

D1: Oh. We have a completely different one here, because we assume that the normal route would be for the telephonist to type it up.

U: The telephonist will not have time to type anything up, depending on the size of the incident room and the amount of phone calls coming in. If it's a small incident room, the telephonist will be running doing clerking and telephones.

In a very busy room you could have a bank of telephonists, they could be receiving thousands upon thousands of calls.

D2: They do have the facility to type it on the system, but not very soon.

U: Well, if they type it onto the system before it goes to the receiver, the system will give it the next number. If it's typed onto the system before he gets it, and I don't care who by, it has a number on it already, he marks it up and it goes and gets indexed. If the manuscript goes straight from there to there, it gets everything done at once here.

D1: So, if it's a message manuscript

U: Well every message will be written down in manuscript. I don't know of any incident rooms whereby she is typing it straight in.

D1: You get the message/manuscript, and there are two routes to take. One it can go in there as manuscript

U: and marked up

D2: To the receiver?

U: Yes. Marked up and everything.

D1: Two: it goes through there and then back that way and it gets typed there.

U: The second one it will get typed up, whether it's there or somebody here. On this method here it gets registered, typed up, or gets a number, and then it goes back to here for reading and action-raising, if necessary.

D2: Back to get registered again?

U: It's registered already because of the number, but it will get indexed and actions raised.

D1: When you say registered, is this the same registering ?

U: Registering it is normally giving it a number.

D1: There is also giving it the nominal phone and address. In indexes, that is usually part of the registering, but is that

U: But with a message it's slightly different. If you go right into the pre If you type it then it purely gives that message a number. I am not sure what it actually does with the

D2: We've taken, as a working definition, we've distinguished between the allocator and the number and the complete registration. So we've defined registration to be that its existence is based upon the system, you've allocated it a number, plus it's cross-referenced to the standard thing it should be cross-referenced to. That is what we deem to be registration.

U: Registration is whereby a document is registered to an individual and given a unique number. Registered to the author and given a unique number. That is why the message is a bit different. If I run the procedure right: if the manuscript document goes here this is what is confusing me, you could have variants of themes on how the thing works here.

D2: Can we distinguish between the index and action writer and the person

U: The title index and action writer was given on the manual system, because he used to index and write actions, but he does not actually write actions anymore. Indexing is giving cross-references for ever. What I said before on this model here, there is a little part of that should just do registration. You can if you want let everybody register within here, and everybody index, but it's far better to have one dedicated person doing registration because he has to be very good at research. Otherwise we will end up with duplicates and all sorts. He has got to be a really thorough researcher.

D2: Is the only thing that you are including in registration the number and the nominal?

U: The nominal address and probably telephone number. But you would not go as far as vehicle and stuff like that.

D1: In a sense, the way that the message is handled, the chap who is doing the indexing action, right or wrong, for the purposes of argument would become the typist.

U: It could do. The message situation is a little bit messy, by having these two procedures. I will keep it very simple. Manuscript can be marked up in manuscript form by a receiver, and then registered, typed, indexed, the whole lot, at one go here by the guy who is doing the registration. Then it's gone, it goes on its way. If it goes from the person receiving the message, the telephonist, to being typed first, typing it on the system will give it a unique number. But then it has got to go to the receiver to get marked up. Now I am not sure when it gets a unique number whether it's registered to a person at that stage or not. I would have to check the system for that.

D2: Does passing something straight from the telephonist to indexing not break the subroute?

U: It ends up going to the receiver, all you are doing is either you give him a manuscript one to play with or you give him a typed one to play with.

D2: So the typing function does not actually have to be done by the indexer?

U: No. It could be done by anybody. You could say Let this person take a hundred messages if you want, then give them to a typist.

D1: So it's a matter of practicality really?

U: Yes. But would you be happy leaving a bundle of a hundred messages until somebody looks at them. It really does depend upon your size and how busy your room is. This is an area that we can possibly look at: how slicker the system can be made.

U: i.e. taking the messages in and what he has done to them, and how they are done afterwards.

D1: A lot of these credit companies, that you ring up and enquire about your credit card, are directly on line and they record notes as you are talking to them. They have headphones and they have got dual application.

U: Well that would be ideal for this position.

D2: How big a part in the enquiry is the message index?

U: It totally rests upon the nature of the incident, and how the public view that incident. If you've got an incident involving young children especially, whether it's rape or murder, you will get masses. It also depends upon the type of appeals or media appeals that the incident room makes, and how they play it. Local radio now is very powerful, never mind national radio, television, newspapers. How we handle this could also become relevant to a major disaster, because instead of going to thousands of messages, you could be going to tens and hundreds of thousands of messages.

D1: We need to address this staffing issue. If the typing of these things can be made nice and slick, then we can actually make the process more

D1: So let us assume that we got a system whereby the telephonist records it all. She records the message and the ID gets allocated. The ID always gets allocated as the system.

U: If they are typing it in it will, yes.

D1: In the meantime we've talked about prioritising messages. Certain messages will get tagged as this really needs attention.

U: I don't know how, does it tell you?

D1: No. It just says that it will. I had visions of trotting across the room saying hey, excuse me.

U: It's good to say that you have prioritised messages, but how you prioritise a message is again on parameters given by the guy in charge of the enquiry. On days 1, 2, 3 and 4, you are so new on your enquiry, how do you know what is going to be relevant and what is not? If the enquiry is regarding Alderley Edge, and people are ringing in about stuff that is definitely divorced from this enquiry, you can say well that can just be weighted and you shove the other stuff through. But that is about as much as, it's a dodgy area. That is why they say prioritise, but they don't give any guidelines on how to do it. And this person has their own, personal priority.

D1: We do have our own thing elsewhere that says we prioritise.

U: I don't think that you can prioritise messages at this stage without some clear instructions from somebody, saying Any messages to do with anything else but ABCD go in that pile, the rest go through the route.

D2: You need a parameter.

U: Yes you do. If it was a major disaster area, that is much easier isn't it? Was he on the plane? May he have been on the plane? Or, should he have caught a different plane? So he was definitely on the plane that crashed, so he goes on that pile, may have been goes in that pile. That is much easier to prioritise. That is more definitive. It does need simplifying a bit, this two methods is just I say the best method is manuscripts, straight to him a natural flow.

D1: This business about filing things. We talked about making copies of files and stuff

U: It tells you that. One will stay with these, and the other will either go to the office manager or the SIO.

D1: Why do messages stay with these? Why do a copy of the messages stay there and a copy of the messages stay over there?

U: I think the answer to that is, because here we have a card system which is from the computer, and the card system, if this guy wanted to look up what message twelve said, he could go to look it up.

D1: Because he needs to read it. But if it's on the system

U: If it's on the system. I think the best answer is, Joe, seriously, that is one of the hic-coughs of going through a card system. He can now deal with the message on the screen, he does not need a copy.

D1: Also he will be able to view it in a manner which does not interfere with what he is currently doing.

U: That is right. Because in this section here, there will only be one copy of messages, so when you want it, I might also want it. There could be a dozen or more people wanting access to that message book at any one time.

D2: Go on to more general things now

D1: When you come to statements and things like that, we had this discussion about whether the SIO sees the unmarked-up statements or the marked-up statements.

U: On a computerised system he should be able to see both.

D1: When he asks for it.

U: The actual system directs that when a statement which messages from it, a statement is typed as four copies. One of those copies goes to the Senior Incident Officer, SIO, which is purely typed and unmarked. Now that creates a problem, in that he reads that and says Well, I want him seen and I want that done, etc. He then comes into this place and says I want actions raising about this, probably before they have got to that. He has probably got his typed copy before they have got to their typed copy to start indexing. And he really starts disrupting their operation. If you could have a very slick, quick and up-to-date here, why give him an unmarked copy? Give him a marked copy straight away. He can look at it on the screen.

U: Yes. You trigger thoughts all the time, but messages certainly are purely left there because that is the only access that they would have to the document, and the lab copy, statements copy, PDFs.

D1: If we were to summarise our general approach to, you want to get away from except for having a physical copy of it somewhere, for backup purposes.

U: You have to file original statements. You will have to file a personal description form. You will have to file house to house enquiry forms. You will have to have a file copy of the message somewhere. But you only need one copy.

D1: And the rest of the system can sort of

U: Not only will it save resources, because you have a guy going round putting all of these things in different docket - he is a busy fellow - and if he puts it in the wrong docket, it can create all sorts of problems. So you will not only save resources, but you will save time. Time is resources, I suppose.

D1: OK. Let us move on a bit then. Another task that was mentioned in the was that the telephonist was supposed to maintain a list of useful numbers. And it describes this as being different from the list of telephone numbers put on the telephone index.

U: What they are saying is that those are useful numbers that might be needed by people on this enquiry. Why store in two lists? It's absolutely stupid. So this guy say I want to ring the hospital, because that is where the victim ended up and the post mortem was, he has to ask her, unless the telephone number is on here he has to go and ask her or him.

D1: You imagine that certain numbers like an ordinary number, is not really to do with the incident, except in terms of the police operation. Whereas, say, the call-back number of the person who has phoned in the message is.

U: I think that no way should they have two lists. Anything that they think is relevant or may be helpful to anybody in the enquiry, not just that person, can be called up from one of these things. If I'm an enquiry officer, and I come in here and I say that I want to call back my person, then he will say that he has not got the number. I then have to go and ask that person. Why can I not just ask one person? What do they mean by useful telephone numbers, do they mean the Water Board, etc?

D1: Well, I don't know. It could be an enquiry officer says I am at this garage now, you can contact me here for the next hour or so.

U: Why should that not be in there? Again, another issue we have to think of is that if they keep useful numbers, should that be part of the enquiry because of disclosure? I am not sure. No way should there be two lots of numbers. I don't see any reason why useful numbers here can't be stored here. Useful telephone numbers: waste disposal, ambulance, whatever they are, Joe Bloggs' number if anybody wants to get hold of him.

D2: Where have these numbers come from?

U: Exactly. It's one of these MIRSAP things, they will say we would like you to prioritise it. They mention the words prioritise message but they don't give you any guidelines how or what circumstances you might want to do it. We've been doing that from the card system days. Useful must mean useful to the enquiry.

D2: Are these only numbers that have been phoned in to the or are they numbers that might have come from a statement or something?

U: What you've got to read in is that when they say useful numbers it's what this person decides are useful.

D2: Would this person be

D1: Does the telephonist make outside calls, when the SIO says get me so-and-so will he do it for him?

U: Can do but basically he's there to receive messages.

D1: I see it as a practical thing, that there are numbers such as the mortuary number, the doctor who

U: But the mortuary will be on here somewhere in the indexes, so why not have the number kept with the mortuary?

D1: So these go in, injected into the system through messages.

U: Yes.

D1: Shall we move on again then? We've talked about that and I think we've covered that, although later we need to expand upon that. We go on to this thing, and follow through the different kinds of things. This is a receiver, and somehow he basically selects the next document to process out of his basket or through the electronic equipment. He does the processing of it and then he hauls it through the system. And then what you find is that, depending on the kind of thing that you've got, you get different variants of what he does. So this is a variant for dealing with a message with a telex, which are handled very similarly.

U: Those are the documents that he reads and marks up.

D1: It's quite simple really that when he is receiving a message from a telex, he will read it, and author actions, urgent actions and non-urgent actions - we are distinguishing between those two, but in other places though they have not been - and he marks up for indexes. So in other words you do the totality of the system, you mark it up and leave it. When you look at other documents, it's a similar kind of thing

U: Which includes statements?

D1: No. This is just other documents, the statements end in actions. So other documents you are saying that he reads it, he assesses it for urgent actions

U: Well, another document will have come in with an action, will it not?

D1: No.

U: Well how could it be retrieved. Well it could come in with that one couldn't it?

D1: Well, we've got a situation where an incoming other document could appear out of the blue.

U: OK. We'll accept that.

D1: In which case there are documents we will come to documents that appear with actions in a minute.

U: So he has got the other document and he reads it

D1: He reads it and then he assesses it for urgent actions, attached in the document form, endorse it with the title, and endorse it with urgent actions.

U: The document form is A4 wide, but are only about that deep, A5 size. And on there is a part for him to give it a title and to say briefly what it's about, and then he can raise actions if necessary under the comments. The urgent actions part is for really reading the actions, reading the statements, etc.

D1: So are you saying that he actually marks up all documents for all actions?

U: Yes. At his level, yes.

D2: What about the statement reader?

U: He gets it later.

D2: And marks it up?

U: What happens is that that other document, with its other document slip fastened to it, will then go and get registered after he has done his bit, and the actions raised that he is suggesting. The document then goes in its own file, and the statement reader, office manager and SIO will then go to that file every day/other day - depends how many there are - go through it, and again there is a space for them to make comments, and if they wish to add additional actions.

D2: The word urgent there we used simply to indicate that they are between what the receiver thinks is urgent

U: No, I would cross urgent out there because it might be some time before anybody else looks at it. It wants to be left very similar to how he does a message.

D1: So he raises all actions that he thinks are necessary from that document?

U: Yes.

D1: Which will not necessarily be all the actions that will ever get raised on that document. Just what he thinks should be. You've got a received other document which goes to the statement thing, so that is right. He still gets it.

U: He still gets it but it's if it actually goes in his basket he will go to the file and pull the documents and then look through it.

D1: Let us go through the time when he gets an action, and all the bundle of documents with it.

U: Which could include another document.

D1: So we need to make sure that is catered for.

U: The first thing is that he reads the return action, which has been an instruction to an officer to carry out a task. He then makes sure that task has been completed. If the task has not been completed then the action should not be completed.

D1: So we've got a question raised: is the returned action completed satisfactorily?

U: If yes

D1: He can do that lot and if no he can do that lot. But beforehand we did some basic thing that you read it and the associate documents. The documents have been returned, and then we will question that

U: In some cases possibly, but not in the majority of cases. They just come flying through. That would be a very specialised action.

D1: Right so that is the kind of preliminary check. And now I am saying Is it satisfactorily completed? And if it's not then you refer it for further enquiry, you endorse the fact that it has been referred

U: He can't refer an action. He does nothing, he just gives it back to the officer. He can write on the action this has not been carried out in accordance with the instructions.

D1: I am using referred not in the sense

D2: For referral?

D1: Yes.

U: So referred has got to come out of there. He can return it for the action to be carried out correctly.

D1: He endorses the action log with the fact that it has been returned to the officer

U: Only on the action.

D2: He writes it on the action form?

U: If he wishes he can write on the action form, but he does not have to. He has got the action and all the accompanying documentation with it. He has read it and said I am not happy, this action has not been completed, go back and complete it. He would either see the officer personally, and say Well, I am not happy with that. You've not covered everything that wanted covering. Go back, or he can write on it and give it back. I would write on it if I was him.

D1: On the system, he records it on the action log

U: The only way he can do anything with it is to complete it. Yes, for a while you would get a list of actions for completion.

D1: So that is endorse

U: You see if it's not completed, it's still with the officer. Really what he is saying is I am not accepting that in this room. It's not in any state completed. It's still with you.

D2: So we've actually got three things there. It's either nothing is completed satisfactorily, in which case it's sent back out

D1: So we are sending it straight back to the enquiry officer, or there is a good reason that he has accepted it and it goes through as an uncompleted action and gets pended by the action officer.

U: There is a facility to do a partially resulted action. There should be no such thing as a partially resulted action. I think we should stay clear of that, I make mention of it, full stop.

D2: It's mentioned in the thing.

- D1:** Does that mean that the action has been raised in the first place, for greater granularity? For example, take a statement from everybody in this room. Really you should have individual actions for individual statements.
- U:** No. It's to allow the receiver, as I read it, to put through an action as complete that really is not complete. And there he can mark it as partially resulted. If it's the nitty-gritty areas whether it is complete or not I am prepared to put complete but I want it marked as partially resulted. So everybody else down the line, from indexers to office managers to the SIO, see that I put it through as partially resulted, i.e. if you are happy with this send it off. Actions should be yes or no.
- D1:** That goes back to the kind of action that you raised in the first place. If you raise a very broad, brush action, then the chances are that
- U:** Parts of the instructions I give will not raise actions. You would not say Interview everybody at the bus stop, you would say Obtain a list of persons at the bus stop, and you would interview each person individually. The rule is one action per person or object.
- D2:** How do you decide when he comes back to you with this whether or not it has been partially resulted?
- U:** If you say Obtain a list of people at the bus stop and he comes back with a list of people at the bus stop then it's complete.
- D2:** Even if there is only four on the list and there were other people there?
- U:** If he says there was ten at the bus stop, but I've only traced four, then he can say these are the only four I've traced and there are six unidentified people. It may be that the four he has identified may be able to identify the other six, so you could raise actions to have those four interviewed, and they would say things like I don't know his name, etc. so it might be quicker to send somebody to the same bus stop on the same time and day, and obtain names that way.
- D1:** Basically you have to leave it up to the people, in order to judge how they do things.
- U:** If that is what they say pending, then what is that on your left back to the same officer yes, that is right.
- D1:** So if it's completed satisfactorily, you result the action and that involves marking up for further action.
- U:** Who is doing that?
- D1:** This is the receiver's job, for action or urgent action, take the documents and mark them up for urgent action. He marks up actions for either urgent or non-urgent actions, and he marks up the associated documents attached on them.
- D2:** Any action? Urgent or non-urgent? Would he completely mark it up?
- D1:** What happens then is that the whole bundle gets forwarded along. So as a result of this, you either get a completed action.

U: Normally, when he were at first here, if he has got an action with a statement, PDF and another document, and it's carried out correctly, he will just write : Register statement, PDF and other document and then initial it.

U: If there is anything really staring him in the face, he also raises action to clarify

D1: So what we really mean is to correct. The net result of that is that you either get a completed action going through the system, or you get an uncompleted action for pending. Or an action that goes directly back to the enquiry officer that has been rejected. What I've got here is a rejected, uncompleted action coming from here rather than the action allocator. You remember our earlier discussion, I thought it was the action allocator that did it.

D2: The thing is now that there are two types that are going to go back, so maybe you do still need that big blue line, because the pending ones are still going to go back.

D1: No. The pending ones will come back as an allocated action mainly.

D2: Through the normal?

U: Through the normal. It just reallocates.

D1: Is there anything about the receiver that we've forgotten? The output from the receiver is the received message, the received telex, the received other document, the completed action and the document, the uncompleted actions

D1: The reason I am doing it like that is that the action allocator - he is the guy who

U: Yes he is.

D2: But that is uncompleted or it's confusing there. Because the uncompleted one is the one that gets sent back out again.

D1: No. I've labelled that as a rejected uncomplete action.

U: His task is to complete correctly carried-out actions. This is words we are playing on here isn't it?

D1: What we want to do is use the terms that are accepted.

D1: Is that for pending?

U: I would say that these were for-pending actions.

D1: Or for-pending, incomplete actions.

U: For pending. (laughter)

D1: So what would we call this one then?

U: It's an uncomplete action.

D1: Simply an uncomplete action?

U: Yes. And he is not having it, so it's returned to the officer.

D1: What we are saying here is that when you come on to the index of action , you have obviously got those things that you sent from the receiver to there. But

you also get marked up statements and other documents coming back from the statement area.

U: You are right. But this bundle that he has had will go to the guy who does the registration within this section. Somebody in this section will result the action, register the statement and other documents, if there are any

D1: It's interesting, because in the MIRSAP procedures, the role as the registration guy is only as an index and action writer.

U: It would be very foolish, practically, to have six different people doing the registration and indexing.

D1: So what you suggest is that we can actually separate this role out into two - one for registration and one for indexing and action writing.

D2: Why is that done? What is the advantage?

U: Research. You can't have duplicates. Initially, when incident rooms started, they thought that registration could be done by anybody, because you have very good people, average people and below average. It was thought that it could be carried out by people who were below average, and let the real indexing be done by the above average. It's probably the reverse, in actual fact, because if you don't have somebody who is really well trained and a very good researcher registering you will have duplicates, you will have people registered to wrong documents, it's horrendous. It needs to be a really good, solid operator registering. You might have to have two doing the registration, but for most jobs one in registration is quite enough. He will result actions and register them. I would certainly have a little registration section within the indexers. Now again, we've inherited what you've got there - indexer and action writing - from the card system. In the card system it was not so bad because they sat round a carousel with all of the cards in etc. Registration was a much different thing there - you could not do as much research and stuff like that.

D2: What has changed then from the guy sitting round the carousel to now?

U: The depth of research and the numbers of objects you get in the system. Let's be straight now. There was only one enquiry that I can think of that went into tens of thousands of objects, and that was the child-killings from Coldstream near Scotland right round the country. It was undetected for ten or fifteen years, and they have put all of those different card system murders into one, you eventually end up putting it on. Other odd ones like that, they never ever get enquiries on a card system. It was kept small was beautiful, and it was good because, once HOLMES was introduced the same philosophy stayed of keeping it small, but once they realised that you can bang everything into the system and it would hold it. And if you saw a graph of incident sizing, of how much objects went into an incident room, it would go like that with cards, and as soon as HOLMES comes it will sort of go up a little like that and then, once they saw the confidence in it, it went like that. Unbelievable. People now are putting everything in as a safety net. They are not making decisions anymore because they feel safer just shoving everything in there, which is wrong.

D2: When you say registration, do you mean.?

U: Registering the documents, giving unique ID's, registering it to the name, address, telephone.

D2: And that is the complete process?

U: That is as far as the registration goes.

D2: One guy is doing all of that?

U: At least one guy.

D1: So, the basic thing is that anything you bring into the system has got to be cross-referenced to the things it should be cross-referenced to.

U: He's not cross-referenced to this bit, apart from the document to people, or registering it.

D1: There is no point in registering a document merely to allocate its number so you know of its existence. That is not much use to man nor beast. What you are saying is that the reason you are register it and link it to the nominal person who thought of it, is because then you've got that connectivity, so that when you touch something else you can see that connection and that might lead you to go and research it a bit further. So it's important in registration to cross-reference it to that degree otherwise there is no point. Otherwise you have to be looking through the list of documents to say

U: I don't really follow what you are saying there. Can you go through that again?

D1: If, for example, all you did was allocate the number - so the system knows this document exists - for that to be useful, a researcher would have to look through the documents each day and register in his mind what is new and what they are about.

U: ... statement already. He must do a lot of research to make sure who he is registering these documents to. To people or objects, to get a unique number for that document, and if necessary he will have their address and telephone number, and cross-reference them to it.

D1: After something has been registered, it's sufficiently

U: at this time they don't exist.

D1: But if you said that he entered the number in, it would be no use to anybody because you would just have big lists of numbers, and the title of the document. What good will it give you?

U: That is all you will see. When he registers it, all you will see is that Joe Bloggs has made statement 10.

D1: Yes. But other indexers and other researchers will not notice that unless it's linked to some nominal.

U: Which is him, registrationally. But that is the only nominal that it will be linked to initially. And he will have resulted the action. So the resulted action will go

to the action allocator. So he will then put it in a file of resulted to get the SIO to file it.

U: The other one is, so that has got rid of the action - the action allocator will put it in a pile for filing. The statement that came in with that action will now go to the typist, and she will type four copies.

D2: What about those copies?

D1: Let's continue moving through this. The next role that we've talked about. We've talked about separating the two.

U: Let's cut the registration part out.

D1: Well let's just go through to make sure that we are doing it right, because I think that the variants that we've got do actually separate out.

D1: If we go to the top level thing we've already talked about, you select the next document, you register, index and action the document. About recording nominals for the. We said that the real requirement is that we do that with everybody as soon as possible, not. We did not actually finish off talking about the vehicles, because it says in the, also PNC checks on vehicles. Earlier you were about to explain that.

U: If you've got a PNC check on a person, it will tell you whether its got a history or not. But if you do a PNC check on a vehicle. There will always be an interest in people in every enquiry, there will not always be an interest in vehicles. I would say You can do a PNC check on the vehicles if necessary.

D1: How do you check this?

U: You can check the procedures, but there is nothing in to say that you can do a batch program of all new vehicles.

D2: Is that something the SIO might instruct?

U: The SIO can instruct, if he wishes, Do not index vehicles. Vehicles do not enter this enquiry at all.

D2: At the moment all the PSEs are batch, twenty-four hour.

U: If you are doing a vehicle PNC, you are going to have to keep doing a list every day. How do you know which are the new ones?

D2: Can you not provide the number, when it's registered?

U: A vehicle has got its own ID, its registration number. The same facility should be extended to vehicles that you are going to extend to people.

D2: The ID number?

U: A facility to, say, give me a list every twenty-four hours of all new entries on the database. In fact they could go one better than that. Why do you want to do a PNC check on all new nominals when you are only interested in men? You might only be interested in men of a certain colour. So why PNC check a white guy or a white female when you are looking for a West Indian, six foot two male?

Appendix 2

This appendix presents a transcript of the sample of video recorded interaction from the CS1 meeting which was used in the analyses of chapters five and six.

D1: Reception. [*writes Reception on TM*] Fault line. [*writes Fault line on TM*] And Help Desk. [*writes Help Desk on TM*] Does your average customer know the distinction between these?

U: Ah, it depends whether they've been around for a while or not, no.

D1: Yeah.

U: Somebody who'd just sort of called the switchboard and said 'Put me through to computing service enquiries' would get reception [*points to TM*] because that's the sort of unspecialised one. I mean fault line [*points to TM*] is specifically for hardware problems. We've sort of discussed that.

D1: Right.

U: Help Desk [*points to TM*] is for technical software type queries. So if you knew which you wanted then you'd call one of those two. [*points to TM*] And that's [*points to TM*] the sort of all purpose one for people who don't know what they want.

D1: Right. Uh, is it first, I mean, this, we have someone here [*draws Reception role on TM*] at the Reception desk

U: Yes.

[*D1 writes on TM*]

D1: waiting. And also someone at the help desk?

U: Yes.

D1: Now. [*writes on TM*] Presumably if an enquiry comes in to either of these [*starts pointing to TM*] it goes to the person who's manning it. Yes? [*stops pointing to TM*]

U: Yes. You mean in person, somebody who walks through the door?

D1: However, however a query is received. What happens with the fault line?

U: Ah, this is where it gets a bit more confusing. Ah, normally [*points to TM*] reception and the fault line, although they're two separate telephones are actually dealt with by the same individual.

D1: Ah. Cloth. [*gestures at Reception on TM*] Ok, what's happened to the cloth?

U: Yes. I mean, that isn't necessarily the case.

[*D2 erases reception role from TM*]

D1: Right.

U: I mean, the theory is that we have the counter there

D1: Yeah.

U: and there is supposed to be

D1: Yeah.

[D2 draws reception role on TM]

U: normally two people sitting behind that counter. Yeah?

D1: Yeah.

U: But either one of them would answer either telephone.

D1: Right. So. So, but, but these lines both go to the same desk.

U: Well yes, you've seen it. It's kind of a long counter just facing the front door as you come in and there are two telephones on it.

D1: Aaah, and if someone calls here [points to TM] with something that your staff think should go here [points to Help Desk on TM]

U: Mm-mm.

D1: What happens?

U: Transfer the call.

D1: Just. Right. So, you have either a phone line directly [draws line from customer to help desk role]

U: Mm.

D1: a phone line to here [draws line from customer to reception role]

U: Mm.

D1: a phone line to here [draws line from customer to fault line role]

or you have transfers - [draws line from fault line to help desk role] that way?

U: Aah, more usually from here [points to reception role] to one of these two. [points to fault line and help desk roles]

D1: Right.

D1: Where'd the cloth go?

[U laughs. D2 hands cloth to D1]

D1: The alternative is to use our hands and it ends up very messy very quickly. [D1 erases line between fault line and help desk]

D1: Right. So, can you show us where [hands marker to U]

U: Yeah, OK. Tell me if I'm using the notation though

D1: Yeah OK.

U: in a way that makes sense to you or not.

D1: Fine.

U: But either of these calls, the person calls the wrong way, the wrong one first would get transferred to the other [*draws arrowed lines both ways between fault line and help desk*] or calls could get transferred from here to there [*draws line from reception to fault line*] or from here to there [*draws line from reception to help desk*] depending on which one the receptionist thinks

D1: Right.

U: is appropriate.

D1: Right. So

U: And of course it's always, also, perfectly possible for the [*points to TM*] receptionist to transfer a call directly to one of the [*pause*] not front line technical people.

D1: Right. So that was the other role we had. [*writes Technical on TM*] Technical. I mean, technical person, technical staff.

U: Aah. It seems to be um a sufficient word. Ah. We don't have front line technical staff, so. Except the help desk of course.

D1: Yeah.

U: So, um, just calling someone technical seems to be sufficient to know that they're second line.

D1: Yeah.

U: Ah, second line. That's short enough.

D1: Right. [*writes Second line above Technical on TM*] Second line. And reception refer calls to them? [*draws line from reception to Second line*]

U: Yeah. I mean, you know, if reception recognises that a problem that a customer has is specific to something that they know a particular individual is the expert in

D1: Right.

U: They'd refer it on to that person.

D1: Yeah.

U: Who wouldn't necessarily be a technical person.

D1: Is that right?

U: If it was an administrative problem they might get referred on to the person who was responsible for that bit of administration.

D1: Yeah.

U: You know, receptionist, ah, knows everything, um, transfers whatever it might be to whoever it might be.

D1: Right. And who else have we got involved [*consults list of roles*]? Aah, technical staff, ..., the manager. The manager role is ... Uh, Well, the extra bit

that they want is to be able to produce management reports, so that's practically outside

U: What we're talking about.

D1: Yeah, what was going on here. So perhaps leave that for the moment.

U: Yeah. They're sort of parasitic outside everything else.

D1: Absolutely. Uh, clerical, administers, administrative staff. Do they play a role in this business? Or are they sort of outside it too?

U: Uh. Well, um, in general the theory is that if a customer wants something done, what they want done will either be an administrative thing or a technical thing. But certainly there's, there is some area of overlap between those two fields. Um, I, I, I, I, I'm not clear to what extent it would be valuable to sort of build fuzzy edges into what we're drawing here.

D1: Right. What would be an example of an administrative enquiry from a customer?

U: Uh, registering the user as, uh, someone with a logon ID on a particular computer.

D1: Right. Who would that go to?

U: Uh, liaison.

D1: Right. Who does, uh, that's the administrative people?

U: Yeah.

D1: That's actually separate from clerical. Clerical would be like Collette who are like departmental

U: Aah. That's true, yes. I mean, there's administrative who are administering sort of what customers do

D1: Yeah. [*writes on list of roles*]

U: and there are also the internal administration

D1: Right.

U: clerical people.

D1: Right. So,

U: And they are relatively separate, yeah.

D1: Right, so we've got an administrative person somewhere. [*gestures to TM*]
Where, where, where would they be?

U: Uh, you mean the people who are administrating what the users do?

D1: Yeah.

U: Yeah, well, they actually sit, physically the people are located right by reception.

D1: Right. Are they the same people?

U: Uh, yeah, again this is the sort of fuzziness. Um. It's kind of

distinguishable that there are the two roles.

D1: Right.

U: Uh, there's the people who are sometimes referred to as the Liaison group uh who are sort of administrators and users and there's the people who are reception people who sort of answer telephones and are there as the, when you come in through the door and say 'Hello, where am I? What's my name?'

D1: Yeah. Front line people.

U: Yeah, yeah, very front front-line people.

D1: Yeah.

U: Uh, but in practice that's actually a group of four people who all sit together and there is an awful lot of interchange between what they do. Essentially I think that's mostly just because um there has to be someone there nine to five um so that there's always someone to answer the telephone or to be there when someone comes through the front door and with two reception staff uh and people being sick and going on holiday and so on in practice it's necessary for all four of those people to work together to just make sure that uh front front-line role is covered.

D1: Uh

U: Um, actually sorry that's made me think. In a sense you could say that there are kind of three lines because if you want to regard it that way you could say that the fault line and help desk are second line because

D1: Right.

U: They are the people yuh, in general, anyone who comes in who knows exactly what they want can go straight to them but somebody who doesn't know what they want would go first to reception then get referred to one of these two and one of these might then refer them on to a more expert expert.

D1: Right.

U: So you could say this is third line, these two are second line and that's the first of all the lines.

D1: Right. So. From there might be a reference back [*draws line from help desk to Second line Technical on TM*]

U: Oh absolutely. Yeah, sure.

D1: And from here

U: Yeah. Mm-mm.

[*D1 draws line from Fault line to Second line Technical on TM*]

D1: And also we we we've this administrative [*D1 writes Admin above Reception on TM*] thing for user logons and stuff.

U: Yeah. Mm-mm.

D1: Is, for our purposes does it seem useful to have this as a separate role or is it combined.

U: Yes. Well, no, it's not really, I think, y'know, [*points to TM*] this is quite clearly a logically distinct role

D1: Yeah.

U: I mean it was actually physically located elsewhere until our offices were split up and then essentially it was just moved over to these offices because uh it was just, it needs to be uh where the users actually could come in to to talk to people. Um. So I think there's a clear sort of logical distinction between these two roles but as I was saying, I'm sure logically you could make a distinction between these two roles

D1: Mm.

U: but in practice as I say the four of them sit together

D1: Mm.

U: and they work together so closely that in fact it's fairly

D1: Yeah.

U: interchangeable who does which of those roles out of the four of them.

D1: Yeah. Again it goes back to you to your question of how to differentiate between a role and a task.

U: Yeah.

D1: At what level does one become another and for our purposes, y'know, we can, sort of, create a new role calling it whatever and having included within it those three primary tasks

U: Mmm.

D1: Um, where, where that's important for us is that if we're going to produce a system to support these kind of things one person sitting at a screen might want to be able to pull up all three of these.

U: Ummmm.

D1: If, if their remit spans them. Otherwise, they may only need to see one.

U: Umm, yesssss.

D1: So we can do sort of clarify how separate or not that these things are

U: Yes, OK well. In terms of what you'd want to have on your screen at any one time I would have thought that uh anyone even among this group of four people that sort of are in this area would either the actual task they were carrying out at any one time would either be a user administration task or it would be a reception task. So they'd want one screen for one thing and one screen for the other one.

D1: Yeah. Right.

Appendix 3

This appendix presents a transcript of the sample of video recorded interaction from the CS5 meeting which was used in the analyses of chapters five and six.

D1: For, for us here, what we're trying to do is really just break down the parts which are which we're going to directly support

U: Mm.

D1: in the software so is that something that you would just do while talking to the customer? Does the use of the form based system need to support that or is that something that is really done pretty much mentally or ... or whatever?

U: Phhh. Yeah. Well, I would have said the latter. I don't see at the moment unless other people suggest it to you that there's any reason to [pause]

D1: No

U: try and set up a form that as it were prompts you to go through a script of now ask the customer this question.

D1: Yeah. Right. Uhh, let's ... there's no point in breaking down stuff into detail that isn't relevant

U: No.

D1: so one and two are OK, just basically talking to the customer. Five recording the query and three referring the query. Ahh. Well five recording the query what's that going to involve at this stage? And now at this point we're not going to make decisions. Uh. We shouldn't once we've got some time on it if you're going to do it while I'm gone but it'll involve Trevor. But just now we'll do what we can with it.

U: Right. I thought Trevor had already given you some suggestions for what he

D1: Yeah, he has. Y'know, what, if we do it now then he can come and have a look later.

U: OK. Yeah.

D1: I mean, what, what Trevor said and maybe we can do it at this stage is that he has at times done a form of the type that we're talking about here. It's just that it would be simple check boxes

U: Yeah.

D1: Uh, with a couple [*picks up copy of form*] of sort of button presses or key presses or whatever. The form he currently uses is like that and on this thing you put a tick to represent an enquiry in in one of these boxes.

U: Yes.

D1: Now he suggested that this layout isn't exactly what he wants and I suggested a different one

U: Mm.

D1: which is uh

U: You've obviously seen this before

D1: Either general enquiry, sales enquiry, which machine, national national services, which is sort of external things

U: Indeed.

D1: instead of those three

U: Mm-mm.

D1: software instead of that

U: Mm-mm.

D1: timetabling and documentation instead of that.

U: Mm-mm.

D1: We said passwords is OK.

U: Mm-mm.

D1: Uh, I suspect again we shouldn't enforce uh having to fill them all

U: Mm-mm.

D1: because at times he's going to want to go to some of them

U: Yeah.

D1: For various reasons

U: Mm.

D1: But that's essentially five, this this five here [*points to TM*]

U: Yeah

is what we want some something on these lines.

U: Yeah.

D1: Uh, so we're, I mean, records, what we're recording here is, uhhh [*pause*]

U: It's on this paper form, it's already done two, you just tick three of those boxes

...

D1: ... but usually they're exclusive

U: Oh.

D1: For example, if it's uh a general enquiry it won't also be an enquiry on a PC.

U: Yeah.

D1: Uh, so in most cases they're exclusive but you do get occasions where for example if someone came in and asked for a Macintosh manual do you click that or that or both?

U: Yeah.

D1: Mm. So there's questions there. Uh, we're going to go through it with Trevor and sort of agree which are the best to do. But for the moment five uhh I, I I don't want to break down recording a query

U: Sorry [*reaching for a pen behind D1*]

D1: OK. Uhh, it should, I mean it's seems to be quite simple in terms of breaking it down for us at this stage specifying what we're going to record.

D2: It's only Trevor that uses the form. Trevor uses it.

D1: Yeah. Yeah. So I mean, well

U: There are other, there are other reception people.

D1: Yeah the reception people use it.

D2: They all use that form?

D1: Yeah. Yeah. If they're just taking the queries and passing them straight on. Uh.

U: I mean you know there's Carol Misselbrook and uh A. N. Other at the moment

D1: Right, so can we break down five now with just whatever uh of these seem [pause] relevant but for the moment uh I'm not sure which of them, one's national services, uh sales, general enquiries, courses and that. Uhh. In fact I wonder is it worth reading this in the absence of Trevor.

U: Uhh, I would suspect given his own strengths uh it gives us a better chance of getting through the other ones.

D1: Yeah. Yeah, 'cause that's that's very detailed decisions there

U: Yes.

D1: and what we can, we won't, we just want to record the query aah. What he needs to record uh that isn't on that at the moment is. That's the nature of the enquiry. [*points at paper form*]

U: Mm.

D1: Which is one of those. If we could produce a simple thing to do that, nature of the enquiry, the others are name and department because the only reason that, well, two reasons that they want to record this at all, one is to say that uh that they are working during the day and the other is to say to other departments that enquiries have been made and hence they are being charged.

U: Yes.

D1: So we need names of departments and others.

U: Well, OK, I mean discuss that with Trevor by all means but I suspect you may find that he will say that typing in a name is too time consuming.

D1: Yeah. Yeah. I feel that myself. Uh

U: The department could be

D1: The department, all right. But can can can we as subtask of recording query here can we have record department and record query type and then go through it with Trevor.

[D2 writes on whiteboard]

D1: And. Right, so the next thing there is identifying a referee and referring a query on. Uh, the first thing we've already identified is identifying who to refer it to uh so we've got that. [draws on TM] Uh. We've now got to decide what's the best way to ident, well, identify who to refer it to and then refer it [draws on TM]

U: Yeah.

D1: obviously

U: Mm.

D1: So, the question is what do we need to do to identify who to refer it to and the partial answer at the moment is the expert list that you talked about, didn't you, that is held at reception.

U: Well, to be frank, I mean it exists there but I rather doubt that it helps very much because except when there are kind of temporary staff there, which does happen but, I would hope that it's only sort of ten per cent of the time you're going to have someone who is both temporary and there's nobody else around.

D1: Yeah.

U: Ninety per cent of the time they do it from the knowledge they have in their head or by asking the guy who's sitting next to them.

D1: Right. Uhh. So to to identify the referee you need to [pause] know uh staff specialities.

U: To a degree. I mean [pause] that's useful as a shortcut but in general um you shouldn't really need to at the reception level know that much about staff specialities 'cause if you pass something on to the help desk they are much more in the business of being expected to know about who's got which specialities than reception are.

D1: Right. Uhh.

U: Or liaison as well.

D1: Yeah. Trevor said that by default they'd have sent it to one of those if they couldn't figure out uh who it was.

U: Yeah, I mean, again, y'know, they may they may modify my comment but my impression of the way they work is that if they've been here long enough they've built up a knowledge in their heads of who has specialities

D1: Right.

U: in which areas. But they don't really need to do that because they can always just ask the person on the help desk or liaison and they'll do the reference to the more specialised one.

D1: Ah.

U: It's just a convenient shortcut if reception can do it directly.

D1: Right.

D2: So don't you just have ...

D1: ...

U: Or fault line.

D1: Or fault line. Well, that's I mean they identify it as faults.

U: They do, yeah.

D1: So Referring a query. Given that a query hasn't been recorded at this stage what in referring it are they simply telling the user go off and speak to so and so?

U: Ah, if it's help desk or liaison yes ah if it's fault line I guess what they are doing under the current system and maybe it'll be different in the new system is sending the enquiry to [*pause*] the people who look after faults on the user's behalf. Similarly, if they did refer it directly to a technical consultant in the back office um they would call that person to say there's a user.

D1: Right. Umm.

U: Y'know it's it's a case of if it's a sort of reluctant person

D1: Yeah.

U: They tell the person to come to talk to the user. If it's somebody five feet away they tell the user to go to one desk or the other.

D1: Right.

Appendix 4

This appendix presents a transcript of the sample of video recorded interaction from the CS7 meeting which was used in the analyses of chapters five and six.

U: There is of course the quibble there that we do deal with people from outside the College so you can have somebody who is none of those three.

D1: OK. Right. So there is at least a fourth category.

U: Mm.

D1: Uh, if we add the customer details, uh, name, uh, login, is there any particular order you'd like to see those in?

U: Phhh. Um. Yees. There are sort of two ways of approaching that. There could be sort of the order which is most convenient for us or there could be the order which is likely to be the way the user will most often come out with it. Um, I don't really know which of those to give more weight to. Um. If you base it on what the customer's likely to say I guess they'll tend to give their name before anything else. ... almost always have to prompt them for the last three.

D1: Right. Are you saying uh sort of ... customer details are you assuming a left to right ordering uh as natural?

U: Well ...

D1: So which, which order then would be most useful to you

U: Uh, I guess frankly to us the most useful things would be first contact details, then login then department, then type and last of all name. We don't care who they are, we want to know what categories they fall into [*smiles*].

D1: Right.

U: For our purposes.

D1: Right. But they're liable to give their name first, you, you

U: Oh yes. As individuals, they tend to think they matter [*smiles*].

D1: Contact gets pretty much priority in both. Uhh. So. Is is that what you said was the ordering or is that You just said name would be last.

U: For our convenience yes.

D1: And contact details first?

U: Umm. Sorry?

D1: Did you say contact would be first?

U: Yes I'm just thinking actually sort of name as far as we're concerned kind of forms part of contact

D1: Right.

U1: details.

D1: So. Right.

U1: Umm. I mean I guess what we're talking about in the little box that we're calling contact is things like phone number and email

D1: Yeah.

U: and so on. So is there necessarily even a single box? Um. I mean it varies in fact because if you've got the email you don't need the personal name as well. But if you've got a phone number you do need the personal name as well.

D1: Yeah. Uhh. So so contact there is important both in terms of what they give you and what you need. What about the other three? Is there any greater importance or order to them?

U: Uhhh. Well login tells you more than most other things

D1: Right.

U: because you know that actually allows you to find their filestore on the computer

D1: Yeah.

U: Uh.

D1: Is login ...?

U: Well you know, it's it can be related to contact because as I say it allows you to track down sort of where details of the user have been registered which can include contact. Uh, um, but that's not reliable.

D1: Mm. Are login and email synonymous in this instance then or are they different?

U: No. They're unrelated. Well, you can deduce one from the other, um, but not reliably.

D1: Right. And department, customer type which is most, more or less important?

U: Uh it varies I'm afraid.

D1: Right.

U: Probably more often than not the department is more significant and

D1: Yeah.

U: customer type is just for formality's sake but you know there are situations where it's actually critical to uh what somebody's feasibly able to do ...

D1: Yeah. Uh. Uh. Does that is that approaching a reasonable ordering? [*points to prototype*]

U: Well that's for our convenience.

D1: Well for your convenience, presumably it's also to your convenience if when they phone you up if they tell you in an order which

U: Yeah.

D1: is reasonably close

U: Indeed. Sure.

D1: So we're talking it's name first [*pointing to Post-It*] then contact then login?
[D1 moves Post-Its on paper]

U: I mean, sorry, I mean I'm not memorising this well enough but I have a feeling we've wound up saying that they're actually going to be in much the same order whichever criteria we're using.

D1: Yeah.

U: Yes

D1: That's fine. Uhh.

U: ...

D1: And left to right top to bottom.

U: Mm?

D1: Left to right top to bottom is uh conventional ordering that you

U: Well as I say it's the way I've been used to reading. You have horizontal lines and you read them from left to right.

D1: Uh again to take it further we really need to get it on screen and ... but we at least get a rough idea of the layout here. Uhh from ... details uhh probably

D2: menu for customer type

D1: this

D2: ... you have these three fields ...

U: Yes.

D2: ... could be on a menu

U: Oh absolutely. I always I always assumed that you'd need to have menus for every possible thing. [*smiling*]

D2: and departments

D1: Yeah that's a reasonable one.

D2: and for contact

U: Well I don't quite see how you can have menus for contact.

D1: Well

D2: There's phone email

U: Oh I see what you mean yes sure I mean you have subfields for different kinds of contact.

D1: You need to to enter a particular field I mean certainly subfields there are not not something that you choose from but something you type into. Uh so what are the subfields of contact?

U: Well

D1: Email phone

U: Email phone postal address and might as well have fax.

[D1 writes on a new Post-It and sticks it on paper]

D1: But they would need to be uh fields which you filled in. Clearly it's the content of the field that you're interested in not just the label.

U: Yes.

D1: Uh. Departments on the other hand

U: Oh certainly that can be a menu

D1: Right. We did in fact have one [looks in papers] uh because uh with Trevor for the other form we wanted a similar list of departments and it turns out that the list is very long.

U: Yes.

D1: So with Trevor we looked at breaking it down by faculty.

U: Yeah.

D1: And and then having a cascade menu for each faculty

U: Yeah.

D1: which made it reasonable

U: Yeah.

D1: so uh maybe we could use that. Customer type again. Just three types?

U: Well no four because there's other

D1: Yeah.

U: when they're not actually a member of the College

D1: [writing on Post-It] Staff. Postgrad. Undergrad. Other. And that [sticking Post-It on paper] goes ... Right. ... And that's the faculties [sticking large Post-It on paper] [U nods] and then you can go sideways and choose each. I mean it was horrendously long.

U: Mm. Yes. Yes sure. I'm sort of taking it as read that you'll um take the opportunity to um do things economically when the opportunity arises.

D1: Yeah. Right. Well. So the ones you actually have to type in will be that that and that and you can just click on those [pointing to each Post-It in turn].

U: Mm.

D1: Probably ... which is here. We've again

D2: Shall I move this one here [*taking hold of corner of large Post-It with faculty list*]

D1: Yeah that's probably a good idea.

[*D2 peels off Post-It and hands it to D1 who puts it aside*]

D1: We've already looked at four and as soon as uh [*D1 begins pointing to paper copy of software prototype screen dump*] problem details. Priority priority and problem description. And problem description was broken down into initial description and further details. I'm not sure if you were still there when we had this part of the discussion the other day.

U: No.

D1: No I think that was just after you left.

U: Mm.

Appendix 5

This appendix presents materials from the CS project usability evaluation. The materials for the CI project evaluation were identical in all but the details of the software and the corresponding tasks required. The first page below reproduces the introduction sheet which was given to the subjects.

Subsequent pages present the tasks which the subjects were asked to perform in the roles of Fault Report, Help Desk, Technical Query and Reception users respectively. Each of the Fault Report, Help Desk and Technical Query task sheets is accompanied by a sheet which was used in the evaluation as a mockup of the user receiving a message that a query had been referred to them and giving the reference number.

Introduction to the Action Request System usability evaluation

Thank you for agreeing to help with this study. We want to evaluate the usability of the new system for recording queries and fault reports and for maintaining an inventory of Computing Services equipment.

The software is based on a system for handling customer queries and fault reports. The software is intended to support users in logging these queries and reports, the work carried out on them and solutions provided to customers. This log should then serve as a database for reference with subsequent queries and fault reports. An associated part of the system is intended to support the maintenance of an inventory of equipment owned and supported by Computing Services.

The aim of the study is to find out how easy the software is to use by people like yourself. We want you to use it to help us find out what problems the system poses and how it could be improved.

We'll give you some standard tasks to do using the software. The aim of this is to allow us to get some feedback about how the software supports this activity.

We are particularly interested in situations in which the system encourages you to make errors in selecting commands and misleads you about what it will do. We are also interested in extra commands that would make the system easier to use.

To get this feedback we'll use a question-and-answer technique. This involves three things.

1. We want you to think out loud as you do each task, telling us how you are trying to solve each task, which commands you think might be appropriate and why, and what you think the machine has done in response to your commands and why. You could think of this as giving us a running commentary on what you are doing and thinking.
2. Whenever you find yourself in a situation where you are unsure about what to do or what effects commands might have, ask us for advice. If you ask us what you need to know, we'll suggest things for you to try. If you get really stuck, we'll explain exactly what you have to do.
3. In addition, we'll ask you questions about what you are trying to do and what effect you expect the commands you type will have. This is simply to find out what problems there are with the system. During our conversations, we want you to voice any thoughts you have about parts of the system which you feel are difficult to use or poorly designed.

Remember it's not you we're evaluating, it's the software. We are interested in what you think so please don't be reluctant to say what you think about the system and the tasks you are asked to perform.

Fault Report tasks

1. Receive a fault report from Reception or Help Desk.
2. Determine that the problem is really a technical query (not a fault report) and refer the query to an appropriate colleague.
3. Receive a fault report directly from a customer.
4. Log the fault report from the customer.
5. Solve the problem and complete the system log for the fault report.

Fault Report

Report 1

System log reference number: 80

Help Desk tasks

1. Receive an enquiry from a customer in person.
2. Begin to deal with this enquiry.
3. While still dealing with this enquiry, you receive another enquiry from a customer by telephone. Deal with this telephone enquiry.
4. Determine that you can't answer the telephone enquiry and refer it to an appropriate colleague.
5. Continue dealing with the original enquiry.
6. Answer the original enquiry and complete the system log for it.

Query referral

System log reference number: 80

Technical Query tasks

1. Receive a technical query referred to you on the system by a colleague.
2. Begin to deal with this referred query.
3. While still dealing with the referred query, you receive a new query from a customer by telephone. Log this telephone query on the system.
4. Determine that you can't answer the new query and refer it to an appropriate colleague.
5. Continue dealing with the original query referred to you by a colleague.
6. Answer this referred query and complete the system log for it.

Query referral

System log reference number: 80

Reception tasks

1. Receive an enquiry from a customer in person.
2. Determine whether the problem is a fault report or a technical query and deal with it accordingly.
3. Receive another enquiry from a customer by telephone.
4. Again, determine whether the problem is a fault report or a technical query and deal with it accordingly.

Appendix 6

This appendix reproduces the software development survey questionnaire which was sent out by email (see chapter two). The questionnaire sent by normal post was comprised of the same questions.

Software development survey

Replies to: eamonn@dcs.qmw.ac.uk

I should be very grateful if you could find the time to complete the enclosed questionnaire on the experiences of software developers and return it electronically to the address above. The questionnaire should take no longer than 30 minutes to complete and filter questions should ensure that no respondent has to answer every question.

I am conducting research into the development of interactive software systems, with a particular focus on the capture and analysis of user requirements. It is intended that this research should lead to the synthesis and collation of principles and techniques of good practice in software development. I believe that the work of practising developers provides the clearest guide to useful methods and techniques. I am interested, therefore, in soliciting developers' views of their working practices. This questionnaire is designed to contribute to this information gathering.

The first three parts of the questionnaire ask for background information on the respondent's individual and group work experience to provide context for the responses in the fourth part. Part 4 asks for information on a single software development project. I should be grateful if you would derive your answers in this final part of the questionnaire from your experiences of one specific project on which you have worked, rather than from your more general experiences of software development.

If possible, please select a project for Part 4 which was commercial rather than for research purposes, which was intended to develop a highly interactive software system and which has been successfully or unsuccessfully completed. If you have not worked on a project which meets these constraints, please relate your answers in Part 4 to a project on which you have worked.

The questionnaire does not elicit the names of any organisation, project or product. All replies are treated as anonymous and confidential. If you wish to provide fuller answers or additional information, please feel free to do so. Confidence and commercial sensitivity will be respected.

Many thanks for your time.

Questionnaire

In confidence

Part 1: Your Background in Software Development

1. What is your role within your development group?

2. How did you receive your training in software development?

- First degree
- Postgraduate degree
- Conversion course
- Employment based training
- Other. Please specify.

3. Could you please give a brief description of your experience in software development (eg, types of project worked on, length of experience)?

Part 2: Your Software Development Group

4. What is the size of your development group?

5. What is the nature of your current development group?

- bespoke software developer
- mass-market software developer
- in-house software developer for a parent organisation
- Other. Please specify.

6. Is there a sales or marketing group associated with the products developed by your group?

- Yes. If yes, go to question 7.
- No. If no, go to question 8.

7. Could you briefly describe in general the nature and extent of normal interaction amongst sales/marketing, the developers and the users and customers of the software?

8. How are the members of your development group organised (in terms of roles)?

9. What types of software project has your group tackled?

Part 3: Group Policy and Procedures for Software Development

10. Does your group have a defined software development policy?

- No.
 Yes. If yes, please give a brief description.

11. Does your group use one or more of the following?

	Never	Occasionally	Often	Always	Never heard of it
(a) SSADM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(b) JSD	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(c) CAP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(d) RAD	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(e) JAD	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(f) VDM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(g) DFDS	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(h) Z	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(i) SSM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(j) Multiview	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(k) Yourdon-deMarco	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(l) Prototyping	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(m) Participatory design	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

If you have not used any of the above, go to question 14.

12. Where you have used one or more of (a)-(m) in question 11 above, would you in general characterise its use by your group as

	strict adherence	loose but complete adherence	using appropriate parts as desired	not used correctly
(a) SSADM	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
(b) JSD	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

(c) CAP	[]	[]	[]	[]
(d) RAD	[]	[]	[]	[]
(e) JAD	[]	[]	[]	[]
(f) VDM	[]	[]	[]	[]
(g) DFDs	[]	[]	[]	[]
(h) Z	[]	[]	[]	[]
(i) SSM	[]	[]	[]	[]
(j) Multiview	[]	[]	[]	[]
(k) Yourdon-deMarco	[]	[]	[]	[]
(l) Prototyping	[]	[]	[]	[]
(m) Participatory design	[]	[]	[]	[]

13. For each of (a)-(m) which you or your group has used, please indicate how useful you found it to be.

	Indispensible	Very useful	Useful	Unhelpful	Hindering
(a) SSADM	[]	[]	[]	[]	[]
(b) JSD	[]	[]	[]	[]	[]
(c) CAP	[]	[]	[]	[]	[]
(d) RAD	[]	[]	[]	[]	[]
(e) JAD	[]	[]	[]	[]	[]
(f) VDM	[]	[]	[]	[]	[]
(g) DFDs	[]	[]	[]	[]	[]
(h) Z	[]	[]	[]	[]	[]
(i) SSM	[]	[]	[]	[]	[]
(j) Multiview	[]	[]	[]	[]	[]
(k) Yourdon-deMarco	[]	[]	[]	[]	[]
(l) Prototyping	[]	[]	[]	[]	[]
(m) Participatory design	[]	[]	[]	[]	[]

Part 4: A Software Development Project

For the remaining questions, please choose a single software development project and relate all your answers to this one project. If possible, select a software system which is commercial rather than research-based, has a high degree of user interaction and has been successfully or unsuccessfully completed.

14. What were the roles in the development team, including your own?

15. How long did the project last?

- person-hours
- weeks

16. Why was there a project at all? Did the project spring from a developer's idea or invention or from a request by a customer?

17. What type of software was being developed?

18. At project inception, how familiar was the development team with the users' work?

19. Was any structured development methodology (eg, SSADM, JSD) used?

Yes. If yes, go to question 20.

No. If no, go to question 22.

20. Please give a brief description of how the methodology was employed (eg, how strictly)?

21. How useful did the methodology prove to be?

22. Who decided (and how) who were the users?

23. Who (if anyone) was consulted for user requirements?

24. How were user requirements discovered?

25. How were user requirements recorded?

26. How were user requirements used in the design process?

27. Was an explicit user requirements specification produced?

Yes. If yes, go to question 28.

No. If no, go to question 32.

28. What form did the user requirements specification take? Please give as much detail as possible.

29. How (if at all) was the user requirements specification validated/evaluated?

30. Did user requirements change after the user requirements specification had been produced?

Yes. If yes, go to question 31.

No. If no, go to question 32.

31. Were these changes incorporated into the user requirements specification?

No.

Yes. If yes, please describe how this was done.

32. How (if at all) was the design evaluated against user requirements?

33. How did the final product compare with the user requirements?

34. How happy was the development team with the product?

35. How happy were the customers with the product?

36. How happy were the users with the product?

37. How was user satisfaction with the product ascertained/measured?

38. Would you have done it the same way again?

No.

Yes. If yes, please describe what could have been different.

Thank you very much for your time and effort in completing this questionnaire.

Would you like to be informed of the survey results?

No.

Yes.

Eamonn J. O'Neill
PhD researcher

Human-Computer Interaction Lab
Dept of Computer Science
Queen Mary and Westfield College
University of London
Mile End Road
London E1 4NS
UK