THE UNIVERSITY OF HULL

Human and Social Aspects of Software Development for Complex Organisations. An Online Ethnography of Software Developers

being a Thesis submitted for the Degree of Doctor of Philosophy in the University of Hull

by

Cecilia Loureiro-Koechlin (MBA)

July, 2006

Table of Contents

List of Figures

List of Tables

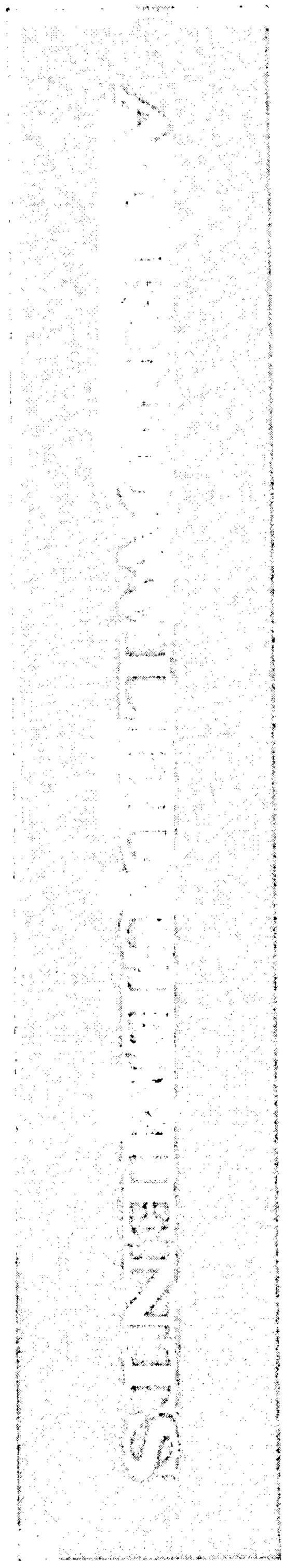
I would like to thank

the people I have worked with, colleagues and users, those who gave me the inspiration to start this PhD

Wendy Robson who was patient with supervision

my husband, family and friends

I am in debt with all of them for their comments and support at different times and on many aspects of my research



Abstract

Summary of Thesis submitted for the PhD degree
by Cecilia Loureiro-Koechlin

on

Human and Social Aspects of Software Development for Complex Organisations. An Online Ethnography of Software Developers

This thesis addresses the problem of human and social issues that affect software development. It is situated within the field of Information Systems and focuses on the processes of software production used within complex organisational processes: particularly decision-making, collaboration and workflow. Human and social issues are problems and situations caused by the members of the target organisations, software developers, and the dynamics of their working and social interactions.

The objective of this thesis is to identify these human and social issues and see how they affect software developers' work. The methodological approach adopted here, is designed from the interpretive point of view. This study takes the perspective of software developers as they possess practical knowledge of complex business settings and current software development practices. Online ethnography is the chosen method that allows this investigation access to virtual communities in which software developers work and exchange experiences. The design of this thesis is as follows:

1. Online data is collected which reflect software developers' beliefs about their work and their target organisations.
2. Data categories are created which show a picture of the current state of affairs in software development.
3. An interpretive theory building strategy is used to create a model of software development based on data categories.

The final outcome of this thesis is developed as a complement to Orlikowski's (2000) structurational model of enactment of technologies-in-practice and takes the form of a descriptive, theory based model. This model contributes to the deeper understanding of software development issues. It presents human and social issues that affect the production of software within three different contexts: software development environment, software development practices and complex business organisations. The model generated in this thesis also suggests that software developers' perceptions of human and social issues in complex business organisations are influenced by the developers' background knowledge and experience.

PART I: INTRODUCTION

Part I of this thesis introduces the reader to the context of the research problem, research questions and research methodology.

Chapter 1: Introduction

This thesis is called "Human and Social Aspects of Software Development for Complex Organisations. An Online Ethnography of Software Developers." It looks at how human and social issues in business organisational contexts and development environments affect software developers' work and the software they produce. The argument of this thesis is that human and social aspects play a significant role in shaping developers work, beliefs, behaviour, interactions and working practices, and that this in turn affects the nature and quality of the software they develop.

This thesis is situated within the field of Information Systems (IS) research and focuses on aspects related to software development (an activity within Information Systems Development) and its product: business software (this thesis is concerned with decision support systems, software designed to support group work and social software, see section 3.1 of chapter 3). The aim of this thesis is:

> to identify and explain how the social issues within the software development environment and the social aspects of complex organisations shape software and the process of software development.

Within the context of this thesis, human and social aspects entail those factors which are normally understood as "people issues" (Cockburn, 2003). In general, people issues are those kinds of problems or situations which are caused by people: their behaviour, their interactions, the way they are organised and their beliefs which affect their work and the work of others. Because they affect people, these issues need to be recognised and addressed in order to improve the quality of work and the products or services provided. Specifically, in the context of this thesis people issues emerge from the human nature of software developers and members of business organisations, users of business software (human aspects), and their working and social interactions (social aspects) which

affect software development (software developers and business software as a product). The social motive in organisations is groups of people working (Coakes et al., 2000, 9) using technologies such as e-mail, the telephone and business software. This applies to both the organisational and the software development environments. Thus the contexts of concern for this thesis are the software development environments, including developers using software development methodologies and software tools and the business settings (with people using business software to work). Following the above, I designed three research questions, which address each context separately:

> Q1: What are the human and social aspects of business environments that make software development complex and difficult?
>
> Q2: What are the social aspects in the development environment that allow or hinder the identification of social issues in business settings during the development of software?
>
> Q3: What aspects of current software development methods address or neglect social issues in business settings? What is social software and does it have a role to play in improving these practices?

The above research questions are concerned with the non-technical aspects of the process of software development. Although it is regarded as a technical activity, software development by its very nature is a human and social activity (Weinberg, 1971) that involves the design, creation, implementation and maintenance of software. However, at work, software developers focus more on exercising their technical skills rather than their interpersonal skills (fieldwork data, see category C4-5 in section 6.4 of chapter 6). The development of technologies to support ongoing and dynamic interactions in organisations (i.e. that consider human and social issues) requires an approach which is very different from engineering (Coakes et al., 2000, 9), the prevailing, technically oriented software development paradigm. One of the main weaknesses of software developers is their lack of sensitivity to the human and social aspects within the business organisations they develop the software for

and the environments they work in. Lack of sensitivity here means either a lack of awareness of human and social issues by developers or if aware, the inability to interpret as to what extent those issues should be considered in the design of the software and/or the development practices they perform. Also this lack of sensibility is extended to the software they develop which do not address the human and social aspects of the human systems they are going to serve. This is claimed to be one of the causes for the high rates of software failure (Cooper and Reimann, 2003, 25). From the above it may be argued that software should ideally be designed to *fit* the human and social characteristics of the organisations, users of software, and the needs of their members.

The "fitness" of business software for a particular purpose (which includes consideration of human and social issues) varies from generic software, which fit many kinds of business settings, to personalised or customised software which need only fit one particular organisation. Though in the case of generic business software, customisation work is usually required to adapt the software to the needs of their users. The range of software considered in this thesis is limited to:

- either generic or customised business software (does not include office tools);
- designed for business organisations or information systems whose human and social aspects are difficult to understand and address by software developers.

Similarly, the range of business organisations considered in this thesis are the ones that contain decision-making, workflow, collaborative work and similar processes, the behaviour of which is unpredictable and adaptable, and in which work is performed by individuals and multi-groups with intricate social interactions. This thesis identifies these types of organisations as complex organisations. As a consequence of this exploration, a model of human and social issues in software development

has been developed that reflects the different areas that software developers should focus on to improve their awareness of human and social issues that affect their work and the quality of the software they develop. The objective of the model is to build a picture of software development for complex business situations that: describes and explains the process that software developers go through when they make sense of social aspects of the target organisation, the information system and the users that the software has to serve and when they assess and compare the technical aspects of software with the social issues within business settings.

As shown in this initial section of the thesis, I will use the term "thesis" throughout this document to refer to the argument of this PhD research project, its aim and research questions, its outcome and the document itself. The term "study" will be used to refer to research papers and in conjunction with description of activities performed in this research project for example as in data collection for this study.

1.1 Research Motivation

This thesis was initially motivated by my personal interest in understanding the human and social aspects of software and information systems development. This interest emerged from my previous working experience as a software developer and systems analyst at the computer centre of the University of Lima-Peru. The initial idea for this study developed as a result of my participation as a systems analyst in a software development project during the years of 1999 and 2000. The aim of this project was to design and implement a Business Intelligence system for the University General Assembly and the University Board of the University of Lima. The University Assembly is the highest hierarchical unit of the University of Lima and meets twice a year to decide on the University's annual plan of operation and development. The University Board is the second highest hierarchical unit of the University.

The board meets monthly to discuss issues such as the control of the University's budget, and to supervise and evaluate the academic processes of the University, and to create, modify or remove academic departments and career programs. To support these activities, the University's unit of Information and Systems (where I worked) was asked to design and implement a Business Intelligence system and I was the leader of that project. Business Intelligence is a commercial term commonly used to refer to database and reporting applications that aid decision making in organisations. With Business Intelligence tools, data from operational applications (internal and external to the organisation), such as marketing, payroll, accounting, and academic records, are consolidated for management use. Thus Business Intelligence allows for internal and external integration of data (i.e. integrating data from internal processes and outside organisations respectively). Business Intelligence tools usually comprise data warehouse, data mining and reporting tools such as OLAP (see glossary for a description of these tools).

At the beginning of the project the project team was taught a methodology for designing the data warehouse (known as the star and snowflake schemas) and reports. These methodologies relied on the availability and nature of the operational data (in the case of the University of Lima, this data came from the academic record and financial systems). Additionally the team members had knowledge of structured (Yourdon, 1989) and object oriented methodologies (Rosenberg and Scott, 1999) for designing information systems and software. Initially the team had a preliminary idea about how the decision-making system could be designed. This idea was based very much on our knowledge of the nature of the Business Intelligence tools and the data available, and not on the University's activities. Figure 1.1 shows a schema of the technology architecture as understood at the beginning of the project. This schema shows how the team planned to design a data warehouse and a set of reporting tools to be used by the members of the University assembly and board.

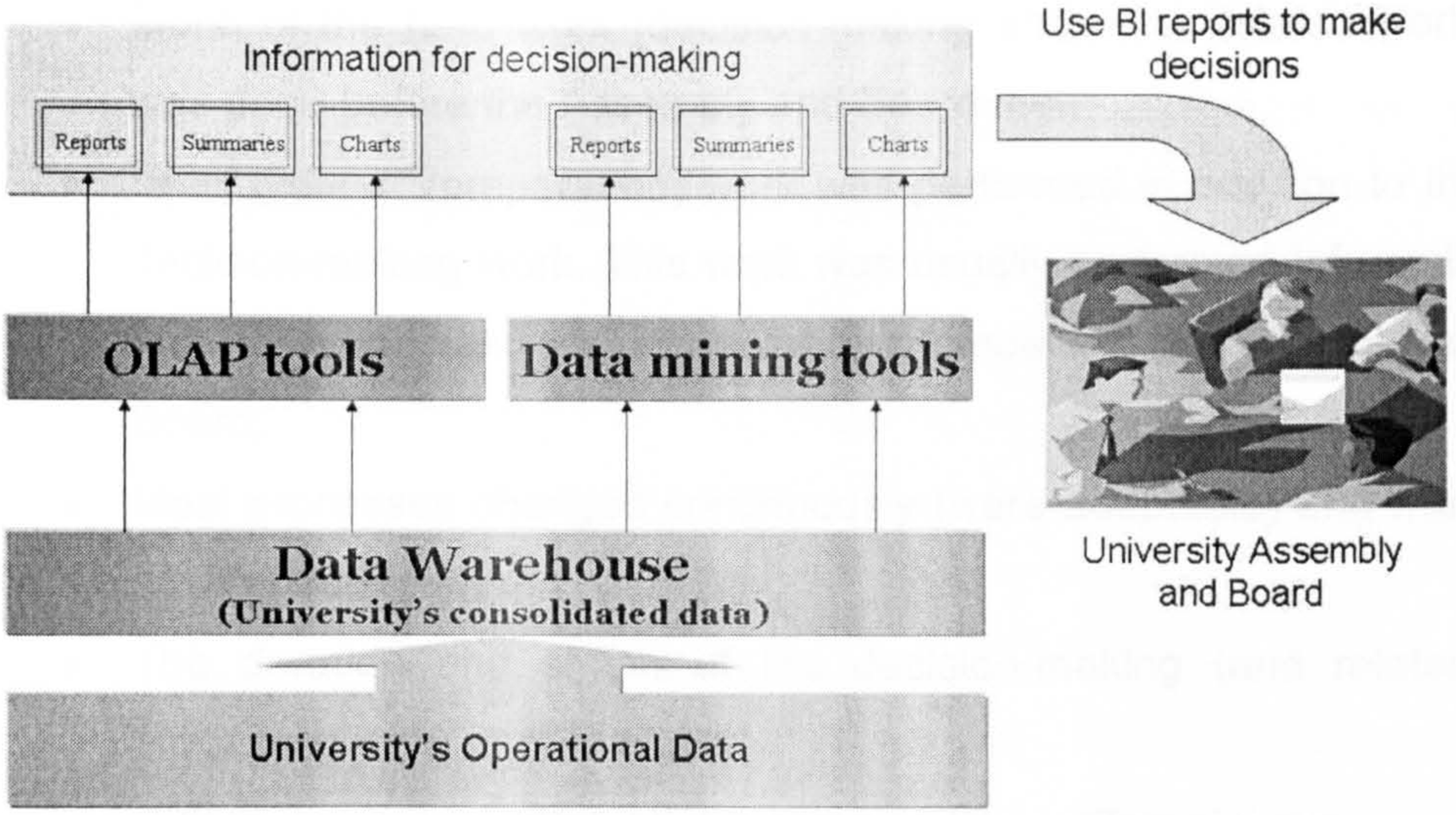


Figure 1.1 Model of a Business Intelligence (BI) system for the University of Lima
Source: Business Intelligence Project – University of Lima (1999)

The specifics of the design of the data warehouse and reports depended on the needs of the members of the University assembly and board. The team decided (myself included) to carry out interviews with them to see what their data needs were. During the interviews the project team realised that the project had to be extended in scope to account for activities and needs of University staff whose work was related to the university Assembly and Board but did not belong to them. In addition to the provision of data the team found that the needs of the decision-makers involved support for their workflow and collaborations. This made us reconsider our perception of the system's architecture as it assumed that the decision-making processes were static. We then tried to incorporate the features of workflow and collaboration into our model. However there were issues which made this difficult. The dynamic and shapeless nature of the ongoing work of decision-makers did not match our structured vision of an information system. The following outcomes from the project interviews reveal the nature of the university's organisational processes as related to decision making:

- There were a big variety of activities which needed to be considered;

- Much of the hard work (decision-making and other related work) was done before the Assembly and Board met;
- "Background" (preparation) work was performed in addition to the decision-making work. This work was usually performed informally and not recognised (or unknown) by members of the Assembly or Board;
- Most processes changed continuously (were adaptable) and were not standardised;
- The duration and shape of the decision-making (and related) processes were unpredictable;
- Different people performed the same tasks in different ways;
- Some people's styles dominated the others;
- There was individual as well as multi-group work.

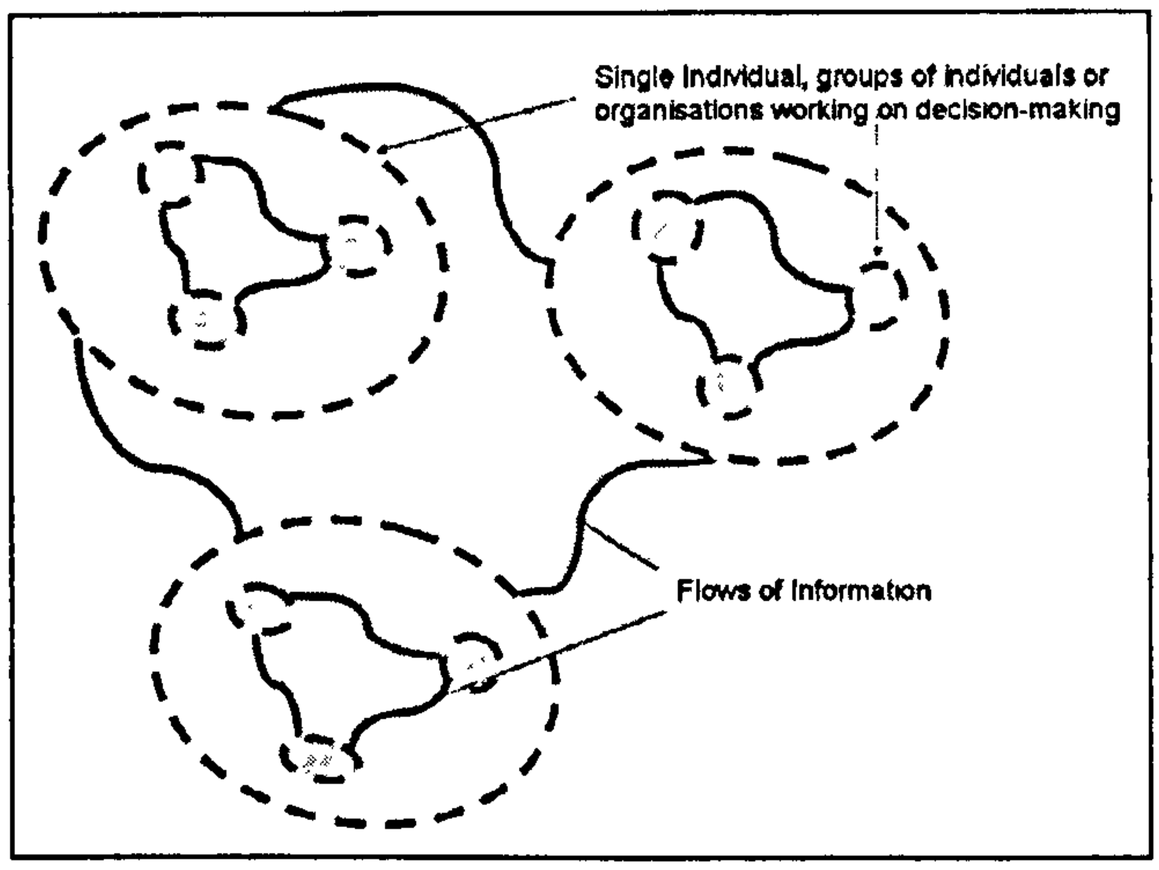
In addition to these findings listed above which are related to the nature of the processes, I also personally realised that the following issues made the design and implementation of software more difficult:

- The information given by the decision-makers was ambiguous (contradictory);
- Users had different views of their jobs and other people jobs;
- Few people (if any) had an understanding of the whole situation;
- Most people were only concerned with the issues for which they themselves were responsible.

Although I did not continue to participate directly in the project, it did change my perspective of information systems and software development. Instead of focusing on the technological aspects of software development I came to appreciate the effects that different and changing demands of people had in software requirements.

The issues presented above, though emerging from a particular project, represent a wider variety of situations. These issues are not technical in nature but come from the human character and social interactions

between people and make organisations complex in terms of IS development, something that is not adequately understood by IS developers and methodologists. (A discussion of software failure due to this reason is presented in "Is software only a technical concept?" which is part of section 3.1 of chapter 3.) I have extracted these issues to elaborate a model that shows how these human and social aspects shape decision-making, workflow and collaborative processes in such complex organisations. The purpose for creating this model is first to present an initial ontology of complex organisations at the beginning of this study and second to contrast this view with the picture first presented in figure 1.1. This model, though, does portray only complex organisational environments and does not include the software development environment which is considered later in this section and in the rest of the thesis. For me working arrangements and practices in the software development environment depend partially on the characteristics of complex organisations as developers have to adjust their beliefs and practices to develop the software for them.



- Circles are actors
- Actors can be individuals, group of people or organisations
- Group and organisational boundaries are defined by people's perceptions (dotted lines)
- Continuous lines represent Flow of Information
- Processes can be started and ended by any actor
- Processes can take any shape
- Processes can split in two or more processes; and two or more processes can merge in one
- Extraction, creation and use of information is implicit in the model

Figure 1.2 Model of Complex Organisations

Figure 1.2 shows a picture of decision-making *processes* in complex organisations. In this model circles represent actors. Actors could be individuals, group of individuals or entire organisations. Groups of people and organisational boundaries are represented by the dotted lines meaning that they are only defined by the people who belong to the group or organisations or by the outsiders who interact with them. The lines stand for the flow of information between the actors, representing workflow or collaborative processes. The model also represents interaction of people within any level of the organisation(s) as well as through different levels; i.e. interaction of people within and between the units of an organisation, interaction within and between groups (or units) in an organisation; or the interaction between organisations. There are no standard procedures about how to interact or with whom. In fact every actor has the power of determining the activities to be performed by being a decision maker. As a consequence the processes may take any shape and number of steps. A decision-making situation might start or end at any point of the organisation and different number of people or groups may be involved.

The model of complex organisations contrasts very much with the structured vision of organisations and information systems I had at the beginning of the project. Structured perspectives of organisations are based on positivist ontologies. They seek "to explain and predict what happens in the social world [(e.g. organisations)] by searching for regularities and causal relationships between its constituent elements" (Burrell and Morgan, 1979, 5). The model of a Business Intelligence system presented in figure 1.1 shows a preconception of organisations which have formally, standardised procedures (Laudon and Laudon, 2005, 6), and rationally coordinated activities with the aim of achieving a goal (Lucas, 1994, 78). This perspective assumes that organisations have "hard" boundaries and little if no social interactions, tensions or conflicts between their members. However, the outcomes of the Business Intelligence project interviews show a different view of the organisation. This second picture identifies a more unstructured and unpredictable view

of organisations in which the human and social factors prevail. Checkland and Holwell (1997) state that organisations are reified social collectivities (Checkland and Holwell, 1997, 80). According to them, the existence of an organisation depends on the recognition of its members and outsiders. Therefore informal organisations could exist alongside formal ones as long as their members recognise their existence. Organisations do not have physical location or shape but the one given by people. People within the organisations make sense of their purposes and decide on the activities that are needed to pursue their goals. People outside the organisations will also identify the organisation's boundaries and have a perception of the purposes and activities performed within the organisation. More perspectives of organisations that consider their human and social aspects exist. An in depth discussion of human and social aspects of organisations and information systems is presented in section 2.1 of chapter 2.

The perspectives of organisations and information systems presented in figures 1.1 and figure 1.2 are not compatible with each other neither are they compatible with software development needs. The project team perceived the organisation as an entity which fitted the structured and predictable characteristics of the software and data available; and which matched their also structured and predictable views of the process of software development. However, the organisation presented different characteristics. Thus there was a mismatch between the unstructured, changing, and unpredictable nature of the organisation and the structured and unambiguous nature of the software. In the case of this Business Intelligence project software developers struggled to connect both perspectives and make sense of the human and social issues surrounding the organisation. Therefore, in addition to the presented human and social issues emerging from business organisations, there are some aspects of software developers, and the software development methodologies that they use, which affect developers' work. From this professional experience I identify the following issues:

- Developers seemed to lack understanding of the human and social nature of decision-making and similar processes;
- For developers, the information from the interviews with users of complex organisations seemed difficult (if not impossible) to interpret and transform into software requirements and software within the available frameworks;
- It was apparent that developers lacked understanding of the nature and potential of software (in this case Business Intelligent tools) to aid decision-making and similar processes.

In summary, the University developers (myself included) showed a lack of understanding and awareness of human and social issues in organisations and did not know or were not able to get help from the development practices they knew. This thesis follows this concern and tries to investigate how developers make sense of the human and social aspects of complex organisations to develop software for them; and to identify the issues that affect developers' work (not necessarily acknowledged as problems by developers). This thesis considers that there are human and social factors that affect developers' work which are within their development environment and the organisation's users of software. Both human and social factors affect the work of developers and the (quality of the) software they produce. Thus the stand point of this thesis is that within the process of software development, the answer to the concern of making sense of complex organisations relies on an understanding of the human and social issues that surround software development. Hence the overarching issue is to build a model of software development which shows what and in which ways human and social issues affect developers.

1.2 Research Aim and Research Questions

Having presented the motivation and research problem for this thesis I will present the aim and research questions. In situations like those described in the previous section, the work of software developers is

affected by their lack of understanding of the human and social aspects of the organisations they develop the software for. These aspects, as shown in the model of complex organisations in figure 1.2, make organisations unpredictable, adaptable and multi-group work entities which do not match the mechanical and structured perspectives provided by traditional IS and software development methodologies. Although there are various attempts by IS researchers to incorporate human and social issues in Information Systems Development (ISD) methodologies, these only consider the aspects in the "system to be served" (Checkland and Holwell, 1997, 114) and do not consider the nature of the development environment at all . Examples of such ISD approaches are The Ethics Method by Mumford (1993), Soft Systems Methodology by Checkland and Holwell (1997), and Multiview by Avison and Wood-Harper (1990; 2003). (Section 2.2.3 of chapter 2, provides an exploration of ISD approaches.) Another attempt at considering human and social issues in organisations, specifically conflicts and competing views, is the Multiple Data Values method by Coakes and Coakes (2000). This method allows formal logic and data modelling to deal with the "mushiness" of organisations (or human and social issues). Though a very useful method for modelling organisations needs it does not consider the information systems or software developers' own perspectives and the human and social aspects within the software development environment. Other attempts to consider human and social issues within the development environment by software development methodologists do not seem to consider the human and social nature of the organisations which are going to use the software. For example the Agile Methodologies such as the Crystal Family pf methodologies by Cockburn (2000a) and Extreme Programming by Beck (1999) are people-centred (Fowler, 2004b). These methodologies are concerned with development practices which build upon the skills of developers and emphasise human and social issues resulting from the interaction between developers at work (more on Agile Methodologies in section 3.2.1 of chapter 3). Thus this thesis tries to bring both sides of software development, the development and the organisational environment, together and aims:

> To identify and explain how the social issues within the software development environment and the social aspects of complex organisations shape software and the process of software development.

As mentioned, this thesis considers that there are three aspects or contexts of software development in which social aspects need to be taken into account: the development and the organisational environment. Therefore I have designed the research questions to address each of them individually. The first context is concerned with the issues emerging from complex business organisations. The second and the third contexts are concerned with aspects of the development environment. In these latter two I have separated the aspects of individual developers and team groups, and the aspects of the development methods that they use. The reason is that the development methods in addition to providing guidelines that determine the way developers work (in the development environment), they also influence the way developers think of their target organisations and hence the human and social issues they might be aware of (which relates to the first context). The specific research questions are presented in table 1.1.

Context	Question
Social Issues in Complex Business Settings	Q1: What are the human and social aspects of business environments that make software development complex and difficult?
Social Issues in Development Environment	Q2: What are the social aspects in the development environment that allow or hinder the identification of social issues in business settings during the development of software?
Social Issues considered in Development Practices	Q3: What aspects of current software development methods address or neglect social issues in business settings? What is social software and does it have a role to play in improving these practices?

Table 1.1 Research Questions

Research Question 1 focuses on those human and social organisational aspects that are relevant to the information system and its software and thus need to be addressed by developers but which are difficult to understand. Research Question 2 focuses on the human and social

aspects within the development environment which shape the way developers work. This question is also concerned with how development practices deal with these human and social aspects within the development environment. Finally Research Question 3 is concerned with how the development practices (methodologies, techniques, etc) currently known and used by developers deal with human and social issues in organisations. A discussion of the research aim and research questions is presented in section 4.5 of chapter 5, which connects them with the research approach and design.

1.3 Research Approach

In order to address the above research questions, this thesis follows an interpretive approach to understand software development from the perspective of *information systems software developers.* This strategy is useful to understand how practitioners see and deal with complex situations and to give emphasis to their practical points of view. During the development of software, software developers build their own pictures of complex environments, through interviews with users, by referring to written documentation, and by implementing the software and testing it. In turn they themselves work also in complex organisations and are affected by their human and social issues. Software developers' vision of the human and social factors that affect their work is relevant and central to complement the theoretical and managerial perspectives in the information systems field.

Some groups of developers work online in virtual communities where they exchange working tips and socialise. In this instance virtual communities of software developers are real working environments in which public discussions are held (Rheingold, 1994, 5) and working experiences are shared. These online "communities of practice" are based around occupational communities whose contexts are "related to the subject matter around which they are formed" (Coakes and Clarke, 2006b, 92).

Software developers, members of online communities share "similar goals, work practices, beliefs, interests, and value systems" (Elliott and Scacchi, 2003, 22). Virtual communities assume different forms such as online forums, weblogs (online personal journals) and wikis (collaboratively-edited websites). In addition, developers also use more private channels like e-mails and instant messaging to enhance their communication effectiveness. Virtual communities of software developers are further explored in section 4.2 of chapter 4 of this thesis. That exploration includes linguistic and pragmatic issues which are relevant to the data collection and data analysis of this study.

Data for this study was gathered following an online ethnographic method (i.e. an ethnography of virtual communities of software developers, see section 4.3 of chapter 4). Ethnography is not a new practice in the field of information systems research. It has proved useful at providing insights into the human and social aspects of information systems and organisations (Harvey and Myers, 1995, 22). The online nature of the ethnography helps to engage a wide variety of developers from different countries, with different professional backgrounds and working on different projects. It can do this since online ethnography does not have financial and time constraints and geographical boundaries (Folkman Curasi, 2001, 367; Parrish, 2002, 3; Sweet, 2001, 2).

The data analysis (presented in section 4.4 of chapter 4) used two techniques: content analysis and conversation analysis. Content analysis was used to classify data in categories or segments of text that "reflect the purposes of the research" (Holsti, 1969, 95) and to look for associations between them. Conversation analysis helped to make sense of the structure of online conversations as they follow different rules from written and oral speech. The aim of the data analysis process was theory building (explained in section 4.6 of chapter 4). Concepts and propositions were extracted from data categories and abstracted from their original contexts (e.g. the online conversations). As these concepts were at a higher level of abstraction than the original developers'

statements, they may stand for other similar pieces of data (Tesch, 1990, 138), i.e. they "may be valuable in the future to explain other organizations and contexts" (Walsham, 1995, 79). It was a process of interpretive generalisation whose validity depended on "the plausibility and cogency of the logical reasoning used in describing the results from the cases and drawing the conclusions from them" (Walsham, 1993, 15). (See chapters 5 to 7 for a description of this thesis' results.) Concepts and propositions were related to one or more research questions and further refined in the contexts of those questions. By relating them to the research questions, the abstracted concepts and propositions were transformed in the research answers. Thus they show what the nature of the human and social issues within the different aspects of software development is. An example of this process is shown in Appendix 3: Analysis and Theory Building. In the final stage of the theory building process the research answers are shaped into a theory based model of social issues in software development. The model connects the concepts and propositions (i.e. the research answers), in a way in which, they explain how these issues affect software developers' work. The model presents this in the form of a "flow" which shows the order in which human and social issues are identified. The process of creation of the model of social issues in software development is discussed in detail throughout chapters 5 to 7. Having presented the research problem and design, in the next section the structure of the whole thesis is explained.

1.4 Structure of the Thesis

This thesis comprises 8 chapters organised in 3 parts. Part I presents an overview of a software development project which triggered my interest in human and social issues. From this experience I extract a set of problems which at that time hindered the work of developers. I then generalise these issues and create a model which summarises the nature of this software development situation. From this model I draw the research problem, aim and questions which in the rest of this thesis I intend to

develop further. Part II presents a literature review that focuses on concepts that appear to be relevant for this thesis such as soft definitions of organisations and information systems, human and social issues in software, software development and the software development environment. Additionally, this part presents the outline of the interpretive, online ethnographic approach of this thesis. Finally part III presents the findings of this thesis in the form of a theory based model which identifies the human and social issues that surround software development and shows how these aspects affect developers' work. This part of the thesis explains the whole process of data analysis and generalisation from data collection to theory generation.

A brief summary of the content of each chapter is provided below:

Part I Introduction

Chapter 1 provides the reader with a context of the research problem, questions and methodology of this thesis.

Part II Context of the research contains the literature review and research design which explain what and how this study is done.

Chapter 2 discusses the literature on organisations and information systems. As organisations "are the context in which IS work is done" (Checkland and Holwell, 1997, 75) the IS field is very deeply grounded in organisational theories. This chapter outlines a theoretical background for software development which is discussed in chapter 3.

Chapter 3 discusses the human and social aspects of software and software development and explores the nature of software developers and development teams. The discussion focuses on the human and social aspects such as adaptability and the

unpredictability of organisations and information systems examined in chapter 2.

Chapter 4 describes the research method. It starts by introducing and justifying the interpretive approach and the online ethnographic method. It also reviews the literature of virtual communities and describes the characteristics of online communities of software developers in which this study was done. This chapter also describes the data analysis and theory building strategies used to mould the findings into a theory based model.

Part III Findings and Conclusions describes and explains the process of data analysis and theory building, and presents the conclusions of this thesis.

Chapter 5 describes the practicalities of the online ethnographic fieldwork. It first explains my interactions with online members and how I started to create themes from those interactions. It then explains the analysis phase of the ethnography, focusing on how I created categories of data and abstracted concepts from those categories.

Chapter 6 presents the findings from the online ethnography organised by data categories. This chapter connects the data categorisation process explained in chapter 5 with the theory based model developed in chapter 7. The findings presented here correspond to my interpretations of software developers' interpretations of human and social issues in their work.

Chapter 7 presents a model of social issues in software development developed from the data presented in chapter 6. This model is the result of the interpretive theory building strategy of data analysis. This chapter introduces with a theoretical framework of information systems and software development which helps to

situate this model. It then discusses the human and social issues covered by the model.

Chapter 8 presents the conclusions of this thesis. It first presents a summary of the findings of this thesis. It then provides a discussion of the research contributions to knowledge. This is followed by a reflection on the research based on Klein and Myers (1999) principles for evaluating interpretive research. This chapter concludes with a discussion of opportunities for future research.

1.5 Chapter Summary

This chapter has introduced the research problem and design of this thesis. This thesis studies the human and social issues of software development environments and complex business organisations whose inadequate identification or treatment can affect the work of software developers. This chapter presented a professional experience which is the motivation of this thesis. I related a situation in which the developers failed to understand the human and social complexities of an organisation. The source of the problem was the disagreement between the structured preconception of organisations that the developers had and the reality of the unpredictable, adaptable and multi-group business organisations. I then connect this problem to the developers' lack of sensitivity to human and social issues within their own development environment and the software development methodologies they use. This thesis believes that in order to make sense of complex organisations it is necessary for developers to be aware of and understand the human and complex issues within their environment and the organisations they develop the software for. Thus the aim of this thesis is to investigate which these issues are and how they affect developers' work. This investigation is carried out using an online ethnographic method of virtual communities of software developers. The data collected is analysed by using the content and conversation analysis techniques. Data is

categorised, abstracted and generalised into concepts and propositions. The outcome of the data analysis, and of this thesis, is a theory based model which addresses human and social issues that surrounds software and software development and explains how these issues affect developers' work. The name of the model is "social issues in software development" and is presented in figure 7.5 of chapter 7 of this thesis.

To start this thesis chapter 2 will describe in detail some organisational and information systems perspectives which emphasise their human and social aspects. This chapter sets the basis for the discussion about software and software development in chapter 3.

PART II: CONTEXT OF THE RESEARCH

Part II of this thesis presents a review of the human and social aspects of organisations and information systems – the context in which software development is done – and then centres its attention to human and social issues in Software Development and related topics such as software and software developers. Also a description and justification of the research methodology presented at the end of this part.

Chapter 2: Information Systems in the context of complex organisations

Having explained the research problem and motivations in part I of this thesis, the first two chapters of part II will explore the literature on organisations, information systems and software development. As the main concern of this thesis is the study the human and social aspects of software development for complex organisations, the first part of this chapter will explore the nature and complexities of (business) organisations. Complex organisations are places where decision-makers and users of software work and interact; they also provide a context for the existence of information systems. Information systems are groups of processes that manage information and which *can be* supported by business software. As business software are designed to fit the needs of particular information systems, the second part of this chapter will explore some definitions of information systems and approaches to information systems development (ISD). Information systems development is an activity that designs and implements organisational information systems which (usually but not necessarily) *include* the development of software. Chapter 3 of this thesis continues the literature review by exploring how human and social aspects are reflected in software and in the processes of software development.

Figure 2.1 shows a context of software development as understood by this thesis. This context helps to explain the relationship between software and information systems development. Figure 2.1 shows that business software are used to support information systems processes within business organisations. Information systems development (the big white arrow in figure 2.1) are carried out to design and implement organisational information systems. Information systems development focuses on organisational and people's objectives at performing their activities. In addition, information systems development is usually concerned with analysing the organisational environment, its processes,

its people and their interactions. Hence the white arrow pointing at an information system within an organisation.

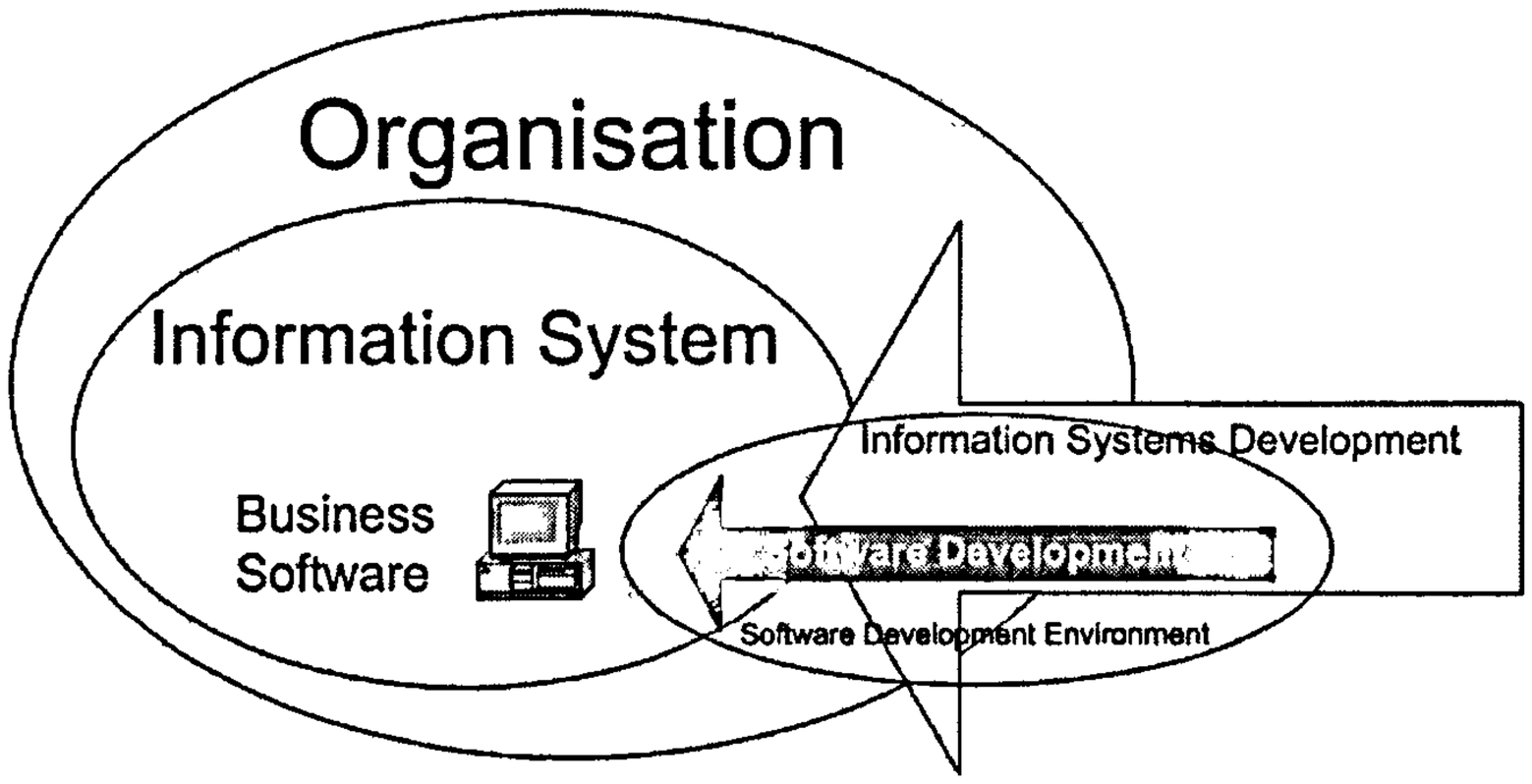


Figure 2.1 The Context of Software Development

Information systems development usually (but not necessarily) involves the development of software. The grey arrow in figure 2.1 shows Software Development as an activity which is part of ISD but which pays special attention to software to support information systems processes. Because this thesis sees software development as a part of information systems development, the literature review discussed in this chapter includes organisations, information systems and information systems development and sets them as the context in which software development happens.

Following the contexts shown in figure 2.1, chapter 2 will review literature on organisations and information systems and chapter 3 will cover literature on software and software development. The reason for this division is the different focus of these fields. While chapter 2 concentrates on the nature of organisations, the people that work there and the processes that run through organisations, chapter 3 will concentrate on the more technical subjects of software and software development. The aim of chapter 2 is to explore different views on organisations which consider human and social issues. This will help to develop the concept of Complex Organisations further. The focus of this chapter is on

concepts that appear to be relevant to the context of this thesis (as explained in section 1.1 of chapter 1: research motivations). This chapter is not an exploration of research in information systems or software development. The literature that covers human and social issues in both topics (IS and software development) together is scarce. Rather this chapter will connect these views on organisations with different perspectives on information systems (IS) and information systems development (ISD). As organisations "are the context in which IS work is done" (Checkland and Holwell, 1997, 75) the IS field is very deeply grounded on theories about organisations. Additionally, this chapter will outline a theoretical background for software and software development which will be covered in the next chapter. Similarly, because business organisations and their information systems are the context in which business software is used and in which business software development happens, software and software development are also deeply grounded on organisations and IS theories.

This chapter is divided in two sections. The first one explores various perspectives which address organisational issues such as adaptability, multi-group work and unpredictability in organisations such as the ones mentioned in the previous chapter. The second section defines and discusses different views of information systems and information systems development. This second section situates information systems development within the context of complex organisations and builds the grounds for the discussion in the next chapter.

2.1 Understanding Complex Organisations

In chapter 1, I presented a problematic situation in which I had to design and develop a decision support application for the University of Lima. One of the problems I identified was the lack of understanding that my colleagues and I had about organisational human and social issues. Human and social issues are problems or situations which are caused by

people, their behaviour, their interactions and beliefs, and which affect their work and the work of others. My colleagues and I shared a mechanistic view of the organisation and followed development methodologies based on this view which ignored these human and social issues. The organisation was expected by us to demonstrate stable and predictable behaviour; however, it appeared chaotic to us. We found social issues such as unpredictability and adaptability of human action, free social interactions between software users, and complicated interactions between workgroups difficult to understand and model into software. In an attempt to understand these problems and comprehend the essence of organisations, this chapter section will examine social views of organisations in the literature. In the next subsection 2.1.1 I will explore some perspectives about organisations which explain how the human and social aspects of organisations define their behaviour. Each perspective presented here explains one or more characteristics of complex organisations as presented in chapter 1 of this thesis. Subsection 2.1.2 will then explore some concepts from Giddens' structuration theory. Structuration is a social theory which connects human agency with social structures. For Giddens, human agency, which is situated at a micro level, explains the macro level phenomena. It is thus suitable to explain how human agency within organisations and the software development environment create social structures which then are (should be) reflected in software.

2.1.1 Defining Organisations

In this section I will present various definitions of organisations which emphasise their adaptable, unpredictable and multi-group aspects as opposite to structured and mechanical organisational views. For the purpose of this discussion I will classify the organisational definitions into two groups: the hard and the soft perspectives. These two approaches have different views of organisations and as a consequence they have contributed to the development of the also sometimes conflicting

standpoints of IS. Hard perspectives of organisations are based on positivist epistemologies. They have characterised the natural sciences for many centuries, and have influenced and continue to influence the social sciences in a similar way. Hard approaches to social sciences seek "to explain and predict what happens in the social world [(e.g. organisations)] by searching for regularities and causal relationships between its constituent elements" (Burrell and Morgan, 1979, 5). Hard perspectives define organisations in a very mechanistic fashion. For example Lucas (1994) describes an organisation as "a rational coordination of activities of a group of people for the purpose of achieving some goal" (Lucas, 1994, 78). Laudon and Laudon (2005) state that traditional organisations are "a hierarchical, centralized, structured arrangement of specialists" (Laudon and Laudon, 2005, 6) whose work is based on "standard operating procedures" (Laudon and Laudon, 2005, 7). Perspectives like these portray organisations as stable entities in which people work in a predictable way, according to predefined structures, to achieve a goal. These perspectives differ very much from the model of complex organisations shown in chapter 1, which present organisations as unpredictable and adaptable entities. Unfortunately these hard perspectives guide most of the content of IS. However these descriptions do not consider the complexities of human action and social interactions which are not predictable and predefined.

On the other hand, the soft approaches to organisations are influenced by anti-positivistic epistemologies which are concerned with human behaviour and the "meanings that people give to their environment" (May, 1997, 13). One of the most interesting definitions of an organisation is this one of Checkland and Holwell (1997): an organisation is a "reified social collectivity" (Checkland and Holwell, 1997, 80). According to them, the existence of an organisation depends on the awareness and recognition of its members and non-members. Therefore informal organisations could exist alongside formal ones as long as their members recognise their existence. Organisations do not have physical location or shape but the one given by people. People within the organisations make sense of their

purposes and decide on the activities that are needed to pursue their goals. People outside the organisations can identify the organisational boundaries and have a perception of the purposes and activities performed within the organisation. This means that only individuals are the ones who make sense of the different purposes and activities within the organisations. Checkland and Holwell (1997) state that

> "all real world management problem situations have at least one thing in common: they contain people interested in trying to take purposeful action" (Checkland and Holwell, 1997, 13).

Every action that is performed in an organisation has a purpose. However not all people's purposes necessarily concur with the official goals of the organisation but perhaps with different personal or group objectives. Within a social collectivity more than one purpose could exist. In fact, organisation members would interact in pursuit of a range of interests (Dunford (1992); Martin et al (1991) quoted in Warne (2003, 107)). People working in organisations could have their own hidden agendas, such as making more money, improvements of particular skills and making contacts, etc. This perspective contrasts with the mechanistic ones which restrict organisation's activities merely to the ones that are performed to achieve one goal. They neglect the idea of having more kinds of activities and multiple purposes within the organisation. The concept of social collectivity can be used to describe some aspects of the model of complex organisations. Each group of people who is part of a decision-making process can be described as a social collectivity which existence lasts as long as the purpose of the decision-making process. The members of each collectivity can recognise who is in and out of their groups and can reconfigure their working arrangements according to the needs of their jobs.

Charles Handy offers three interesting descriptions of modern organisations which also present them as "social collectivities" (as defined by Checkland and Holwell (1997)). The first account is the Shamrock Organisation. Resembling the three leaves of a shamrock,

organisations, according to Handy, comprise three groups of workers. The first group or leaf is the "professional core". The professional core is comprised by qualified professionals, managers and technicians "who are essential to the organisation" (Handy, 1989, 72). The core decides on the goals of the organisation and guides the activities of the other two groups. Workers in the professional core are indispensable to the organisation and cannot be replaced easily. The second leaf of the shamrock is made up of people who are "contracted out" because they "are not crucial to the organisation" (Handy, 1989, 73). These people are the ones who actually do the work which is designed by the professional core. The third leaf is "the flexible labour force, all those part-time workers and temporary workers who are the fastest growing part of the employment scene" (Handy, 1989, 74). This description of organisations emphasises that each group should be managed, paid and organised differently (Handy, 1989, 72). However, according to my experience, and as shown in the model of complex organisations (see chapter 1), work, e.g., information systems and business processes, could cross the different leaves of the shamrock in the form of (working or social) interactions between groups of people. The Shamrock Organisation is a useful concept to identify the different groups of people that are part of an organisation but does not emphasise the connections that could exist between them.

Handy's second complementary perspective of organisation is the federation or federal organisation. A federation is "a variety of individual groups allied together under a common flag with some shared identity" (Handy, 1989, 93). A federal organisation encourages autonomy of its members but within bound. It combines "variety and shared purpose, individuality and partnership" (Handy, 1995, 33). Different groups of people within the organisation are "bonded to themselves as well as to the centre" (Handy, 1995, 11) This is a decentralised system in which the centre does not direct but co-ordinates the activities and makes sure that the work is being done. Every entity within the federation, individual or group, has the power and ability to decide on how to perform its tasks. Flow of information between groups of people and between groups of

people and the centre are needed for coordination and keep the organisation working. Contrary to the Shamrock Organisation, the concept of the Federal Organisation emphasises the working connections between the groups and the groups and the centre; and the freedom of action within the groups and among them. These are aspects that are reflected in the model of complex organisations.

A third complementary perspective of Handy on organisations is the Intelligent Organisation. An intelligent organisation is comprised by brains, information and ideas. Brains are clever people but "on their own are not enough" to add value to the organisation (Handy, 1985, 112). Brains need good information and ideas to be able to add value to the organisation. "Intelligent organisations have to be run by persuasion and by consent" (Handy, 1985, 131). This is because their members have the information and the expertise to fight for their ideas at the same level as their managers do. Drucker (1998) refers to them as *knowledge workers* and like volunteers they have to be motivated by challenges. The only way to maximise their performance is "by capitalizing on their strengths and their knowledge rather than trying to force them into molds" (Drucker, 1998). Brains or Knowledge Workers are useful concepts to explain the freedom, unpredictability and adaptability of action in the model of complex organisations. As (groups of) decision-makers can discern what to do in particular circumstances, the actions they undertake dictate the shape of the processes they are working in. Therefore the start, shape, or end of a decision-making process is uncertain. Similarly, as decision-makers have the power to alter a decision to accommodate it to new circumstances the decision-making processes become highly adaptable.

Another account of organisations is the organisation as a "nexus of decisions". Stohr and Konsynski (1992) define decisions and decision making as the true essence of the organisation. For them the organisation's purpose is to make decisions within a business environment (an aspect in common with the model of complex organisations). Organisations are defined by their decision opportunities,

authorities and opportunities (Marakas, 2003, 88). This perspective is independent from the physical manifestation of the members of the organisation who could be part of complex networks of individuals made up of team groups and computerised decision support systems (Marakas, 2003, 88). This view of organisations emphasises the human and social aspects of decision-making processes, group work, and working and social interactions within the organisation. All of these aspects are sources of unpredictability and adaptability in organisations which are also highlighted in the model of complex organisations.

Other organisational perspectives exist which could explain different aspects of the model of complex organisations. In this section I have presented the ones I believe are the most relevant as they address the main aspects of the model of complex organisations. Three other interesting perspectives are discussed in appendix 9 of this thesis which contribute at a lesser extend with explaining the nature of complex organisations. The first perspective is Knowledge Management which focuses on how knowledge is created, stored and utilised in organisations, like for example in decision-making processes. The second perspective is Complexity Theory which tries to explain the (apparently) chaotic nature of human systems like for example business organisations. Finally, the third perspective, Actor-Network Theory, tries to model social environments, such as organisations, as networks of actors which are connected by their interactions.

The model of complex organisations presented in chapter 1 could be explained in terms of the previous accounts of organisations. The concept of social collectivity of Checkland and Holwell (1997) refers to groups of people involved in decision-making processes. Informal organisations are created as the information flows and as the group meetings are being run. Their existence could be brief but is meaningful for their members as they perform purposeful actions. Similar to the model of complex organisations, the concept of social collectivity also does not imply any kind of activity or standard procedure. It does not imply either a

predefined structure or hierarchy of any type. Social structures are created on the run from people's actions. These structures change and evolve through time as the need of people and their environment change. The shamrock organisation metaphor shows how decision-making processes may cross through the leaves of the shamrock regardless of the kind of people who is working there. Despite the fact that the professional core is the one responsible for the direction of the organisation they do not do all the work. Much of the stages in decision-making processes that have to deal with gathering information, verifying facts, arranging meetings, etc., might be carried out by people in the other leaves of the shamrock. The concept of federal organisation adds the ideas of independency of individuals and groups at making decisions. The core of the federation guides people but does not control them. People and groups are free to decide how to act or what to do in a certain situation. The views about intelligent organisations and knowledge workers show how people in an organisation have their own responsibilities and are experts in their field of action, and therefore should have the freedom to act as they decide. However, when they are committed to a decision-making process (organisation as a nexus of decisions) they respond also to a common purpose. Coordination and agreement between individuals is done by motivation, challenges and peer discussion. The importance of these perspectives of organisations to this thesis is that each of them emphasise (though only partially) the organisations' human and social aspects which make them behave in an unpredictable and adaptable manner. These accounts of organisations help to enrich the model of complex organisations, as presented in chapter 1, and to highlight the importance of human and social aspects in organisations. This is especially important in the fields of information systems and software development which are guided by mechanistic perspectives. As mentioned in the introduction of this section, having mechanistic views of organisations that behave unpredictable and are adaptable because of their human and social aspects can hinder the job of a software developer.

Having presented several views of organisations in the next section a discussion of Giddens' Structuration Theory is presented. Structuration theory is a social theory which concentrates on how human agency (micro level) explains social structures (macro level). Because structuration is a social theory it can explain how organisational structures emerge from human action in organisations. Structuration theory is a useful tool that complements the model of complex organisations by providing a connection between the macro and micro level perspectives. Being a complex theory situated at an ontological level of knowledge whose concepts need deep explanation more space is dedicated to this theory. Also structuration theory plays an important role in the moulding of the findings of this thesis (see chapter 7).

2.1.2 Giddens' Structuration Theory

Giddens' structuration theory "is a general theory of social organisation and has a primarily ontological focus." (Jones et al., 2004, 298). It is an emergent, process theory (Jones, 1999, 130-131; Orlikowski and Robey, 1991, 258) which incorporates the concept of "duality of structure" that explains the nature of human action in society. Structuration theory is a useful tool to describe and explain the nature of organizations and social interactions. In this section I will explore some structurational concepts which are tightly related to the concepts of adaptation, change, flexibility and multi-group work presented in the model of complex organisations. These structurational concepts present a different perspective at a micro level which explains the macro level phenomena.

According to Giddens, social systems and organizations possess emergent characteristics. This means that their structures are not predetermined or inherent but are a result of a process of evolution through time and space as these systems interact and adapt to their environments (du Plooy, 2003, 45). For Giddens human action and structure presuppose one another (Giddens, 1979, 53). Thus Giddens

defines structure and human agency as a duality in which both are mutually dependant and recursively related aspects of social systems (Rose and Scheepers, 2001, 1,3; Rose and Hackney, 2003, 259). By the duality of structure Giddens means "that social structures are both constituted by human agency, and yet at the same time are the very medium of its constitution" (Giddens, 1976, 121). The duality of structure solves the philosophical dilemma between positivist phenomenological views of society. Positivist standpoints see structures as institutional and objective properties of society (Rose and Hackney, 2003, 258) whereas interpretivist perspectives see society as conformed by meaningful actions. Before structuration theory appeared, the social sciences field was divided by these two apparently mutually exclusive perspectives. Giddens has developed structuration theory to accommodate these two traditions under one frame so as to offer a more complete explanation of the nature of social reality. However, he had to redefine the concepts of structures and agency to account for their mutual dependency (Giddens, 1979, 53).

Structuration is the process whereby the duality of structure evolves and is reproduced over time and space (Rose, 1998, 4). Structure is both, the medium and the outcome of the process of structuration (Brooks, 1997, 137). For Giddens, structure is the rules and resources implicated in social reproduction (Giddens, 1979, 64). Structure does not exist on its own but comes to life through human activity and interaction (Walsham and Han, 1991, 3; Rose and Scheepers, 2001). This means that it exists only in the memory of people and that it is reified only when people use or follow the society's rules and resources to act and interact. Therefore structure has only "virtual existence" (Giddens, 1979, 63). It is an abstract and intangible rather than objective property of social systems. Similarly, human agency is social action reproducing existing structures or producing new ones (Walsham and Han, 1991, 78). Action or agency does not refer to a series of discrete acts combined together, but to a continuous flow of conduct. The notion of action refers also to the characteristics of the actor (Giddens, 1979, 55). The nature of agency is

also a recursive one as it constitutes and is constituted by structure (Orlikowski and Robey, 1991, 147).

Giddens defines three modalities of the process of structuration through which the realm of social action is linked to the institutional realm (Jones et al., 2004, 302; Orlikowski and Robey, 1991, 148). All these three modalities are not mutually exclusive aspects of structuration but are interrelated. A schema showing the relationship between the modalities, agency and structure is presented in figure 2.2. The first modality is the "Interpretive Scheme" which represents the stock of knowledge drawn on people's interpretations of behaviour and events that is used to convey and understand meaning; that is to make communication possible. The second modality is represented by the "Facilities" (or resources) used by people to accomplish their desired outcomes. Facilities are enacted in the "wielding of power" (Rose and Scheepers, 2001, 4) and produce structures of domination showing asymmetrical distribution of resources (Walsham and Han, 1991, 84). There are two kinds of resources, Allocative Facilities which are material resources derived from human dominance over nature, and Authoritative Resources, which are the non-material capacities of harnessing human action (Rose and Scheepers, 2001, 3). Finally, the last modality is the "Norms" which dictate whether conduct is appropriate or not according to moral codes within a social collectivity (Checkland and Holwell, 1997). Conducts are legitimized (i.e. made appropriate) by the use of sanctions during interaction. These three modalities explain how people make sense of the social structures and act according to them, and how social structures emerge from these actions.

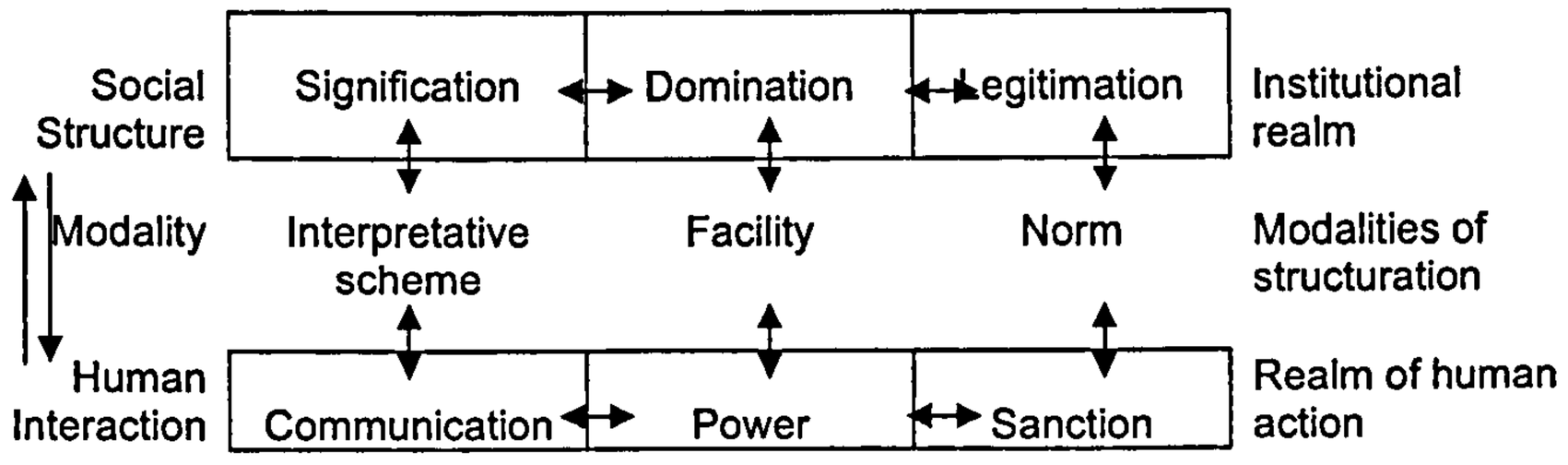


Figure 2.2 The Modalities of Structuration
Adapted from Giddens (1979, 82)

Giddens' theory also provides an in-depth explanation of how people act. He says that within the realm of action, people's actions are guided by their knowledge of norms and facilities that are available to them. Every person accumulates knowledge from their past experiences which possess aspects represented by the modalities of structuration. People learn how to communicate with people, how to exercise power and how to act according to the norms of the social system they are in. According to Giddens there are two levels in which people can use the knowledge they possess. First, people can use their knowledge at the level of practical consciousness which reflects the things that people "know how to do" (Giddens, 1979, 73). Thus practical consciousness is the actor's ability to act according to his stock of personal knowledge (Orlikowski, 1992, 8) but which the actor is not able to formulate discursively (Giddens, 1979, 57). The second level, discursive consciousness, is the actor's ability to explicitly articulate his actions and motivations in terms of his knowledge (Giddens, 1979, 73; Walsham and Han, 1991, 78). Discursive consciousness is a more advanced form of consciousness which also implies practical consciousness. To achieve a level of discursive consciousness people have to reflect on their actions and act in a reflexive way. Reflexivity is the capacity of routinely observing, analysing and understanding one's actions. When knowledgeable and reflexive actors improve their interactions and get the results they expect they perform these actions more consistently and they become more stable. Giddens calls this phenomenon routinization of human actions. Routines are "habitual, taken-for-granted" activities (Giddens, 1984, 376).

They form patterns of interactions which become institutionalized, that is they become part of the structure of the social system in which they are interacting (Orlikowski, 1992, 9). This is the process of structuration of social practices which constitute both individuals and society. Routinisation of social practices is only a temporal event as social structures continue to change and evolve due to changes in their environments. Therefore "social change" happens when new practices replace others. As mentioned, structuration is a process of evolution. When the knowledgeable and reflexive human consciously adapts to new situations, social practices evolve into new ones which are then spread through mechanisms of social and system integration. Structuration processes could therefore give rise to seemingly complex and chaotic situations.

For Giddens, social practices and hence social structures are spread through two different channels. Social practices can be spread through social integration and system integration. Social integration happens in situations of co-presence when actors interact face-to-face and has a cohesive effect at determining social practice. System integration happens through distant interactions and have wider effects at replicating social practices (Rose and Scheepers, 2001, 4). One example of system integration is human interactions and the spreading of rules and resources through the internet. Another example of this is the use of telephones and mobile phones and the creation of rules which guide communication through these media. When the mechanisms of social and system integration stretch to spread social practices geographically and over time a phenomenon called "time space distanciation" occurs. Time space distanciation refers to the separation of time and space which in traditional societies are linked through place (Walsham, 1998, 1085). Time space distanciation is a phenomenon that occurs in modern societies and shapes modern life. Giddens explains this with an example about the invention of the clock. In pre-modern societies, people associated "when" (time) with "where" (space), for example people knew what time it was from the position of the sun and the size of the shadow.

With the invention of the clock and its diffusion the dimensions of time and space were separated. The clock expressed "a uniform dimension of "empty" time" (Giddens, 1990, 17) and therefore time did not depend of space. Also, in modern societies, the dimensions of place and space have been disconnected. Place is associated with the idea of locale: geographically situated, physical settings of social activity. In pre-modern societies place was always associated with space since they both implied physical presence. However, the invention of the internet now has turned "place" a "phantasmagoric" (Giddens, 1990, 18) concept, different from space as for example virtual communication is available.

Structuration theory provides a perspective that could inform information systems and software developers to understand complex organisations. The structurational concepts explored in this section are connected to the concepts mentioned in the model of complex organisations and play an important role at explaining complex organisational situations. The concept of structuration implies adaptation of social structures as social systems interact and adapt to their environment. Although social practices can become routinized this is only a temporal state as practices will always change to adapt to changes in their environments. At a macro level, structuration explains how social structures emerge within organisations from human action. These social structures are then observed by software developers who will then transform them into code. At a micro level structuration explains how individuals act reflexively based on their knowledge. Finally, the issue of multi-group work is related to the concepts of social and systems integration. Social practices are spread through different groups of people which then share similar social structures and are able to communicate and work together.

As it has been presented, Giddens' structuration theory does not pay attention to information systems or technology. Many authors had argued though that structuration has the potential to explain social phenomena in the use of technology (Walsham and Han, 1991). Although Giddens' himself neglected this particular area, there are other researchers who

have followed the structuration path and studied information systems and technology (DeSanctis and Poole, 1994; Orlikowski, 1992; Orlikowski, 2000; Rose and Hackney, 2003). The presented section had the purpose to introduce structuration concepts in this thesis which help to enrich the model of complex organisations but which are also the bases to the work on information systems and technology presented in section 2.2.4 of this chapter.

In the next section I will explore some topics within the field of information systems and information systems development. Section 2.2 will narrow the scope that section 2.1 had. Rather than seeing organisations as a whole, section 2.2 will only address the organisational processes known as information systems. This will lead the reader to the discussion of software and software development presented in chapter 3.

2.2 The Information Systems field

Information Systems (IS) is a relatively new field (less than one century old) which emerged after the construction of the first computers. At that time, when computers were first coming out in the 1940s and 1950s and until very recently, practitioners and researchers have tended "to focus on the technical aspects" (Warne, 2003, 107), such as technology, computer sciences and software programming. However, changes in business environments posed new challenges to information systems specialists. The recognition of information as an asset with a "critical role in contemporary organisations" (Laudon and Laudon, 2000, 14) has made IS grow into a "multidisciplinary field" in which "no single theory or perspective dominates" (Laudon and Laudon, 2005, 17). Disciplines such as "management and organisation theory, sociology, systems thinking, political science, social psychology, etc" (Checkland and Holwell, 1997, 9) in addition to information technology contribute to the study of IS. Although the multi-perspective approach to IS helps to enhance and enrich the field it also contributes to make it complicated (researchers and

practitioners need to deal with the increasing diverse information). Thus there is a need in this thesis to clarify my perspective in this field. In the next sections I will first explore some definitions of Information Systems and then concentrate on Information Systems Development as this is the context in which software development for complex organisations (the topic of this thesis) happens.

2.2.1 Information Systems

The field of IS addresses the management of information, the creation and implementation of information systems, and their utilization to benefit the organisations. Many definitions exist of information systems (Fitzgerald et al., 2002, 4). The introduction of different approaches to the field has contributed to enrich the definitions of IS, though they have also brought more confusion. Different definitions of IS generate dissimilar conceptualisations of the field. As there are lots of definitions of information systems it is important to cite the ones which reflect the interpretive, phenomenological ontology of this thesis. I start by mentioning some conventional definitions of information systems, such as the following, which emphasise a technical perspective and then continue with more interpretive definitions:

> "systems involved in the gathering processing, distribution and use of information." (Beynon-Davies, 2002, 4)

The next definition shows a functionalist perspective:

> "An information system provides procedures to record and make available information, concerning part of an organisation, to assist organisation-related activities." (Flynn, 1998, 3)

However, following the soft approaches to organisations presented in the previous section, the next definitions emphasise their social aspects. These definitions outline the field of action and give an idea of the

perspective that will be used in this thesis. The following definition is a bit technical but takes into account the participation of people:

> "A system which assembles, stores, processes and delivers information relevant to an organisation (or to society), in such a way that the information is accessible and useful to those who wish to use it, including managers, staff, clients and citizens" (Buckingham et al (1987) quoted in Avison and Fitzgerald (2003, 19)).

The next definition does not even consider computers and software:

> "...the nature of an information system is that it is a function supporting people taking purposeful action..." (Checkland and Holwell, 1997, 110).

And finally this definition from Handy sets out the role of information systems within an organisation:

> "The nerves of the organisation without which none of the systems would function; serves the three above [adaptive, operating and maintenance systems], running through them and around them" (Handy, 1985, 337).

Having presented various definitions of information systems it is important now to state the approach of this thesis. These four definitions show three important aspects of information systems. First, the role of people and their objectives at using organisation's information, second the role of the information systems at serving people purposes and supporting organisations' existence and finally the computer software which represents the technical face of information systems. As a factor that influences organisation's members, it is thought that information systems "often institutionalise a certain way of work" and "enforce standardised procedures" (du Plooy, 2003, 43) especially if the use of inflexible software is involved. In modern Information Systems software is the actual means through which information is stored, distributed and made available to users. Therefore the development of software is very much connected with the development of information systems. In fact, nowadays, in most cases, the development of information systems is thought of as the development of software and vice versa.

2.2.2 Information Systems Development

Information Systems Development (ISD) is at the core of the IS field (Fitzgerald et al., 2002). This topic is also addressed from different perspectives. As shown in the previous section, this thesis' standpoint emphasises social aspects rather than the technical ones. However this thesis' focus is on software development. Thus I subscribe to the following definition that relates organisation needs with software:

> "ISD is often conceptualised as two mutually dependent processes: analysing an organizational (social) situation, and designing and implementing computerized information support for it" (Rose and Scheepers, 2001, 2).

Complementing the above definition, Rose and Scheepers (2001) state that information systems developers have to be aware of the human and social issues within organisations. Human and social issues are those kinds of problems or situations caused by people, by their behaviour, their interactions, by the way they are organised and by their beliefs which affect their work and the work of others. In the context of software development human and social aspects are the characteristics of people developing or using software, the interactions between them and the effect these aspects have on software. Because they affect people, these issues need to be recognised and addressed in order to improve the quality of work and the products or services provided. Software will therefore be built to support social interactions between users. Handy (1985) thinks developing information systems is a complex task. For him the designer has to consider four elements: "the purpose of the information, the person for whom it is intended, the nature of the technology and the type of operation" (Handy, 1985, 384).

Checkland (1985) adds to all the mentioned definitions by stating that ISD is a "form of enquiry" with three components: "an intellectual framework, a methodology, and an application area" (quoted in Avison and Fitzgerald (1995, 420)). The intellectual framework is the "philosophy that guides

and constrains the enquiry" (Avison and Fitzgerald, 1995, 420). Here we have concepts and beliefs about the nature of organisations and information systems and how information systems development should be done. The methodology arises from the transformation of the framework into tangible plans and instructions which then are applied in the real world or application area. The three components of ISD are shown in figure 2.3. Figure 2.3 is useful to explain the area of concern of this thesis. The present thesis is focused on investigating how software developers perceive their jobs, in particular the human and social aspects. That is, this thesis explores the software developers' beliefs about human and social issues in their jobs: the organisations they develop the software for (the application area) and the methodologies they use to develop software. Thus this thesis looks at the intellectual framework that software developers possess and it does so by exploring the developers' perspectives on the other two components of ISD.

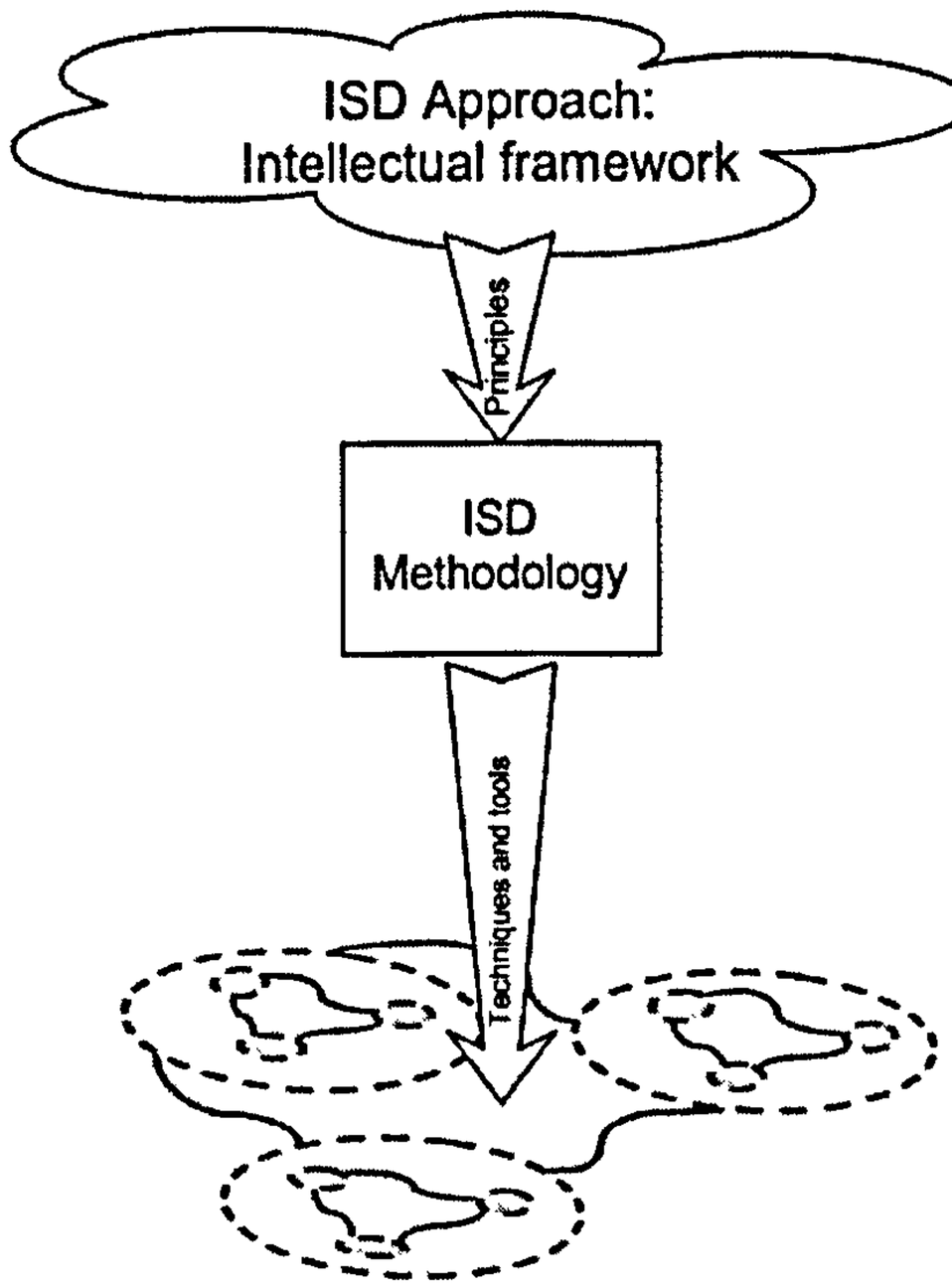


Figure 2.3 Components of ISD

The following section will look at ISD intellectual frameworks or ISD approaches as explained in the model in figure 2.3. The purpose of this exposition is to present the current perspectives that exist in the IS literature some of which are reflected in the data collected in this thesis (shown all through chapter 6, but especially in categories C1-8 and C6). This exposition will help the reader to compare what is said in the IS research field (as presented in the following section) with what is actually believed and done by software development practitioners (as presented in chapter 6). ISD methodologies, the second component of ISD, will be explored in the next chapter after a discussion of software and software types. As the concern of this thesis is software development I will concentrate only on software oriented ISD methodologies guided by ISD Software approaches (as opposed to methodologies that focus only on the analysis stages of the development process and do not address the development of software at all).

2.2.3 ISD Approaches

The intellectual framework for information systems development represents the philosophical assumptions that guide the process of development. In this thesis I refer to those intellectual frameworks as ISD approaches or ideas or actions intended to deal with an information system development situation. To present the variety of approaches that exist I have classified them according to themes or schools of thought (I followed Avison and Fitzgerald (2003) classification of approaches to ISD which are inclusive enough to accommodate a huge range of perspectives and themes). This classification contains seven groups of approaches which are not necessarily mutually exclusive. Each group has different views of organisations and information systems which guide differently the development of information systems. Each approach that I have included in this classification has been chosen either because of its past or present influence in software development practice or because of its awareness of human and social issues as it is the case of the people

approaches or because of its potential for the future development as they account for new trends in the technology such as the web approaches.

The groups of approaches are presented in table 2.1. A more detailed discussion of ISD approaches belonging to each group is presented in appendix 10.

Approaches	**Relevant issues for the Model of complex organisations.**
Organisational	Offers a holistic view of the organisation or group of organisations and focus on aligning the ISD process and the IS with other organisational processes.
Modelling	Concerned with creating models of the problem situation. These models are intermediate steps between the intangible ideas of the problem and concrete design specifications.
People/Participative	Seek the involvement of designers alongside users from different areas of the organisation to bring more insight into a problem situation.
Engineering	See organisations and information systems as stable and predictable. Emphasise the use of methods which are measured for their performance and productivity.
External development	The development effort is taken away from the organisation either by buying software or hiring a third party to develop the software for them.
Software	Focused on the development of computer software. Look for the improvement of quality software.
Web approaches	Do not predefine their activities. Responsive to every day situations.

Table 2.1 ISD Approaches

The above list of approaches is by no means an exhaustive compilation of approaches but an overview at the spectrum of perspectives that exist. These cover four important issues in ISD: the understanding of the

organisation and the problem situation, the modelling of the situation and the information system to be built, the development of software and the participation of users and designers within the process. These perspectives are not mutually exclusive. In fact some approaches overlap others. In section 3.2.1 of chapter 3 I present a discussion of some software development methodologies which are not necessarily fully connected to any of these approaches (except prototyping). That is, these methodologies do not totally represent the principles presented in the above sections. The methodologies shown in chapter 3 are the ones more used or commented on by software developers during the fieldwork. It is important to note here that the disconnection between the ISD approaches, popular in the IS literature, and the software development methodologies used in practice represent a gap between research (or theory) and practice which this thesis also wants to highlight.

In the next section I will explore technology, information systems and information systems development from a structurational perspective. Structuration is used here to explain how social structures emerge from the use of technology and especially software. The concepts explored in the next section will be then referred to in chapter 7 to help mould the findings of this thesis and to provide a theoretical framework for the final result.

2.2.4 Structuration theory in the Information Systems field

In this section I will present two applications of the structurational perspective in the IT and IS fields. The first one is the model of structuration of IT by Orlikowsky (1992; 2000) and the second one is a structurational interpretation of the Information Systems and Information Systems Development fields by Rose and Scheepers (2001) and Rose and Hackney (2003). While there are a few more attempts to use structuration theory in IS research, Orlikowski and Rose's attempts are the more relevant to this thesis. Orlikowski's perspective is useful to

understand the nature of complex organisations that use technology tools and Rose's perspective presents information systems (and software) development as a structuration phenomenon and situates it within a structurational organisational setting. Two other interesting attempts to use the structurational perspective in IS research are the use of structurational concepts in the evolution of Multiview 2, a framework for information systems development, by Avison et al. (1998) and the use of structuration theory to understand the social and organisational issues that surround the use of collaborative tools by Evans and Brooks (2005). These two attempts however are focused on specific instances of information systems: an IS methodology and collaborative tools. As this thesis possesses a wider focus these studies will not be explored more in depth.

a. Structuration theory of Information Technology

Structuration theory can provide insights into the nature of technology and information systems. Orlikowski's (1992) "duality of information and technology" is one of the best known attempts to use structuration theory to explain the relationship between organizations and technology. Orlikowski (1992) uses Giddens' structuration theory to build a theoretical framework that "explores how technology is created, used and becomes institutionalized within an organization" (Brooks, 1997, 137). Figure 2.4 shows Orlikowski's model of structuration of IT. According to Orlikowski (1992) information technology constitutes and is constituted by human agency. That is, IT is "shaped by and shapes human action" (Jones et al., 2004, 318). On the one hand information technology is an objective set of rules enabling or constraining human action and contributing to the creation of social practices involved with the use of IT (arrow b, Figure 2.4). On the other hand information technology is the social product of human action within specific structural and cultural contexts (arrow a, figure 2.4) (Orlikowski and Robey, 1991, 151). Once the use of technology is stable it becomes routinized and appears as part of the social structures of the organization (or institutional properties of the

organisation) that influence human action (arrow c, figure 2.4). However when technology is changed to the extent that it does not become institutionalized, the consequences of interaction with technology will influence the organizational context (arrow d, figure 2.4) (Brooks, 1997, 138).

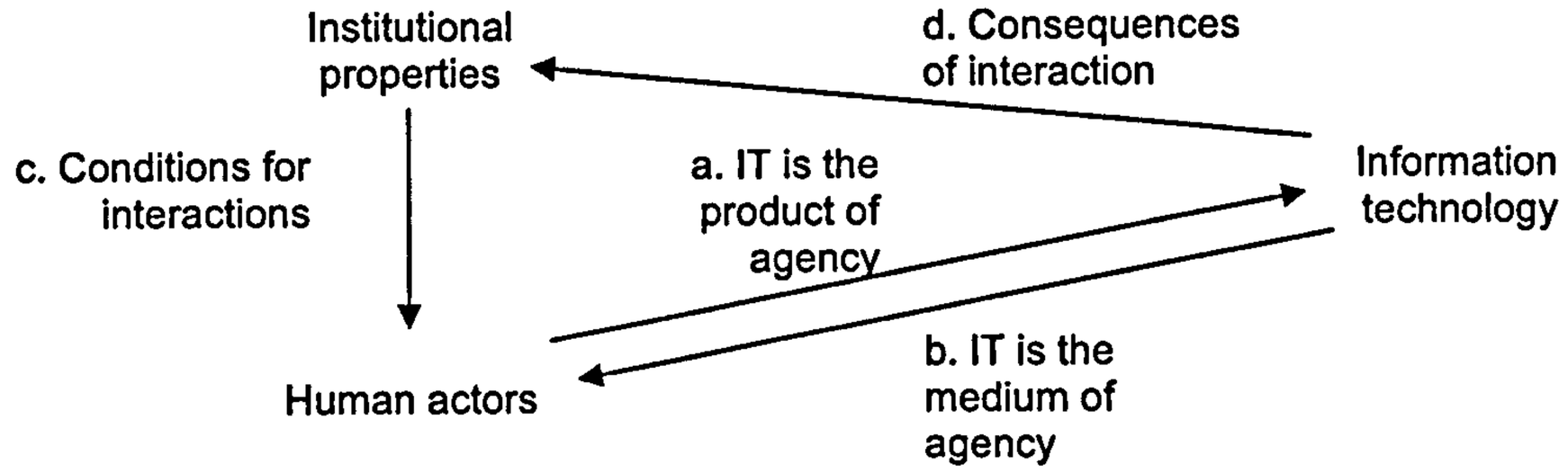


Figure 2.4 Structurational Model of IT
Source: Orlikowski and Robey (1991, 17)

The role of IT as a medium of agency (arrow b, figure 2.4) is further developed by Rose and Hackney (2003). According to Rose and Hackney information technology can enable and stabilize interactions or constrain them. The role of enabling and stabilizing interactions is the aim of technology designers. However the idea that IT constrains interactions may be caused by the difficulties in effecting changes to it. The nature of the constraint could also be an inherent characteristic of technologies. These inherent characteristics may be seen as a result of wider social processes or absolute features of technology (Rose and Hackney, 2003, 264).

Orlikowski also explores the temporal dimension of structuration. She sees technology as having two iterative modes (Orlikowski, 1992, 408). First she identifies the "Design Mode" which represents human action affecting technology, that is, the realm of human action of structuration. Second, she recognises the "Use Mode" representing technology affecting human action, that is, the institutional realm. As information technology is developed in a different setting and at a different time from the one in which it is used, one of the modes of the duality usually

becomes obscured (Brooks, 1997, 138) to the people involved with the other mode. For example, researchers who investigate ISD may see technology as a subjective phenomenon conformed by the actions and interactions of developers which at the same time are shaped by the organizational goals. These researchers may also neglect the factors affecting technology use in the future. Researchers who investigate use of technology may see technology as artefacts influencing users' behaviour and may not consider the factors affecting its development. For Orlikowski (1992) this is a misleading disjuncture (Orlikowski, 1992, 409). Rather than treating design and use as disconnected modes of IT, Orlikowski's structurational model suggests that artefacts could be modified through their existence, that is, technology can be changed by users through their interactions with it. Orlikowski calls this the interpretive flexibility of technology. However, despite the opportunities for change, routinized views of technology and interactions often develop. These routinized views are caused by the interactions between organizations and technology and are not inherent in the nature of technology (Brooks, 1997, 139). In the case of software development, as I see it, this means that software developers use a static snapshot of the organisation to do their job, neglecting the possibility of change. Although developers know that a big deal of their time is devoted to maintenance work (to account for changes) they do not consider this as part of the development job. Maintenance is usually disliked by developers and taken as a second class job (fieldwork data, see Maintainability in C3-2 in section 6.3 of chapter 6). This thesis' standpoint concurs with Orlikowski's position. Both modes of technology should be connected. This thesis studies the "design mode" but it is the observation of the "use mode" through time by developers which this thesis pays especial attention. I refer to this as "developers being aware of human and social issues in the organisational environment".

According to Jones (1997), the weakness of Orlikowski's structurational model of IT is that her conceptualization of technology as a material product of human action does not concord with Giddens account of

technology (Rose and Hackney, 2003, 258). For Giddens "technology in so much as it exists, casts it as a resource to be employed by human action" (Rose and Hackney, 2003, 258). Structure only exists in the memory of people (Jones, 1999, 109) and agency is a human attribute, therefore there is no place for a materialistic account of technology like the one proposed by Orlikowski.

In a later work Orlikowski (2000) proposed an extension of her structurational perspective which overcame (as she states) the criticisms mentioned above. Orlikowski (2000) separates two aspects of technology: technology as "artefact" and "technology use"; and further defines the latter as technology-in-practice. She then proposes a practice lens to understand the recursive interaction between people, technology and social action.

Orlikowski (2000) states that, seen through practice lens, structures are not embodied in technology-in-practice but emerge from them (Orlikowski, 2000, 407). Rather than examining technology and its properties, the practice lens looks at human action and examines how it enacts structures through interaction with technology. Orlikowski develops the concepts of emergent structures and enactment of structures which define technology-use as a social structure. For Orlikowski emergent structures are structures that come to life from human action. This view contradicts the old perspective which saw structures as embodied or inscribed in technology. Also by seeing social structures as enacted, Orlikowski rejects the idea of technology or structure appropriation. Appropriation of technology is a concept brought up by DeSanctis and Poole (1994) which shows how a given rule or resource within technology is brought into action (DeSanctis and Poole, 1994, 129). Thus technology appropriation sees people as always using technology as it was designed. However this is not always the case. Users of technology can translate into action social structures not predicted by technology designers. For example, users can invent new ways of using technology, they can ignore certain properties of

technology, or they can make mistakes and use technology wrongly (e.g. select the wrong options to get wrong results). Therefore, different social structures emerge from technology use which can correspond or not with the intentions of technology designers. This point is relevant to this thesis. Software developers should be aware that the software that they develop may be used in different ways (e.g. users subverting the system) and that the users may want the software to be adapted to other uses different than the ones originally planned. The changes in requirements and attitudes by the users may be explained by the concepts of emerging structures and translation into action explained above (see fieldwork data, category C3-2 adaptability in chapter 6).

The second model of Orlikowski's is called "Enactment of Technologies-in-Practice" and is shown in figure 2.5. It is based on Gidden's duality of structure. Orlikowski explains that users of technology are situated "within a number of nested and overlapping social systems" (Orlikowski, 2000, 411). Therefore the use of technology will recreate other social structures apart from technology-in-practice. Similarly, users will draw on many varied social structures when they use technology. These factors make the use of technology an adaptable and unpredictable phenomenon which influences the way information systems professional see organisations.

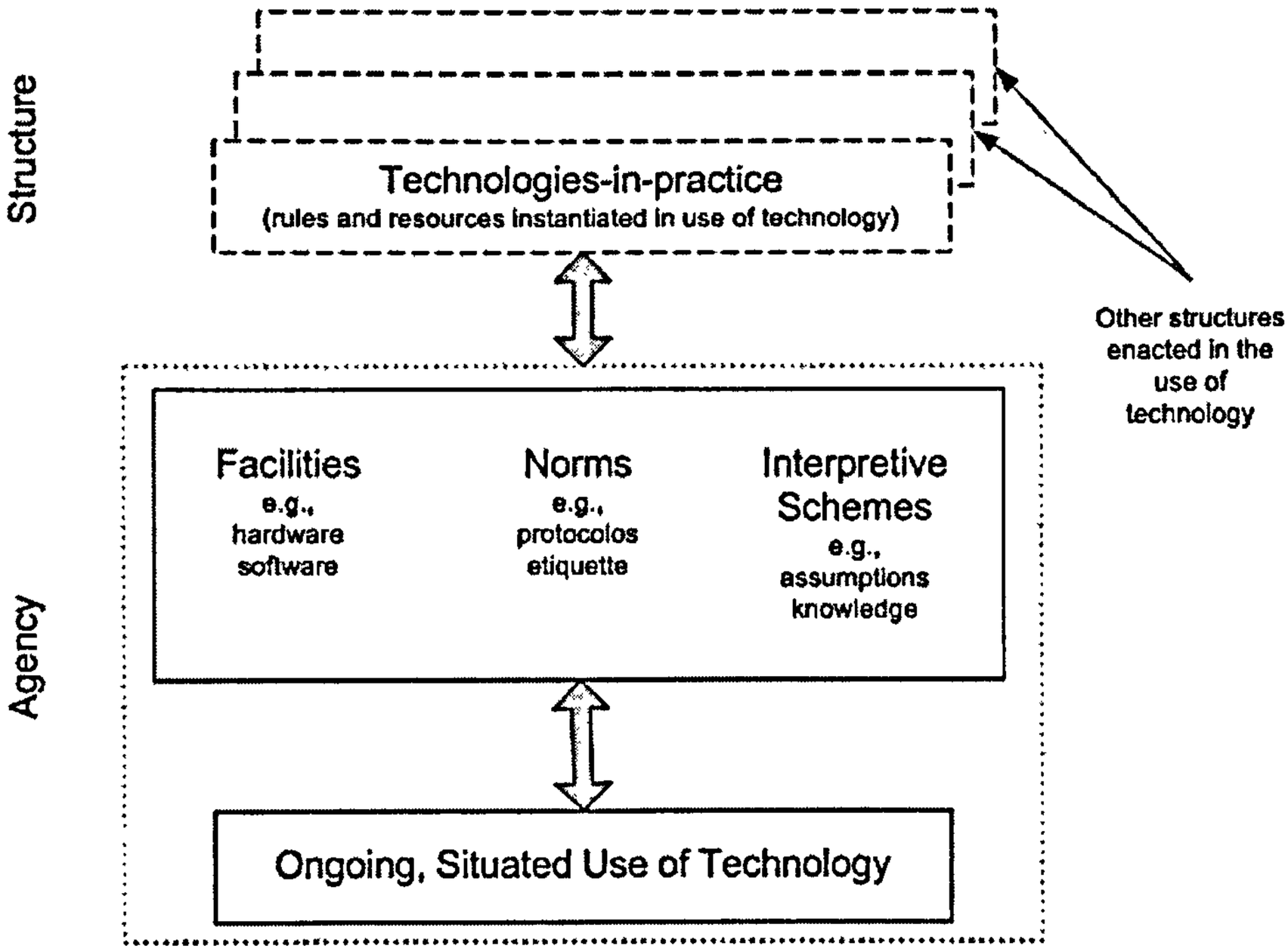


Figure 2.5 Enactment of Technologies-in-Practice
Source: Orlikowski (2000, 410)

Over time, constant use of technology may become routinized and may be seen as predetermined (by users and information systems designers). Routinized use of technology may impede change as people would think of technology as a fixed and stable object. However, this view is only provisional (Orlikowski, 2000, 411) as technology can be modified by its designers (e.g. new versions of a software could be released) or users can innovate their use of technology. Thus, technologies are never fully stabilised or "complete" even when people choose to treat them as fixed objects for a period of time (Orlikowski, 2000, 411).

In addition to explaining the relationship between technology and use of technology, Orlikowski's practice lens can be used to analyse organisational situations where people work and interact with IT tools. For example, Schultze and Orlikowski (2004) use a practice lens perspective to study the use of technology mediated network relations. The practice lens help them to "highlight how macrolevel phenomena such as interfirm

relations are created and recreated through the microlevel actions taken by firm members" (Schultze and Orlikowski, 2004, 87).

The structurational perspective of IT of Orlikowski (1992; 2000) presents a view of IT which can also be applied to business software and software development. We can separate the software artefact from software use. Software use can also be seen as a recursive phenomenon. The use of software creates and recreates social structures within organisations which are (hopefully) perceived by software developers. Users draw on diverse social structures to use software and social structures emerge from that use. Use of software changes and evolves over time. Thus change of software can happen during design time (realm of human action) and during software use (institutional realm). This perspective corresponds to the one shown in the model of complex organisations where people, and especially software users, present adaptable and unpredictable behaviour. Because of this, Orlikowski's model of enactment of technologies-in-practice is used in chapter 7. Orlikowski's model forms part of a theoretical framework (which I designed) in which the findings of this thesis are situated. The enactment of technologies-in-practice provides a view of software use that is continuous in time and that generates social structures which are relevant for the development and/or maintenance of software.

In the next section I will present another structurational perspective which adapts structuration theory to ISD in an attempt to bridge the gap left by Giddens because of his disinterest in practical applications of his work.

b. Structuration theory and Information Systems

Rose (2000) and Rose and Scheepers (2001) describe Information Systems (IS) and the process of Information Systems Development (ISD) in terms of structurational concepts in an attempt to influence IS research and practice. For them Information systems are subsets of social practices that are determined by a task (Rose and Scheepers, 2001, 9).

Information systems development is a "specialized social practice, subject to its own evolving interactions and structures, and highly partial and fallible." (Rose and Hackney, 2003, 265) Figure 2.6 presents a diagram showing the relevant sets of social systems (organisation, business system, information system, software and information systems development) that surround IS. For Rose and Scheepers (2001) Information systems have three components:

1. A social system devoted to collecting, storing and disseminating relevant information (could be the organisation, the business system or the information system)
2. Computer and communication technologies where information practice may be heavily enmeshed with use of the technologies. As technology is composed of material artefacts it cannot be described as social systems.
3. A social system devoted to the development maintenance and management of the technology (ISD).

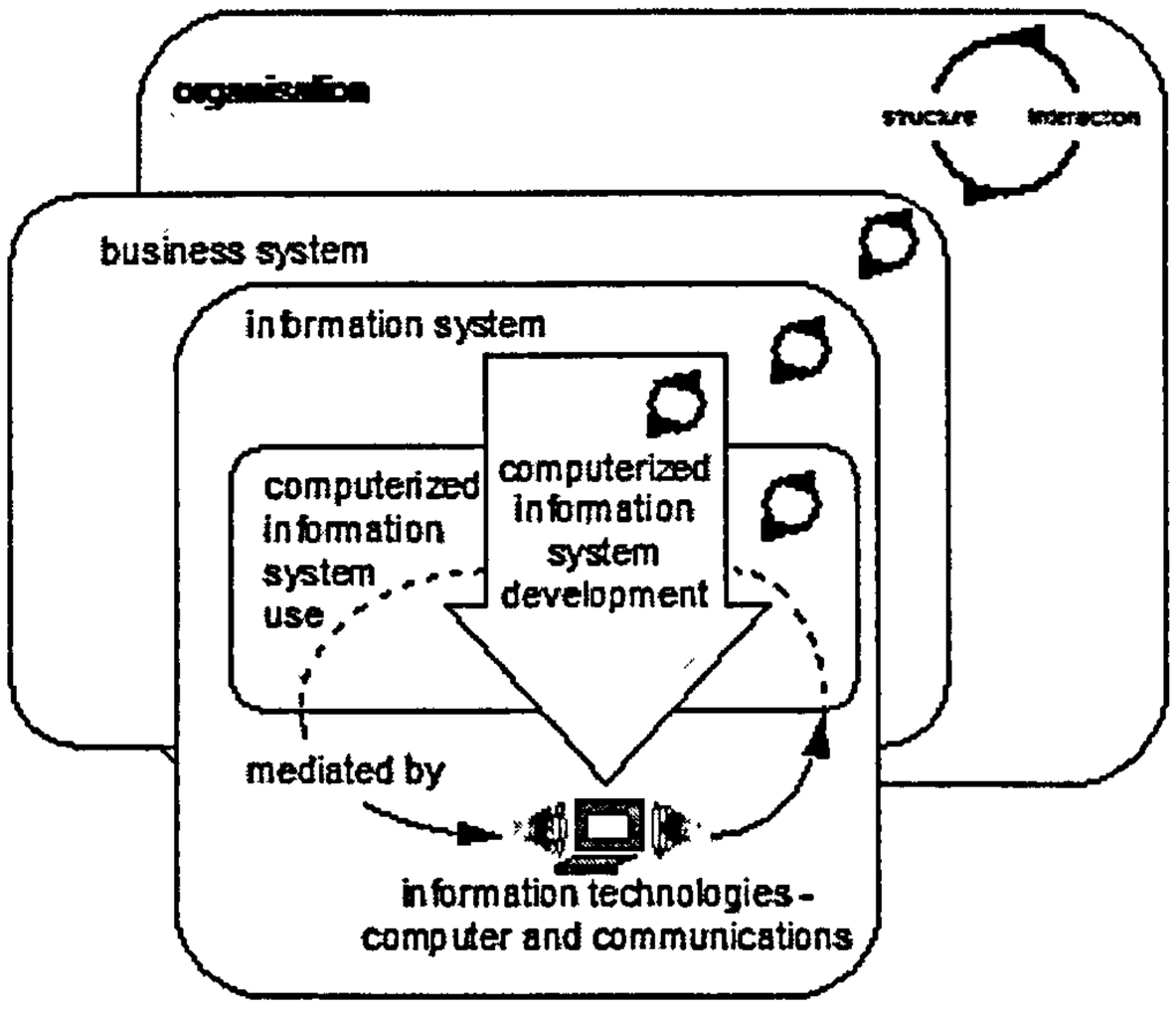


Figure 2.6 Social Systems around IS
Source: Rose (2000, 150) and Rose and Scheepers (2001, 10)

Within the context in figure 2.6, Rose and Scheepers (2001) refer to the role of discourse to explain the relationship between structure and agency

in IS. Discourse is characterised as information and is the medium of structuration of information systems (Rose and Scheepers, 2001, 9) as it is used by knowledgeable and reflexive actors to institutionalize their practices. In the institutional realm, information systems represent the structure of the social system which designs and uses it. In the realm of human action information systems also routinize many practices through the mechanisms of system integration and time-space distanciation (Rose and Scheepers, 2001, 10). Translating this into the software development field, Rose and Scheepers portray information systems as social systems which are supported by material resources such as software, which are designed by another social system (software developers). Software designed by developers reflects the structures of the social system that built it and that one that uses it. Following this idea this thesis tries to bring together both sides of software development: the development and the organisational environments (this is stated in section 1.2 of chapter 1). Also software may routinize social practices through the mechanisms of system integration and time-space distanciation. Finally, a point worth noting here is that Rose and Scheepers suggest the use of IT to leverage social practices which are already well stabilized or institutionalized.

2.3 Chapter Summary

In this chapter I have discussed several views of organisations and information systems, especially those ones which emphasise their adaptable, unpredictable, and multi-group work aspects (as shown in the model of complex organisations presented in chapter 1). I have paid especial attention to structurational perspectives of organisations and information systems which focus on the social phenomena at a micro level but which can also be extrapolated to a macro level by demonstrating the relationship between human agency and institutional properties (Orlikowski and Robey, 1991, 154). The reason for choosing structuration is because it can explain how organisational structures

emerge from human action in organisations especially through the use of software. This is important to this thesis as it is concerned with the developers' awareness of human and social issues in organisations users of software and their ability to translate those issues into software. Additionally, this chapter has described modern organisations and their information systems as unpredictable and ever changing. Business processes can cross organisational boundaries and involve different stakeholders. In doing their jobs, people may have their own personal purposes apart from the organisational ones. These and other human and social aspects of organisations shape the context in which business software is used and software development happens. Thus this chapter has set a background for the discussion about software and software development in the next chapter. In the next chapter I will explore software in the context of information systems, software development and some issues concerning software developers.

Chapter 3: Business Software and Software Development

Having discussed complex organisations and information systems in chapter 2, in this chapter I will review the concepts of software, software development and explore the characteristics of software developers. In chapter 2 I noted that business organisations and their information systems are the context in which business software is used and software development is carried out. Figure 3.1 (repeated from figure 2.1 in chapter 2) positions software development as an activity within information systems development (shown by grey arrow). Information systems development focuses on analysing the organisational environment and designing their processes by fulfilling their organisational and people's objectives. Within information systems development, software development is aligned with the same organisational aims but through the implementation of a software tool.

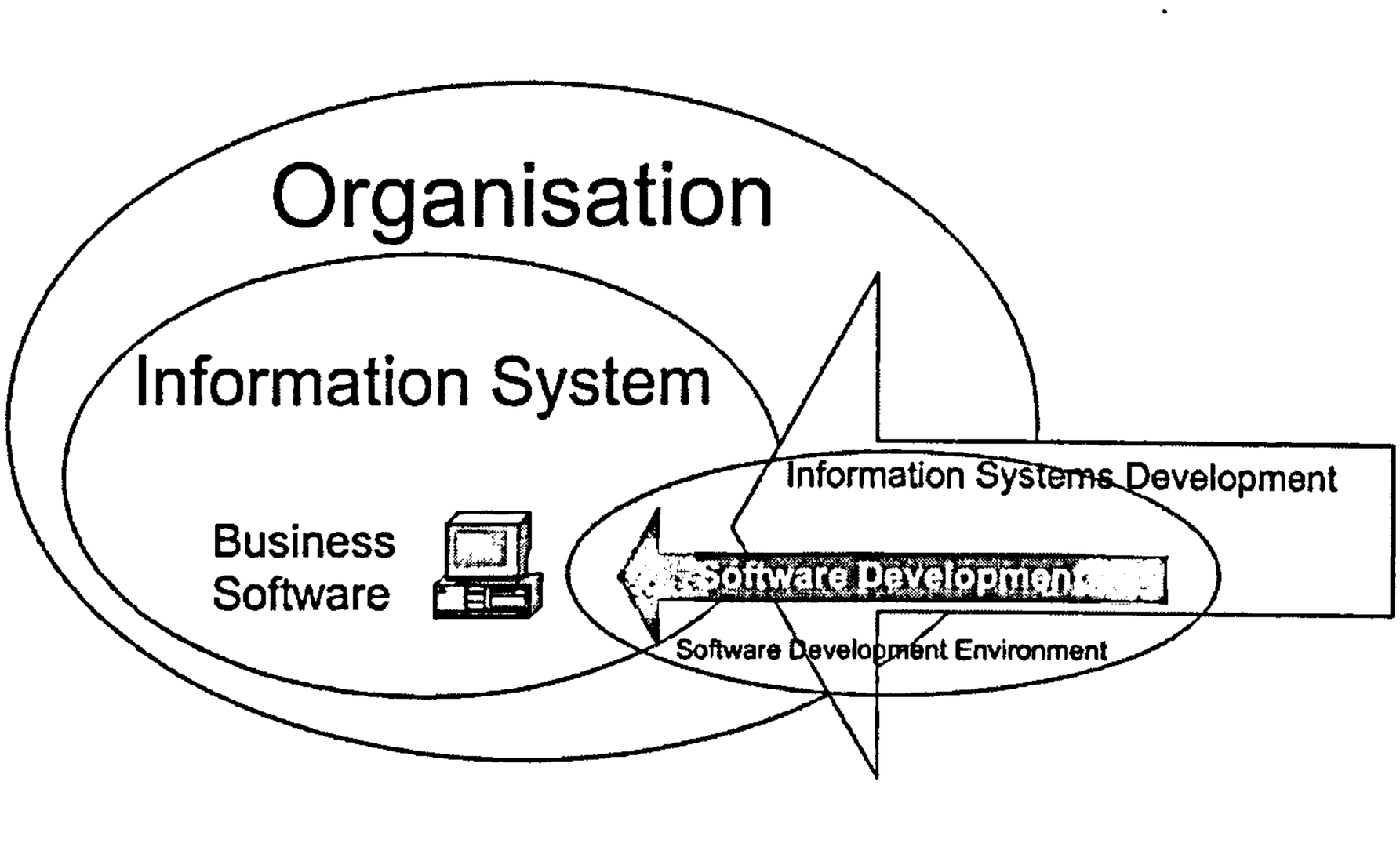


Figure 3.1 The Context of Software Development

In this chapter I will introduce the software development environment as another context which influences software development work. Development practices, working arrangements and software developers'

beliefs affect the way they see organisations, their users and the software they need to develop. This chapter will also deal with the concept of software as it is the product of the software development process and the one that business organisations will use. I will explore how human and social issues, from both the organisational and the software development environments shape software. Although software and software development are more technical topics than the ones discussed in the previous chapter, I will focus on the human and social aspects surrounding these concepts so as to complement the views presented in chapter 2.

The objective of this chapter is to compare the nature of (business) software and software development with the human systems they are serving (the organisation and its information systems). The purpose of this is to see how social issues within business organisations are reflected in software development and hence in the software that results, and to explore any other social issues emerging from the development environment. Human systems were discussed in chapter 2 as complex organisations which contain adaptable, unpredictable, multi-group processes. However, the software that organisations' members use is seen as predictable pieces of code by software developers. Though code is capable of being changed (adapted), it usually does so at a slower pace than the human systems they serve. At the same time, the environments in which software is developed resemble the ones of organisations in the sense that they contain unpredictable, adaptable and multi-group processes. The human and social factors in software development, though usually neglected in the past, are gaining more importance as new human-oriented development methodologies are being created (see section 1.2 of chapter 1). Thus I will explore how the increase in awareness of the human and social factors within the development environment could help developers to be more aware of the human and social factors within the business setting they are developing the software for.

This chapter is divided in two sections. The first section defines software, and business software in particular, and their characteristics and presents a classification of software which is relevant to the kinds of business processes within complex organisations as presented in chapter 1. The second section explores the characteristics of software development and discusses some of the most well known (and most used in practice) software development methodologies and explores the characteristics of software developers focusing on their role in the development process as individuals and as team members.

3.1 Software

Software was introduced in chapter 2 (section 2.2.1) as the technical face of information systems, which supports people working and sustains organisations' existence. In practice software is the visible outcome of the implementation of an information system. New rules and improved processes are learnt and applied or executed through computer tools. Thus, for software developers and users, software is a more tangible (and hence comprehensible) concept than information systems. Before deepening the discussion on software for information systems (or business software) I will provide an opening definition and classification. This will help the reader to appreciate the particular features of the type of software (and therefore software development processes) that this thesis is focused on.

From a technical or engineering perspective software is the set of commands or instructions used to control a computer to accomplish particular tasks. Software is a unique kind of product, it cannot be compared to other engineering products (Mann, 2002, 36). Software is an intangible product made up of bits and bytes. Only when one uses it does one understand the value (usefulness, look and feel) of its features (Fowler, 2004b). Likewise, Spolsky (2002b) compares software with an iceberg in which only 10% of it is visible to the users (e.g. the graphical

interface) and the 90% is under the water (e.g. formulas, algorithms, and connections with database).

Software is a broad concept as understood by software developers. Software could be games, or word processors or payroll systems. Software can be embedded in hardware, (e.g. calculators), exist online on the web, or sold on CDs. In fact, within the software field there are different software perspectives which differ in their views depending on the purpose and technical characteristics of software they deal with. Technically speaking, Spolsky (2002a) identifies five "worlds" within the software field which are not necessarily mutually exclusive.
Spolsky's classification is done from a "business of software" perspective and is based on the technical aspects of software and its development. He classifies software as:

1. Shrinkwrap: or off-the-shelf software which is designed to be used by a large number of people. Software developed in this way is not made to measure but contains generic features which can be used in more than one setting. Difficulties with shrinkwrap are that it has to reach very high levels of usability to achieve success among thousands of users. Also, development of this type of software becomes complicated because this software may be run in many different kinds of computers. Developers have to make sure their shrinkwrap software has similar behaviour among all those platforms. An example of a shrinkwrap software is a word processor.
2. Internal: is software designed to work for only one kind of situation in only one setting. Because internal software is going to be used by a limited number of people and be run in a fixed and known technology platform its development may be seen as less complicated. However, according to Cockburn (1995), customisation of software to particular tastes of users and settings makes its complexity grow exponentially (Cockburn, 1995).

3. Embedded: is software that is installed in a piece of hardware (not normally identified as a computer) and usually cannot be updated. For instance software for digital cameras. Complications for embedded software are the limited amount of memory available and that there are no second chances for improvements as no changes can be done once installed.
4. Games: are software developed to hit one particular kind of market. Market analysis is done to discover the preferences of people and the features they are willing to see in games. A problem with many games is that once users play it to the end, they will not want to play it again. Developers are therefore forced to be faster at creating, developing and releasing new games. Games have the same quality requirements as embedded software and a huge "financial imperative to get it right the first time" (Spolsky, 2002a).
5. Throwaway: is software created temporarily, exclusively for internal development purposes. This software acts as a bridge to obtain a further objective. For example, a routine to change the format of data from one database standard to another.

This classification, though technical, provides a view of where software for business organisations could be situated. In business organisations, the context in which this thesis is located, software is also known by other names such as business software, business application or business solution. Business software is any computer application that helps to run the organisations and which is aligned with their information systems, objectives and strategies. Typical examples are accounting and payroll systems. From the above classification, business software can be internal, and be designed to fit one specific organisation. Business software can also be shrinkwrapped. Good examples of shringwrapped business software are business solutions such as PeopleSoft Enterprise (owned by Oracle ®) or mySAP ERP (owned by SAP), though most of these applications require further development to be adjusted to the

particular needs of the organisations. This thesis is particularly interested in the development of business applications which are customised to fit the unpredictable, adaptable, flexible and multi-organisational nature of business settings (as shown in the model of complex organisations) either if software are shrinkwrapped or internal. Therefore, in either situations (shrinkwrapped or internal) attention to the (kind of) organisational settings in which the software is intended to work is needed.

McConnell (1993) identifies some external and internal characteristics of software, which I am featuring in table 3.1. For McConnell, external characteristics are the ones users and developers care about; and internal characteristics are the ones (mostly only) developers are aware of.

External	Internal
Correctness: the extent to which the software is designed and implemented according to what users and developers have in mind (Cockburn, 1995) Usability: the ease with which users can learn and use a software Efficiency: minimal use of technical resources. Software takes advantage of the problem and machine in which it is to be run (Weinberg, 1971, 21) Reliability: having few failures and performing as required Integrity: the degree to which the software present data accurately Accuracy: how well a system does the job it is supposed to do (regardless of its correctness) Adaptability: the degree to which the software can be used in different ways than the originally specified.	Maintainability: the ease with which programme code [instructions in a computer programme or software] can be modified (corrections or improvements) Flexibility: the ease with which programme code can be modified to other uses other than the originally specified Portability: the ease with which software can be moved to a different platform Reusability: the extent to which a programmer can use a piece of programme code in another programme Readability: the ease with which programme code can be read and understood at the technical level Testability: the degree to which the software can be assed to find out if it meets the requirements Understandability: the ease with which the software can be comprehended at the system-organisational and statement levels.

Table 3.1 Characteristics of Software
Adapted from McConnell (1993, 557-558)

Another characteristic of business software, explored by Cooper and Reimann (2003) is "fudgeability". Fudgeability is the ability of software to let users "perform actions out of sequence or before requisites are satisfied" (Cooper and Reimann, 2003, 189), that is, the software does not compel humans to endure inflexible business processes. For Cooper and Reimann (2003) fudgeability is a success factor because fudgeable software allows flexible human processes to flow naturally. For example, good software is one that lets users perform a transaction that will allow a million pound deal to be signed even if the deadline has expired.

There are further external factors that affect software's development and behaviour. Business Software is usually developed under pressure. If it is shrinkwrapped, the pressure comes from the market; if it is internal, the pressure comes from the stakeholders or users. Cockburn (2001) states that "business and technology worlds have become turbulent, high speed, and uncertain, requiring a process to both create, change and respond rapidly to change." (Cockburn, 2001) For this reason, in the software business, a "just acceptable product developed quickly will be preferred to an excellent product produced slowly." (Chisholm et al., 1996, 232)

Is software only a technical concept?

The nature of software, or what it involves, is not only a technical set of aspects. Software also contains human and social aspects resulting from its development and use. By human and social aspects I mean the characteristics of people developing or using software, the interactions between them and the effect these aspects have on software. For example: the goals of the users of software, their preferences for communicating among themselves and how developers communicate with users. Unfortunately, "software creation is limited by the ability of people to express their thoughts" (Cockburn, 1995). This is also reflected in the available software modelling languages which do not posses all the nuances of the spoken language. For example, ideas expressed in English will have to be adapted to the UML (Unified Modelling Language,

one standard for modelling software.) standards for software to be developed. Valuable information could be lost in the process of translation between one language to the other. Similarly, software programming is also limited by human's ability to think about the problem and the solution (Cockburn, 1995). Software is a result of people's thoughts and depends on whether developers are able to devise a good solution for their users. On the same line, the evolution of software is limited by the capacity of humans and human social systems to deal with change (Constantine, 2001, 128). For example, the acceptance of new and improved software is always dependent on the capacity of their users to adapt to it and to the new processes its use will be involved with. According to Perry (2004) three are three sources of evolution. The first source is the domain or the real world which evolves in two ways: independently of the software and as a consequence of the software being operational in the real world. The second source is the experience of users of software and developers who accumulate knowledge about the software and the domain (the organisational and software development contexts respectively). This knowledge is used then as feedback to change or improve the software. The third source of evolution is the processes in the form of methods, technology and organisation which provide the contextual culture and structure for software systems to evolve.

Because it is affected by market pressures and depends on humans' ability to create it, software is predisposed to failure. Wrong conceptions of its nature make people overlook what is important for it to be successful. Constantine (2001) states that while hardware has become more reliable through the years, "software has become far less so" (Constantine, 2001, 128). Good software should be "usable, reliable, defect free, cost effective and maintainable" however it is none of those things (Mann, 2002, 34). According to Warne (2003), many authors have studied the degree of failure in the software industry, and have found, though through different measures, high rates of software projects failure (Warne, 2003, 106). The reasons for software failures have been

associated with the mechanistic approach to software development and not to the inability of technology to perform satisfactorily (DeMarco and Lister, 1999, 4). Mechanistic approaches are the ones focused on the characteristics of the technology and not on the characteristics of the human systems which will use it. So far the prevailing software development approaches have been the technical, engineering, software centred ones (Cooper and Reimann, 2003, 6; Clegg, 2000) which define quality in a technical way (Pfleeger, 1991, 5) but which neglect the human and social aspects of the organisations (Wastell and Newman, 1996, quoted in du Plooy (2003, 43), Clarke and Lehaney (2000, 10)). A narrow, mechanistic focus is the cause of faulty software, where inefficiency, poor designs cause discontent in users who find the applications difficult to understand (Mann, 2002). Cooper and Reimann (2003) state that failing to imbue software with humanity and not taking into account users' goals causes software failure (Cooper and Reimann, 2003, 5). Also, Winograd and Flores (1986) state that software failure is caused by not designing software appropriately for the context (the social and political situation) in which they will be operated (Winograd and Flores, 1986, 84). Additionally, Cooper and Reimann (2003) also state that not having reliable development processes contributes to that failure (Cooper and Reimann, 2003, 25). Lyytinen and Hirschheim (1987) define software failure as "expectation failure". That is when a software project does not meet the stakeholders' expectations. For them there are three types of failure. The first one is correspondence failure, and happens when the objectives of a project are not met. The second is process failure, when the project cannot produce a functional system. The last one is interaction failure, when the users reject the software totally, by not using it at all, or by partial use (Lyytinen and Hirschheim (1987) quoted in Warne (2003, 106)). So, in their different terms all these authors lay the problem of software failure at a narrow focus on technology and hence to improve the rate of successes of software it is very important, to encourage a closer look at human and social issues when developing software.

A good example of considering humans when developing software is Cooper and Reimann's (2003) approach to modelling software. Cooper and Reimann (2003) define two kinds of models relevant for software development. First is the users' mental model of the problem situation and how they do their jobs, and second is the implementation model which reflects how software works. In between these models there could be a spectrum of "represented models" which developers use to represent the system and show to their users. The relationship between these models is shown in figure 3.2. Cooper and Reimann state that for the software to be successful, developers need to use a model which is closer to their users' mental model and far from the technological model. A good explanation for this, in the context of groupware software, is given by Orlikowski (1997). She says that when users' mental models differ from the software implementation model, the technology will be unlikely to facilitate their work (Orlikowski, 1997, 231). Users may try to use the technology differently in a way that concurs with their mental model but which does not concur with the developers' intentions at developing the software.

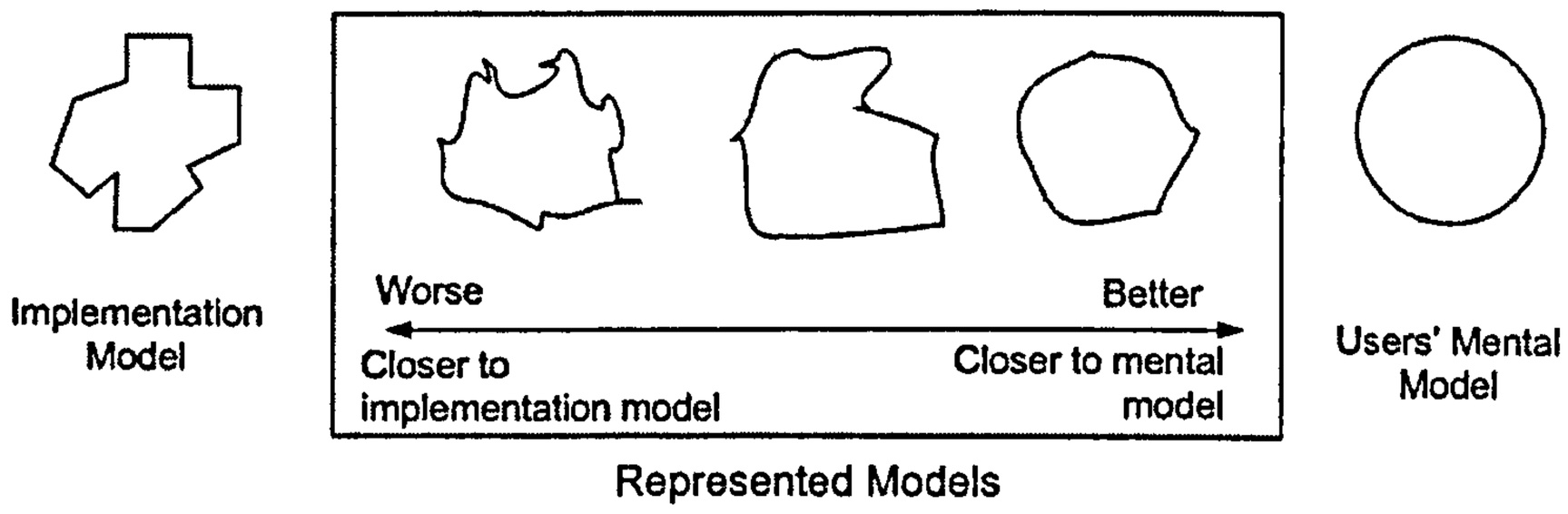


Figure 3.2 Representation of Models of Software
Source: Cooper and Reimann (2003, 22)

Cooper and Reimann's model is aimed at software developers. It explains, in a very simple way, how different models can emerge from the same situation, depending on the perspective of the viewer. It also emphasises the importance of the users' ideas of how their job is (or

should be) done and, how at the end, it is them who will decide on whether the software is good or not *for them*.

A classification of software systems relevant to the model of complex organisations

In chapter 1 and chapter 2 I described the contexts within which the kind of software that interests this thesis is used. The kind of software used in the contexts I am referring to is one that serves business decision making processes in which groups of individuals are involved and which could cross organisational boundaries. These processes are highly adaptable and unpredictable due to the human and social factors involved. At this point, I will provide a classification of software that takes into account the above circumstances and issues and which are relevant to the model of complex organisations. In the next sections I will discuss the following kinds of software.

- Decision support systems
- Software designed to support group work
- Social Software

The above classification describes how human and social issues within complex environments are inscribed in the software and how these issues affect the development process. This classification is also ordered chronologically and could be considered as a historical account of the evolution of social software. This chronology shows how the interest in human and social issues evolves through time.

3.1.1 Decision support systems

Generally, decision support system (DSS) software is built upon mechanistic models of organisation and decision making processes. These models of decision making are based on typologies of decisions

and decision processes. One of the most well known classifications of decisions is the one created by Simons (1977). According to him there are different kinds of decisions for different kinds of problems. Table 3.2 shows this.

Decision	Problem
Structured	Repetitive problems for which standard solution methods exist.
Semi-structured	Problems that have some structured elements and some unstructured elements.
Unstructured	Fuzzy, complex problems for which there are no cut-and-dried solution methods.

Table 3.2 Types of Decisions
Source: Turban and Aronson (2001, 11)

Similarly the process that decision making support software follows in general resembles very much the one shown in figure 3.3. In the intelligence phase decision makers need all information available to simplify and understand a problem situation. During the design phase, decision makers design possible solutions by, for example, using mathematical algorithms. Each solution is then contrasted to the real problem situation to assess its validity. The best solution is selected and tested in the choice phase and finally implemented. This framework for decision process could represent a complete process or just part of it as a decision making process could involve many decision making sub-processes. For example, people resolve at a certain point within the wider decision making process to look for more information, to summon other people, to postpone decisions or to cancel the process. These are also smaller decision making processes within the wider one.

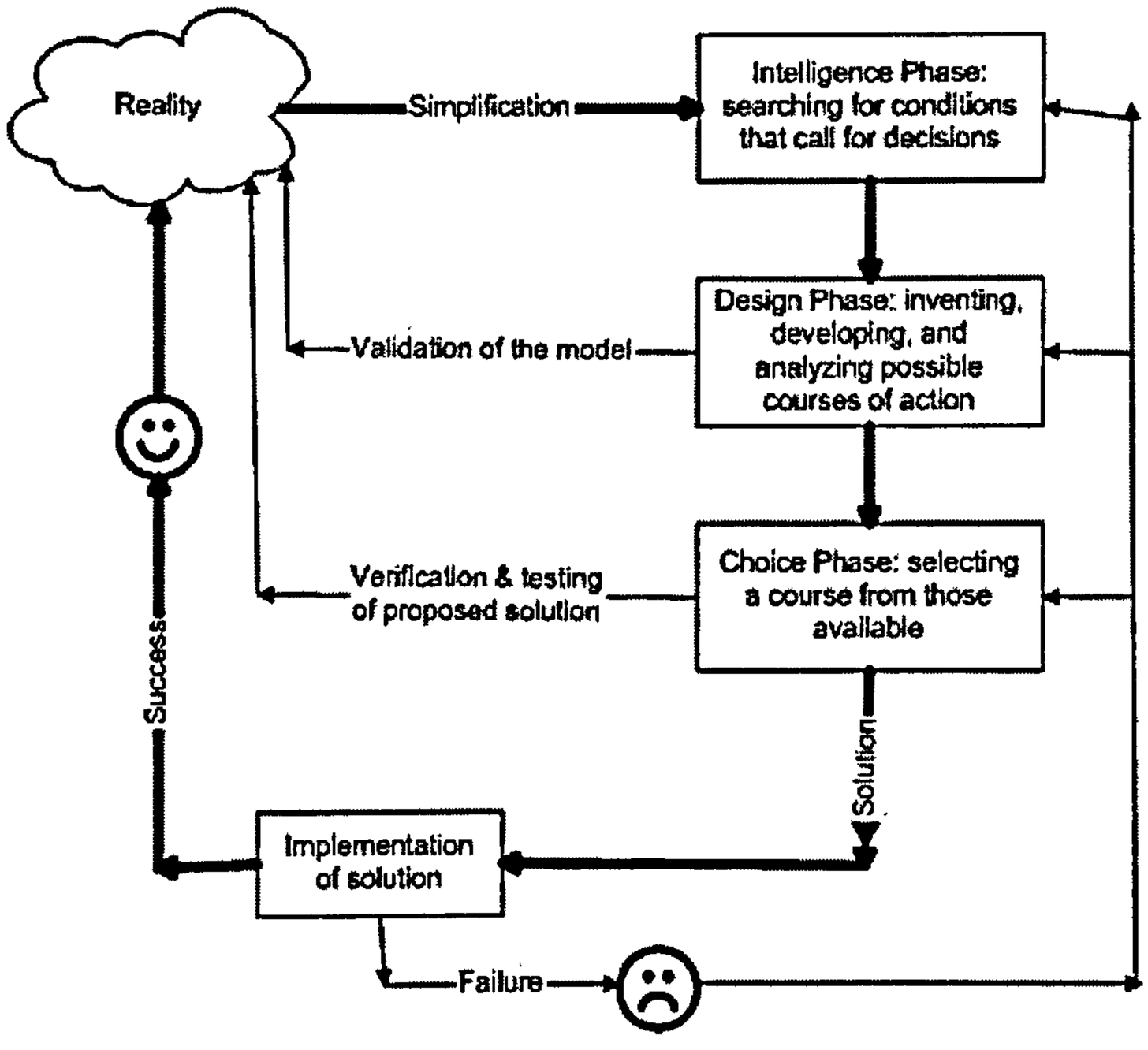


Figure 3.3 Decision-Making process
Source: Turban and Aronson (2001, 42)

Following this framework, different kinds of applications will be created to deal with different kinds of decisions. Similarly there are applications built to support one or more of the stages in the process of decision making. These applications could also be developed and sold separately. It is usually the collection of support tools which are known as DSS. Generally a DSS is a computer based system that supports complex decision-making and problem solving (Gachet, 2001, 214; Shim et al., 2002, 111). Similarly, a Group Decision Support System (GDSS) facilitates the solution of problems by a set of decision makers working together as a group (Wilson, 1991, 10). DSS and GDSS contain tools that are used for data analysis, such as OLAP, data mining tools (see glossary for a description of these tools), and mathematical and analytical models (Laudon and Laudon, 2005, 420). Usually, DSS and GDSS are used at the higher levels of the hierarchy of organisations as they aid in the development of their strategies.

Holtham (1992) states that DSS and GDSS should be designed to fit the culture and style of managers and to summarise critical organisational and environmental data that help them to make decisions. According to Holtham (1992) there are three factors that need to be considered when designing DSS: organisation competencies which give the comparative advantage, culture and management style, and business environment (opportunities and threats) (Holtham, 1992, 37). However, in practice, efforts to improve DSS and GDS tools are made in the area of efficiency at utilising storage space and speed at responding to users' queries. Unfortunately, one of the consequences of this narrow focus, as explained before, is software failure.

3.1.2 Software designed to support group work

In this section I will explore the characteristics of software used to support groups at doing different business activities and hence seem to be very closely relevant to this thesis. These activities include, communication (e.g. sharing information among team members), collaboration (group members performing tasks together) and decision making (GDSS classifies as well under this topic). Chronologically these kinds of software were conceived after the DSS and generally involve also the support of decision making. The kinds of software included in this discussion are Groupware, CSCW (Computer Supported Cooperative Work), Collaboration tools and Workflow.

> "Groupware is software that supports and augments group work." (Greenberg, 1991)

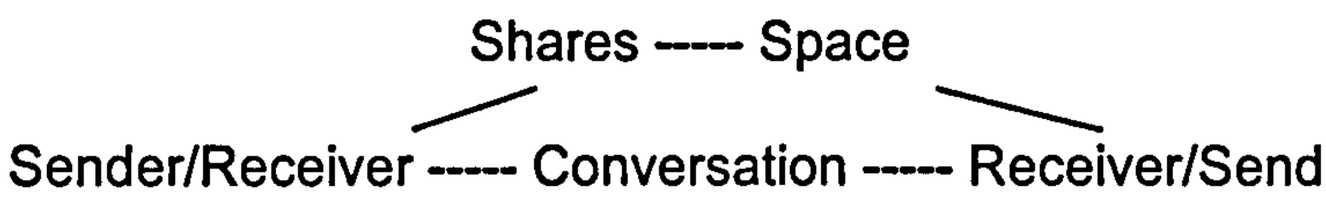
> "CSCW is understood to be a generic term which combines the understanding of the way people work in groups with the enabling technologies of computer networking and associated hardware, software, services and techniques." (Wilson, 1991, 6)

Groupware and CSCW are basically the same thing. They are two different names to describe the same kind of software. Whereas groupware is a term mostly used by practitioners, CSCW is a more academic term which describes the "scientific discipline that motivates and validates groupware design" (Greenberg, 1991, 1). Groupware applications focus on facilitating communication between people. Groupware supports events like group meetings or conferences. It offers tools for brainstorming, classification of ideas, prioritisation and voting. These activities can be performed in one room, with all the members present or can be carried out in distributed environments, with members in different parts of the world. These meetings and conferences can be synchronous or asynchronous. This means that either people are communicating at the same time or at different times and the groupware tools help them to keep track of the conversations.

Collaborative tools are similar applications to groupware but their emphasis is on supporting people who are working together to produce or create something. Although they are used in social contexts this does not mean that the elements of those social aspects are included as features of the collaborative tools. In fact the methodologies used to develop collaborative tools are predominantly technology oriented (focused on efficient use of space and improving speed). In fact, the aim of collaborative tools is to make collaboration faster, better, and effective. Schrage (1997) uses a conversation metaphor to explain how collaborative tools work. For Schrage (1997), collaborative tools "embody the visual and verbal languages that people need when they have to do more than just transmit information" (Schrage, 1997, 170). Schrage (1997) uses a "collaborative model" to explain how collaborative tools enhance conversations. In normal conversations, people take turns to explain their ideas following the model below

Sender/Receiver ----- Conversation ----- Receiver/Sender

In conversations ideas are exchanged from sender to receiver but not shared. Thus according to Schrage (1997) conversations lack of memory. The collaborative model includes a new dimension to conversations in the form of shared space which participants can use to enhance their communication. This shared space resembles a whiteboard in which track of conversations is kept.



A different kind of tool but which is also used to support group work is the workflow system. Workflow tools are designed to help people on routine, repetitive tasks usually required at low levels within the organisations (Aldred, 1994, 68). This is different from groupware and collaborative tools which are designed to support more unstructured tasks. Workflow systems are built to make work "controllable" (Van der Aalst and Van Hee, 2002, 3) by providing predictable and inflexible processes. Some tasks within the workflow system can be performed without the participation of people, and some others require human intelligence, for a judgement or a decision. Although workflow and groupware (or CSCW) are usually used at different organisational levels (workflow at the operational and groupware at the strategic) they may overlap at the tactical level. For example, a workflow procedure may trigger a groupware application and vice versa (Aldred, 1994, 68).

Despite being built with different purposes all these kinds of technologies share a similar characteristic: they all have all the design challenges of individual user applications supplemented by the requirements emerging from the involvement of working groups (Grudin, 1994, 95). This makes group work software a very difficult field to work in. Kim (2004) argues that collaborative tools "are nowhere close to fulfilling their potential" (Kim, 2004). King (1991) also states that "crucial aspects of work are poorly understood" by systems developers who design software for group work (King (1991) quoted in Walsham (1993, 192)). As a consequence

severe difficulties arise when those systems are introduced. King therefore argues for collaborative design in which systems designers learn from their users and users learn from designers (King (1991) quoted in Walsham (1993, 192)). Also Grudin (1994) calls for an improvement of groupware development practices by considering the nuances of the working places in which the software is going to be used and the changes its introduction into those working places the groupware will bring (Grudin, 1994, 95). Kim also argues for a consideration of human aspects at building applications by being "people-centric" (Kim, 2004).

In addition to the inclusion of human and social aspects in group work software design practitioners and academics bring up some other issues that need special attention. For example, groupware and collaborative tools may not be that good at aiding communication. Tang (1991, 11), in his study of collaborative drawing, found that there was information that was generated in the process of drawing that the computer did not grasp and therefore was missed by participants. Lea and Spears (1991) found that "groups communicating via computer produce more polarized decisions than face-to-face groups" (Lea and Spears, 1991, 155). This is because computerised environments enhance disinhibition and therefore provokes the emergence of extreme arguments. Tatar et al (1991) found that the collaborative tool that they were studying was supporting a "parcel-post" model (Tatar et al., 1991, 62) of communication in which the interactive nature of communication was lost. On a different level, Van der Aalst and Van Hee (2002) state that workflow systems do not provide flexibility to their users and are difficult to adapt. For Van der Aalst and Van Hee workflow systems and workflow development need to go under structural modifications to deal with the new changing market requirements (Van der Aalst and Van Hee, 2002, 192).

The availability and use of software tools that support group work can be considered a step forward towards including human and social aspects. However, as it has been shown in this section, there still are problems at understanding human and social issues and applying that knowledge in

the development of software. In the next section I will discuss social software, which is a type of software that pays explicit attention to social interactions between users of software. Social software represents the latest step in the software development field in the consideration of human and social issues in software. However, social software is not necessarily aimed at organisational contexts but the much larger World Wide Web.

3.1.3 Social Software

Social software is both an old and a new concept. It has been around the software development field for decades but known by different names. According to Allen (2004) social software is a term that has evolved from office automation, groupware, CSCW (computer supported cooperative work) and CMC (computer mediated communication) (Allen, 2004); and hence social software flows from the discussion in the previous section. However there are some differences from the type of software mentioned above. Social software is software that allows, encourages or facilitates social interaction within groups (Shirky, 2003) and facilitates the control of the information created through those interactions (Arnold, 2003). Modern forms of social software include weblogs (online personal journals, for example www.blogger.com), Wikis (collaborative-edited websites, for example www.wikipedia.org), IM (Instant Messaging, for example MSN Messenger) and online forums. Although social software in general is not considered as "business software", it is relevant to this discussion because it is the category of software which pays explicit attention to human and social aspects of groups using software and which addresses those problems noted in the previous section.

Boyd (2004) identifies the potential of social software at providing means of self-disclosed communication to people, though through limited social life models. Unfortunately, she says, current social software is based on

unreal social models which force people to behave unnaturally. The characteristics of social software according to Boyd (2004) are:

- It allows users to have "parallel lives" and "multiple selves" freeing them from their physical bodies
- It encourages pseudonymous participation. By using nicknames users can hide their true identity
- Social software attempts to formalize how people should construct and manage their relationships
- Users of social software act as autists when they are forced to engage into abnormal social interactions

(Boyd, 2004)

Despite its drawbacks social software is being improved to fulfil users' goals. Basically, the goals of users of social software are to communicate and to get information. Therefore social software has to provide reliable means for communication and information storage, searching and sharing (Sedelnikov, 2004b). However, there is more to social software design than providing means for communication. Other human and social aspects of users are considered. Social software takes into account user's interactions with computers but also interactions between users through the computer. Ideally social software is designed considering the individual and social characteristics of their users. That is to acknowledge the user of social software as a group and not as an individual (Shirky, 2004a; Shirky, 2004b; Shirky, 2003). Additionally, designers identify the kind of community they are designing the software for to know which kinds of behaviours should or not be encouraged by the software. This means that the emergence of social software expands the application of usability guidelines to "group-usability" guidelines which consider social interactions. Spolsky (2004) names this characteristic of social software as "social interface" (Spolsky, 2004). According to Spolsky, the success of social software is dependent on how the social interface of an

application fits the characteristics of the group of users and how it helps the whole community succeed at using it. Additionally Spolsky (2003) states that "in software, as in architecture, design decisions are just as important to the type of community that develops or fails to develop." Sedelnikov (2004a) emphasises the importance of considering user-to-user interaction at developing social software and proposes the participation of social psychologists in the development process. (Spolsky and Sedelnikov's statements are also true for software that support group work as there is also a (smaller) community and user-to-user interaction that need to be served).

Consideration of social issues is not an easy task. "Social interactions are far more complex and unpredictable than human/computer interaction and that unpredictability defeats the classic user-centric design" (Shirky, 2004a). However some guidelines are provided by some authors. For example, Spolsky (2004) states that for social software design one has to look at sociology and anthropology. Whereas Boyd (2004) says that designers have to think about what social practices they are aiming to address and what values we are inserting while trying to address them. Additionally, she recommends that developers should make social software fit into people's lives and practices rather than creating artificial needs. Yet social software has created many artificial needs which are now part of our lives. In relation to this, in the next section I will explore how current software development practices address these and other issues.

3.2 Software Development

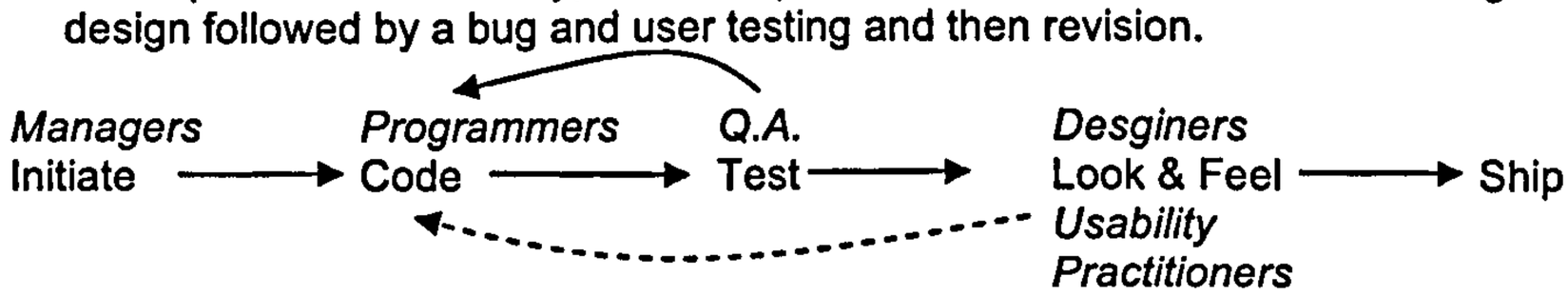
Software development involves the design, creation, implementation and maintenance of software. In the broader sense, the term software development can be applied to describe the development of any kind of software, as described in the previous section. However, in this section I will focus on the development of business software. Software

development involves – but is not limited to - programming computer code. Software development comprises a variety of activities such as planning, design and testing. Although it involves the use of technologies, it has long been recognised as a human activity (Weinberg, 1971, 3). Cooper and Reinmann (2003) provide an interesting account of the evolution of software development, which describes and explains how it changed from being a mere programming task to a more elaborated job. The evolution of software development is shown in figure 3.4.

1. Originally, programmers did it all: in the early days of the software industry, smart programmers dreamed up useful software, wrote it, and even tested it on their own. But as their businesses grew, the software business and software products became more complicated.

2. Managers brought order: inevitably, professional managers were brought in. Good product managers understand the market and competitors. They define software products by creating requirements documents. Often, however, requirements are little more than a list of features and managers find themselves having to give up features in order to meet schedules.

3. Testing and design became separate steps: As the industry matured, testing became a separate discipline and a separate step in the process. In the move from command-line to graphical user interface, design and usability also became involved in the process, though often only at the end, and often only visual presentation. Today, common practice includes simultaneous coding and design followed by a bug and user testing and then revision.



4. Design must precede the programming effort: a goal-directed approach to software development means that all decisions proceed from a formal definition of the user and his or her goals. Definition of the user and user goal is the responsibility of the designer - thus design must precede programming.

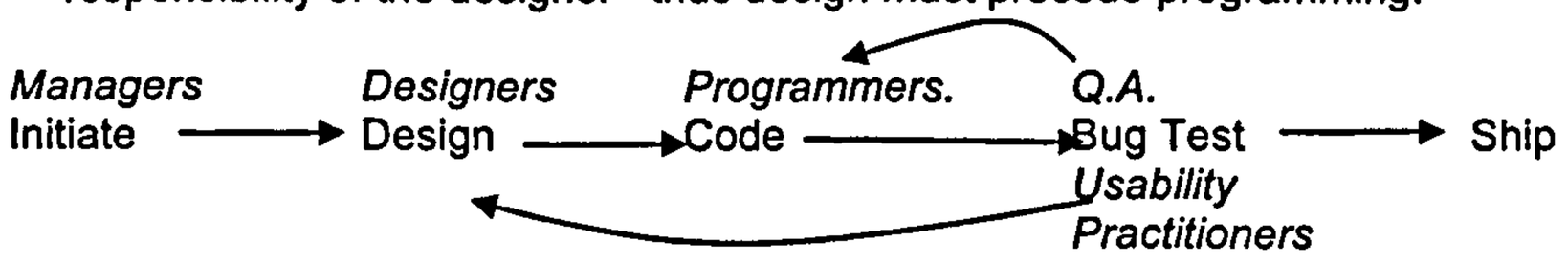


Figure 3.4. Evolution of the Software Development process
Source: Cooper and Reinmann (2003, 6)

As the above exhibit shows, the activity of design has gained (theoretical) importance and with this has developed a social side (software designers collaborating with users) within the development process. However, in practice not all software development is produced as shown in stage 4 of evolution. Much of the most successful commercial software is done without the benefit of analysis and design (Constantine, 2001). For example, Bill Gates declares that he does not want his programmers to do design (Constantine, 2001). Constantine (2001) also states that even the vendors of modelling tools do not use their own tools. One way of explaining this phenomenon is that for developers the task of programming itself is a design activity. This means that coding and design cannot be done separately (Reeves (1992) quoted in Fowler (2004b); fieldwork notes). However, this does not mean that developers should work without an underlying plan. Fowler (2004b) states that not having plans leads software projects into chaotic states and argues for development methodologies to impose disciplined processes. Along the same line, Fowler (2004a) states that new development methodologies (like XP, Extreme Programming) are abandoning "big up front design" in favour of design along coding. This new way of doing things improves the productivity of developers as it allows them to respond to changes faster. Fowler (2004b) also identifies four characteristics of software development which describe it as a complex activity. These characteristics portray software development environments as complex organisations where adaptable, unpredictable and multi-group work is done. The characteristics of software development are:

- In software: construction is so cheap as to be free
- In software all the effort is design, and thus requires creative and talented people[1]
- Creative processes are not easily planned, and so predictability may well be an impossible target.

[1] Possibly what Bill Gates means by not wanting his programmers to do design is that he does not want design as a separate activity but one which is part of the programming task

- We should be very wary of the traditional engineering metaphor for building software. It's a different kind of activity and requires a different process

  Source: Fowler (2004b)

Fowler (2004b) also states that because software development is mostly a design activity (meaning design + coding), it is difficult to plan and cost. This is especially true in development of business software. Business software is subject to changing and unpredictable requirements (Fowler, 2004b) maybe more than other kinds of software. The reason is the human and social factors within business organisations and the nature of their jobs which dynamism is difficult to design. For example, in software development for mobile phones it is more feasible to come up with a plan and a design relatively fixed (after market research), and work onwards with no significant change. The requirements for business software (particularly for internal software) on the other hand are more variable as they are more dependable on the unpredictability of humans. This topic was also discussed in chapter 2, where unpredictability, adaptability and multi-group work where explored as human and social issues in complex organisations.

From the discussion above, software development is affected by human and social issues emerging from the organisations that develop the software and the organisations that will use it. In the following paragraphs I will expand on this idea.

Weinberg (1971) states that the development environment is a "rich and complex environment, full of human involvement, change and misleading appearances (Weinberg, 1971, 64). This is probably one of the reasons why Cockburn (1995; 1998; 2000b; 2003) argues that people (developers) are the "first-order" factors in the success of software development and relegates process factors as second-order issues. Cockburn studied several software development projects in a quest for

the perfect methodology which would lead any software project to success. He designed many methodologies and tools and implemented them in real projects. However, these methodologies and tools did not work as expected or were not accepted by developers. In fact, the methodologies had little effect on the outcomes of projects (Cockburn, 2003, 10). On his many interventions, Cockburn found out that:

- The people on the projects were not interested in learning his system (tool or methodology)
- They were successfully able to ignore him, and were still delivering software, anyway.

Source: Cockburn (2000b)

Cockburn then deduced that there was something that he was missing that was at the core of software development. This something was the determinant factor of the success or failure of any software development endeavour. As he explains, he examined his fieldwork notes and found that the following sentence

> "A few good people stepped in and did whatever was needed at the time." (Cockburn, 2003, 53)

was uttered consistently by team members of successful projects. This plus the assertion made by many project managers of projects he studied:

> "Just give me a few good people and let us work in the same room, and we'll deliver you software." (Cockburn, 2003, 10)

led Cockburn to the answer he was looking for: people, their expertise, strengths and weaknesses. Therefore Cockburn (2000b) decided that human factors were the determinants of success. As developers should work in the same room the role of communication (and hence its social aspects) becomes significant. This however, is not the first time that

human factors have been regarded as important in the software development field. There have been other writers, though very few, who have also emphasised human and social factors in software development. For example, Weinberg (1971) discusses the characteristics of programmers and teams of programmers from a cognitive perspective. Also, DeMarco and Lister (1999) examine the sociology of software development teams and propose some guidelines to avoid failure.

Following his breakthrough Cockburn continued to study developers to identify their characteristics and determine how they affect positively or negatively software development projects. Cockburn (1995) found out that in general methodologists had been neglecting the fact that people work in a non-linear way. This means that "they do not follow any predictable sequence in going from problem to solution" (Cockburn, 1995). This characteristic contrasts with the way methodologies enforce work on developers in a predictable way. Cockburn (2000b) identifies four main characteristics of people (in general and developers in particular) that explain their non-linearity:

1. **Developers are communicating beings:** developers need to communicate to do their jobs, share ideas, collaborate and socialize. For Cockburn, the best and richest way for developers to communicate is face-to-face. Face-to-face communication allows people to engage in real-time, like in a question and answer interaction. Face-to-face communication also allows developers to communicate through different means like vocal inflection, gestures and other visual cues. For Cockburn, if developers are impeded from communicating face-to-face or communicating in general the effectiveness of their jobs will be affected negatively. See figure 3.5.

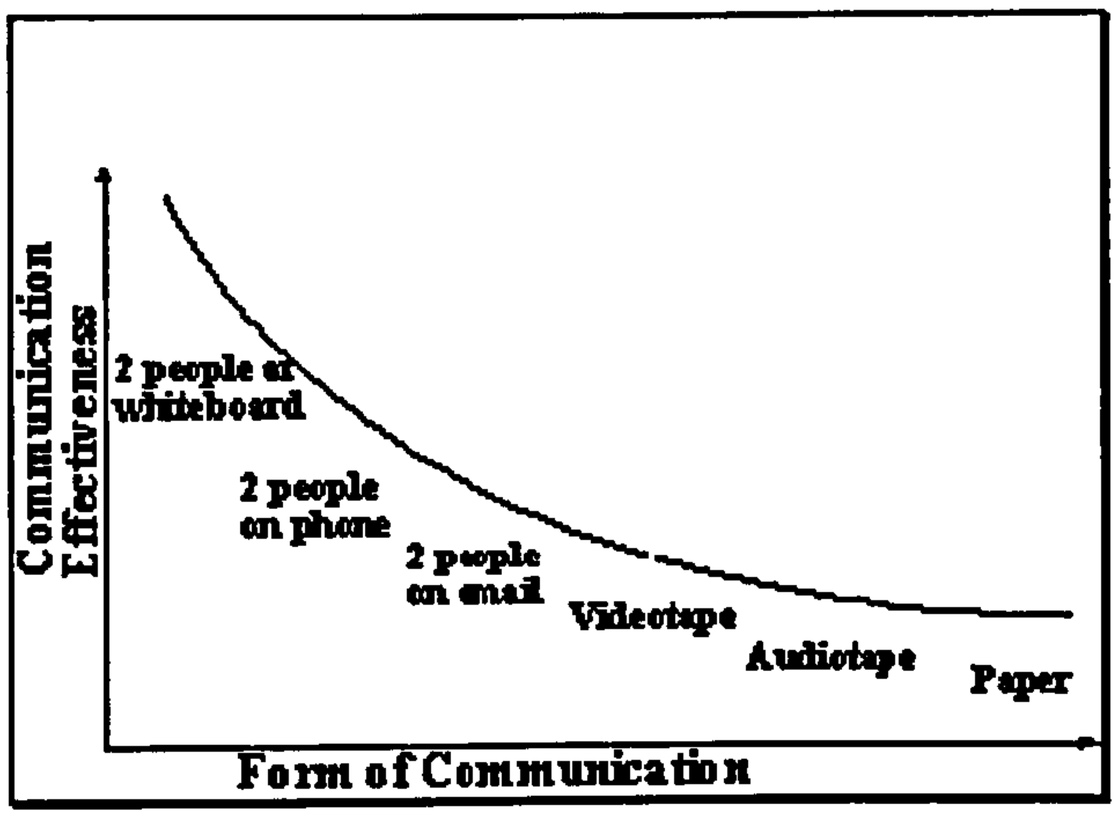


Figure 3.5 Modes of Communication
Source Cockburn (2000b)

* Another form of communication which is not considered in Cockburn's diagram is communication through virtual communities. If I follow the sequence in the above curve, virtual communication would go alongside "2 people on email" (e-mails are also tools used in virtual communities though not the most important). Then according to that diagram virtual communities would offer a not so effective form of communication. However, I would argue that virtual communities offer a more effective form of communication that the one the diagram would suggest (if virtual communities where included). This thesis' standpoint is that virtual communities of software developers are "places" where real development work can be done and relevant topics are discussed. I present a defence of this argument in section 4.2 of chapter 4.

2. **Developers tend to inconsistency:** Cockburn states that people can do similar things repeatedly but they will never be exactly the same. Ignoring people's inconsistency is what makes high-discipline methodologies fail in environments in which developers cannot be consistent.

3. **Developers vary**: some developers are good at programming algorithms; some others are good at programming GUI's

(Graphical User Interface). Some developers like to work in teams, others like to work alone. Cockburn (2000b) states that knowing how to combine developers in teams and assign the right jobs to take advantage of their abilities are important factors in development success.

4. **Developers are interested in being "good citizens"**; they take initiative and are good at looking around: These characteristics are reflected in the expression mentioned before "a few good people stepped in and did whatever was needed at the time" (Cockburn, 2003, 53) and are an answer to the problem of inconsistency.

The characteristics mentioned are concerned with the nature of individual developers and their relationships with other developers as they work in a development environment. However, there are other human and social aspects emerging from the organisations they are working for (e.g. users) that affect their jobs. Some of these aspects have been explored in chapter 2 as human and social aspects in complex business organisations. An example of how these aspects can affect developers is changing requirements from organisations whose functional specifications are difficult to specify due to their high rate of change, the lack of formal procedures, or the inability of users to describe their jobs. In this line, one of the most commented human related aspects in the software development literature is the relationship between developers and users.

Modern software development practices emphasise the participation of users in development teams. In this way users transfer their knowledge of the problem situation in a more effective way. The reason for having users closer is that neither programmers nor analysts "are best suited for stating the requirements" of a system (Cockburn, 1995). Having users as part of the team improves user-developer communication and helps developers to immerse and understand the business of the users and their requirements (du Plooy, 2003, 53). Many human and social issues

arise from this relationship. For example, Warne (2003) reports that conflict between users, and between users and developers "can impact on the success of the project" (Warne, 2003, 129). For Warne it is very important to consider these kinds of conflicts and manage them effectively to increase the probability of success (Warne, 2003, 129). Spolsky (2002b) provides some practical examples of how conflict between users and developers hinder the development work:

"We built it *exactly the way they wanted*. The contract specified the whole thing down to the smallest detail. We delivered exactly what the contract said. But when we delivered it, they were crestfallen."

"Our miserable sales person agreed to a fixed price contract to build what was basically unspecified, and the customer's lawyers were sharp enough to get a clause in the contract that they don't have to pay us until 'acceptance by customer,' so we had to put a team of nine developers on their project for two years and only got paid $800."

Source: Spolsky (2002b)

Due to these kinds of problems Spolsky's position has become radical. He recommends developers to work under the assumption that "customers don't know what they want" and recommends fellow developers to "stop expecting customers to know what they want" (Spolsky, 2002b). This position also corresponds to some software approaches such as those in which business analysts are assigned to investigate users' work and to deliver a requirements document or prototype so developers do not need to interact with their users. The effect of human and social issues on developers, such as their relationship with users, and the relationships between users, is a decisive factor at choosing their methodology. Some developers would prefer to work closer with users whereas others would prefer to put some distance between them. Then the question that needs to be asked is how well methodologies are supporting software development.

3.2.1 Software Development Methodologies

In the following sections I will explore some of the software development methodologies which are used or referred to in practice. During my fieldwork, the online conversations in which I participated helped me to identify the kind of issues and software development literature the developers are interested in. I found the following methodologies the most commented by developers, either because they were using (or had used) them or because they had an interest in them. The methodologies presented in the next sections are:

- The waterfall Lifecycle
- Prototyping
- Agile Methodologies

In section 2.2.3 of chapter 2 I presented a set of ISD approaches which focus on concepts and beliefs about the nature of organisations and information systems and how information systems development should be done. The set of methodologies presented here are not ISD approaches or intellectual frameworks but the actual plans and instructions that are applied in practice.

a. The Waterfall Lifecycle

The waterfall model is also known as the traditional lifecycle. It is been influenced mainly by the ISD organisational approaches such as Project Management and Business Process Reengineering, and may also follow modelling approaches such as the data and process oriented approaches. The waterfall lifecycle is a software development methodology that emphasises division of labour. In the waterfall model the whole process of software development is divided into stages such as analysis, design, coding and testing. A stage doesn't start unless the previous one has finished. So that one stage's outcome becomes the

next stage's input. Also, work on the stages is performed by developers with different expertise (such as programmers and testers). The Waterfall methodology is usually criticised because of its inflexibility and its focus on technology (Eason (1998) quoted in Jackson (1985, 23)). The waterfall lifecycle works under the assumption that requirements never change. Software is created based on a snapshot of the organisation known at the start of software development. And it is assumed that this snapshot will reflect the organisation's nature any time in the future. For this reason, projects define in advance the cost, the time and design of the software. Work on the waterfall lifecycle is performed under these parameters. If a deviation from this plan occurs, it is considered a mistake and corrective actions are taken. However, more modern forms of waterfall, like the spiral, allow repetition of stages to correct their outcomes. The Waterfall life cycle also emphasises the creation of formal documentation. In fact, it is this documentation which is passed from one stage to another and is built upon in each stage. This is also a source of criticism as it takes time for developers to complete documentation and to update it at the same time they are programming. Nevertheless, the waterfall lifecycle is still used in many projects. It suits large projects in big corporations where processes are standardized and requirements are structured and well defined (Laudon and Laudon, 2005, 464).

b. Prototyping

Prototyping is not really a proper software development methodology but a popular practice which is used as part of many methodologies. It follows the prototyping, engineering approach presented in section d of appendix 10, (which is an extension of section 2.2.3 of chapter 2). A (software) prototype is a "working model" (Allen and Frost, 1998, 10) or a preliminary version of "all or part of a system before full commitment is made to develop it" (Smith, 1990, 42). Prototyping consists of building that version of the software rapidly and inexpensively for end users to evaluate (Laudon and Laudon, 2000, 372). Prototyping is an ongoing learning process (Bodker and Gronbaek, 1991, 331) in which users and

developers sit together so developers can get first hand knowledge about the problem situation from the users and users can get a very accurate idea of how the software will look like and behave. Prototyping counters the weaknesses of the waterfall lifecycle in which interaction between developers and users is not promoted. Other benefits of prototyping are that it speeds up the development considerably and that it avoids tons of documentation which are replaced by the prototype itself. There are three kinds of prototypes:

- Type one: a software prototype which is created with the same tool that the real version is going to be created. The prototype is considered by developers and users as the real product. In fact, the real version will be completed from the prototype version.
- Type two: a software prototype which looks like the real version but which does not have the functionality. It is used to discuss with the users how they want it to look and work. The real version will then be started from scratch.
- Type three: a paper prototype. Paper prototypes contain screenshots of the would-be system. They could be hand written or designed with specialized tools.

The advantage of a type one prototype is that the programming work is done from the start and that if the job is done well then it will save time for the product to be delivered. Type two prototypes are faster to develop as the developers use specialized tools and because no work on the functionality is done. A plus of types one and two prototypes is that being hi-tech techniques they help to show off the latest technology (Cockburn, 1995). The disadvantage of type one and two prototypes is the advantage of type three prototypes: showing users a working prototype (type one or two) would make them think that most of the job is already done (Spolsky, 2002b). By using prototypes that do not look like the real version (prototype type three), but show sufficiently enough how it will work, users, will not get the wrong idea and developers will not be put under

pressure. Also being a lo-tech task it helps to maximize the cognitive effects in design (Cockburn, 1995).

c. Agile Methodologies

Agile methodologies are a set of software methodologies that follow the People or Participative approaches presented in section 2.2.3 of chapter 2 (and presented more in detail in section c of appendix 10). The agile methodologies emerged as a reaction to the bureaucracy of the monumental methodologies (Fowler, 2004b) such as the traditional lifecycle. Cockburn (2001) defines an agile methodology as:

> "a development approach that primarily addresses the problems of rapid change: changes in market forces, systems requirements, implementation technology and project staff occurring within a single project's development period." (Cockburn, 2001)

Although there are many agile methodologies they all share common goals as specified in the "agile manifesto" (see glossary for a definition). The goals are:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Source Cockburn (2000a, 178, 179)

Agile methodologies aim to achieve user or customer satisfaction. Whereas the traditional methodologies conceive success as finishing projects on time and on budget, agile methodologies believe that the question should be whether the software has business value to their customers (Fowler, 2004b). Thus their relevance to this thesis as they are concerned with what the customers want.

The philosophy of agile methodologies is that as modern organizations evolve and adapt to their environments rapidly so should software development. Following this, agile methodologies are adaptive rather than predictive (Fowler, 2004b). Contrary to engineering centred methods (like the waterfall lifecycle) which try to plan and predict everything in advance, agile methodologies work on the assumption that organisations, users, and their requirements change. Agile methods are also people-oriented rather than process-oriented (Fowler, 2004b). Agile methods build on the skills and experience of the developers, like for example making sure that experts are part of the team (Cockburn, 2001) to guarantee the success of the project and to allow new developers to learn from them. Other requirement from agile methodologies is that developers should have a common focus, mutual trust and respect, the ability to collaborate and deal with ambiguity (Cockburn, 2001). These characteristics represent a strong explicit recognition of the social aspects in ISD.

Other characteristics of Agile Methodologies are that they de-emphasize up-front analysis and design and minimize the documentation effort (Rising, 2001). This does not mean that no analysis or design is done but that analysis, design and programming are done concurrently in increments. Incremental development is that kind of development in which the software is divided in parts which are delivered in iterations or increments. An increment involves an iteration of analysis, design, programming and delivering of software with new (improved or corrected) functionality. It is called increment because, with every iteration, the software gets closer to completion and is improved in quality. With incremental development the most important functionality is delivered first to the users to give them time to use the system and give valuable feedback to the developers. This allows developers more time to continue developing the rest of the system. Incremental development is the source of adaptability of the agile methodologies. Developers account for changes in the requirements between each increment and are given the

time to implement them. Users also work knowing that the software they are using will be modified according to their feedback until they are totally satisfied with it.

One of the most commented upon agile methodologies (in online forums) is Extreme Programming (XP). Although XP is very programming oriented it also shares the agile philosophy. One of the most popular practices promoted by XP is Pair Programming. In pair programming developers share a computer, while one developer is programming the other one is looking for errors. The pair switches roles from time to time. Although it may be slower, pair programming reduces the overall risk of a project (Beck, 1999) as the quality of the programming code is improved. Pair programming also increases communication among the development team (a social aspect of software development). Knowledge about the system and about programming is spread rapidly.

Another popular agile methodology is the Crystal Family, which in reality is a set of methodologies designed for different kinds of projects. Cockburn (2000a), the author of the Crystal Family, states that his methodologies are people and communication centric (Cockburn, 2000a, 166). Cockburn emphasises that project managers should take advantage of the particular abilities of their team members to draw on their "success modes" and overcome their "failure modes". Cockburn states that it is good to have a variety of people in a project team as "mixed teams often outperform homogeneous teams (Scully (1998) quoted in Cockburn (2000a, 46)). Heterogeneous teams allow individuals to work on the areas they are specialized in. However, communication and personality issues arise in these kinds of environments. Within his methodologies, Cockburn addresses some of these issues and suggests some strategies to overcome them. These strategies are based on the following principle:

> "Software development is a (resource-limited) cooperative game of invention and communication. The primary goal of

> the game is to deliver useful, working software. The secondary goal, the residue of the game, is to set up for the next game. The next game may be to alter or replace the system or to create a neighbouring system." (Cockburn, 2000a, 35)

Cockburn defines software development as a cooperative game to stress the human factors and the difference with engineering perspectives which put emphasis on the process rather than in people. In fact it is people who need to be paid more attention as software development is about working with people, working in teams, dealing with emotions, striving for accurate and honest communications (Chisholm et al., 1996, 233). Following this, in the next section I will explore in more depth the human side of software development by discussing some of the characteristics of software developers and the social issues emerging from the development environment.

3.2.2 Software Developers

Developer is a term used to define people who perform a varied set of activities within the software development process. As mentioned in the previous section (figure 3.4), at its first stages of evolution, software development was thought as a programming activity but now it has evolved to include planning, design, testing and other tasks. In a similar way, the term developer has gone from meaning programmer to meaning designer, tester or all of them at the same time. Figure 3.6 shows some of the roles a developer in the area of business software can play. Apart from programming the software, and depending on the arrangements at work, some developers would have to perform other tasks designed to either prepare or implement the software. Roughly, if working at a small ISV (Independent Software Vendor) for example, developers will have to be multi-tasking and perform various roles. However, if working at large firms, they will be able to specialise in only one role (Cockburn, 1995; Sink, 2004). From figure 3.6, in some development environments

developers have to either visit their users' organisations or work with them in-house to gather information and understand the situation. With this information they would design the software specification and the technological architecture needed. Developers would also have to plan the cost and time of the project. This is particularly important if they are being paid by their customers (as opposite to working in-house). In most cases the introduction of software involves some changes within the users' organisation(s). In these cases, software developers act as designers or re-designers of organisational processes. Once developers have programmed the software, it has to be tested for bugs or errors within the code, either by programmers themselves or by dedicated testers. Additionally, depending on the methodology followed, developers will have to document all or some of their activities, at different levels of detail. The most common documentations are analysis and design documentation, also known as functional and technical specifications.

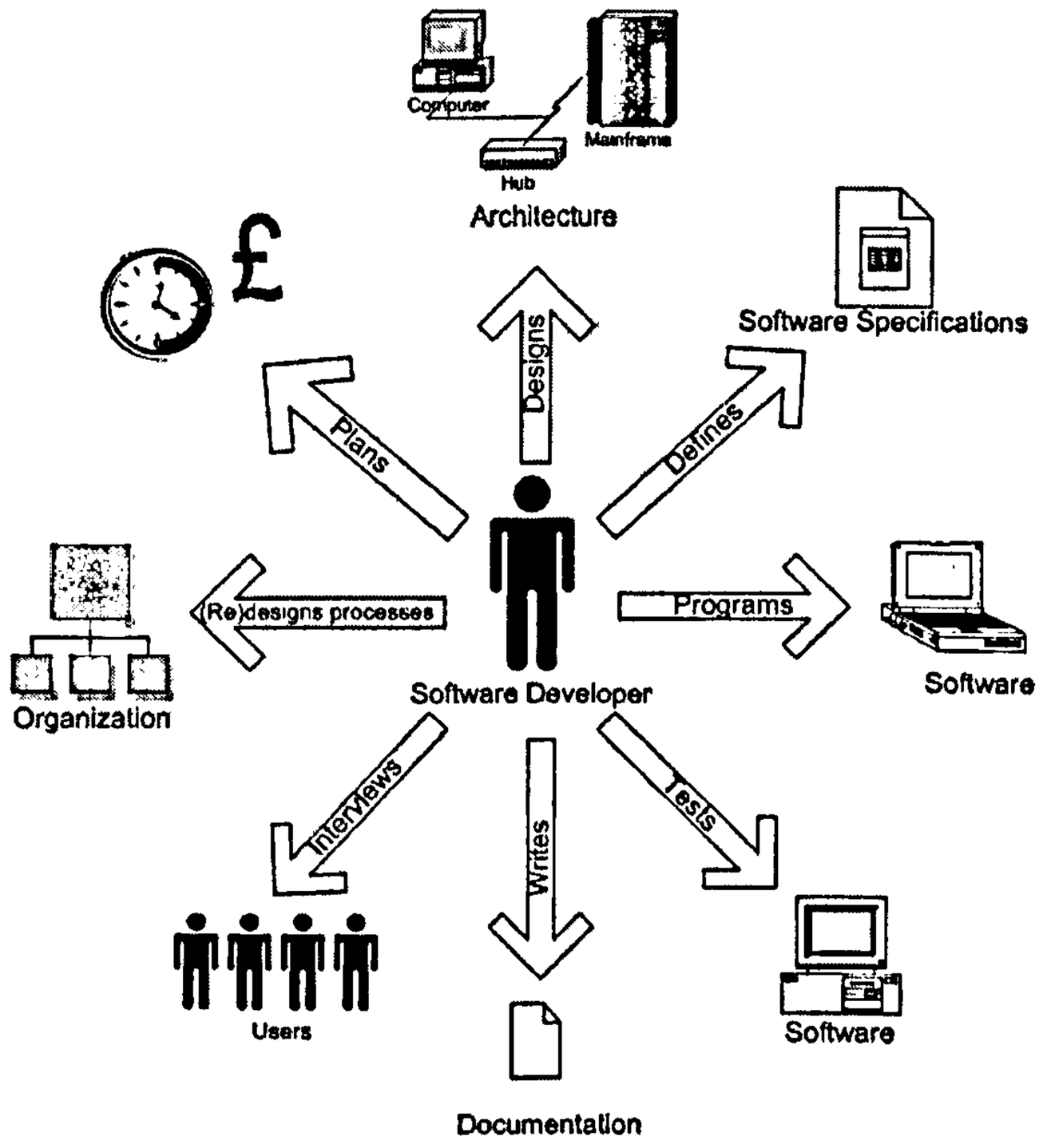


Figure 3.6 Multiple Roles of a Business Software Developer

Having surfaced the varied set of roles for developers a working definition of developers is needed. In the broader sense a developer is someone who will "contribute in multiple ways to make the product successful" (Sink, 2004). However, there is no agreement yet among practitioners about how to call each kind of specialist and to decide on his responsibilities (fieldwork data, see category C4-6 in section 6.4 of chapter 6). In fact, developers in the software business have their own terms. In some environments the name coder is preferred to developer; in others the name programmer is better liked. Likewise, some particular development activities are regarded as mundane in some environments. For example, some developers would call themselves architects to mean that they are not doing programming anymore and that they are at a higher status now (Fowler, 2004a). In other cases developers are encouraged to do more than just programming. For example, Spolsky (2002b) argues that developers (programmers) should also do the analysis and design of software as, for him, users or customers are not of much help at specifying their own requirements. In other cases, developers regard themselves as programming machines and are not concerned about writing specifications or meeting with their users. Preferences or skills may also influence these discrepancies of meanings. For example, some developers (e.g. architects) like to think in abstractions and imagine what they can do with their ideas in hypothetical situations and others (e.g. programmers or coders) like to focus on specific problems and code the solution for those problems (Spolsky, 2001). Architects will therefore be able to see the bigger picture and design solutions while programmers will be better suited for working on parts of those solutions. Yet both, architects and programmers, will share a view of themselves as developers.

Needs for different specializations or skills vary from project to project. Factors like size of development team, kind of software to be developed, platform and tools, and availability of users call for different expertise. In general, it is very difficult to decide whether a developer is the right person for a project. Sink (2004) lists some tips which in his experience

have helped him to evaluate developers. For him one has to look at experience, educational background and references. This provides a sharp picture of the developer's technical abilities. Also, for Sink, if the candidate has a university degree there will be a big chance of success. However, as Sink states this does not provide any certainty about the personality and communication skills of the individual. Second is to look for "self-awareness". A good developer is someone who never stops learning, who knows what he knows and what he does not know and who is always focused on his future (and not on his past). Good developers are always keen on learning new technologies, new programming languages, etc. However it does not end there. Because programmers are practical people, they would be discontented if they are not able to apply their knowledge (Weinberg, 1971, 99). Third, in the case of small projects, Sink recommends looking for someone who can be versatile, that is someone that can do more than programming software. However, in the case of large projects and larger environments pure coders would fit in well. Finally, although Sink does not put much emphasis on coding abilities he also states that it is important to look at developers' code (samples of programmes) to assess their quality. Even if the post is not for a programmer position, for example: designer or tester, a developer would require good knowledge of programming to perform those jobs.

Having said that, software development has not only been conceived as a technical activity. For example Weinberg (1971) saw that programming is both an individual and a social activity (Weinberg, 1971, 45, 52). Programming is an individual activity because it depends on the individual technical ability of the programmer. When Weinberg did his study, in the 1960's, he found that most programmers preferred to work alone in a place were they could concentrate and not be disturbed. Things have not changed much and Weinberg's study has proved to be still accurate (Cockburn (2000b) and fieldwork data, see category C4-7 in section 6.4 of chapter 6). On the other hand, programming is also social. Programmers need to work with peers to ask for help and compare notes. Others may be turning to them too. Recently, modern agile

methodologies have been emphasising the need to work in teams and to improve communication among developers, recognising the social aspect of software development. Developers therefore need to improve their person and communication skills apart from their technical ones to be able to work in teams and deliver software.

As professionals, developers form their own opinions about their jobs. They learn to like some things and dislike others. As a consequence of this, developers experiment some emotions which can affect their work. These emotions are triggered by social interactions between them and by the use of methodologies or tools. Some of these emotions are:

- Fear of incompetence
- Longing for the old way
- Frustration over not knowing how to proceed
- Anger at the methodologists, the authors, tool vendors who claim transitions are easier than they are
- Distrust over methodologies that change every year or so
- Trapped in a way that does not make sense and an observation that the "experts" have no responsibility for the success of the project – it is the responsibility of self.

Source: Chisholm et al. (1996, 234)

The consequences of these emotions could be, for example, developers who reject a methodology, developers who apply shortcuts (for methodologies) and increase in project turnover. These consequences also would affect the job developers are doing and the quality of the software they develop.

Most studies of developers portray them as technical people. However, as human and social aspects are being recognised as relevant, the question of how to develop developers' skills to identify and deal with

them arises. Du Plooy (2003) states that the role of an information systems developer (which I translate to a business software developer), is, apart from building the software, to introduce it "in a manner that takes cognisance of the notion of the social context" of the information systems it is going to serve (du Plooy, 2003, 42). To understand the social context of their users, developers should establish a closer relationship with them. In some cases (e.g. agile methodologies) the participation of users in project teams is encouraged. Also some researchers and practitioners (Bentley et al. (1992) and Spolsky (2004)) propose the use of ethnographic techniques to grasp the nuances of their users' working environments. Fisher (2003) identifies other characteristics needed by developers to be able to establish rapport with users and among developers, and deliver usable information systems (i.e.: business software). Table 3.3 shows a classification of roles that could be performed by a developer and the related skills needed. Although Fisher's approach is technical, she raises important points that should aid the development of information systems and software that considers the human and social aspects of their users. Technical communicators are developers in charge of the design and dissemination of technical information to users. Interface and graphic designers are in charge of designing the graphical user interfaces of the software. Ergonomics studies the relationship between people and their working environments, and in this case, it studies how software fits within.

Profession	Skills
Technical communicators	Written and oral communication Understanding of users Understanding of text layout and design
Interface designer	Understanding screen layouts Design of icons Understanding users' workflow Use of colours
Graphic Designer	Understanding how to colour Design of graphics, charts, illustrations
Ergonomics	Understanding how people should work

Table 3.3 Skills needed to design human elements of systems
Source: Fisher (2003, 206)

Working in Teams

As mentioned before, software development is a social activity. Developers need to be able to relate to other developers and work in teams. DeMarco and Lister (1999) define the ideal software development team as a "jelled team". A jelled team is "a group of people so strongly knit that the whole is greater than the sum of the parts" (DeMarco and Lister, 1999, 123). Characteristics of a jelled team are:

- Low turnover during projects: team members are committed to their tasks and do not leave the team until the work is finished
- Commitment to teams is not gained by status of their positions or money but by enjoyment of their job. As Cockburn (2000b) states, good and committed developers are "good citizens", they will stick around until their job is done and the user is satisfied
- A strong sense of identity: team members recognize that they are part of a team and not individuals working alone
- Joint ownership of the product: individuals recognize that the software they are creating is not a product of their individual work but a product of the work of the team

Source: DeMarco and Lister (1999, 127)

One of the most important factors that aids the creation and maintenance of jelled teams is strong channels of communication. Having development working environments which allow physical proximity, and that allow developers to talk to each other easily, e.g. by talking face-to-face, by using white boards, by having meeting areas, etc. improve the effectiveness of their work (Cockburn, 2000a). However with the increase in popularity of global or "geographically distributed software development" (Herbsleb et al., 2005, 524) some issues emerge. Problems experienced by distributed teams are the reduction in frequency of communication, especially informal communication (Handel and Herbsleb, 2002, 1; Herbsleb et al., 2005, 524), a lack of shared context, lack of cues regarding availability and lack of trust for sharing information (Shami et al., 2004, 3). In this context the role of online, virtual environments as working and communication tools gains relevancy. (See section 4.2.3 of chapter 4 for a detailed discussion of virtual communities as communities of practice). In the literature, there are differing views as to whether multi-site software development can be done using software tools to communicate. Herbsleb et al. (2005) in a study of distributed software development report that most people preferred face-to-face contact because it allowed them to understand each other's worlds (Herbsleb et al., 2005, 530). Likewise, Shami et al. (2004) on a similar study report that developers who where located in the same place communicated and engaged better than developers who needed cross-site collaboration (Shami et al., 2004, 1). On the other hand, Handel and Herbsleb (2002) emphasise the potential value of synchronous messaging in the software development workplace as it is used "overwhelmingly for discussion or for articulation work to coordinate projects and meetings, and to negotiate availability" (Handel and Herbsleb, 2002, 8). Also Herbsleb et al. (2005) state that collaboration tools like chat and IM provide a faster pace of interaction as well as asynchronous multi-party conversations which help projects to overcome time zone differences (Herbsleb et al., 2005, 532). (See a discussion

about the linguistic and pragmatic characteristics of virtual communities of software developers in section 4.2 of chapter 4).

Other factor important for team work is amicability. Cockburn (2000a) states that "amicability is the weaker cousin to trust" (Cockburn, 2000a, 89) but an easier target to achieve. Amicability within a group reduces the chances of conflicts. On the other hand, factors affecting negatively the formation of teams are mentioned by DeMarco and Lister (1999). These are:

- Defensive Management: managers who do not trust on their developers and hinder their job.
- Bureaucracy: or paperwork. According to DeMarco and Lister (1999) paper work is a waste of developers' time. Bureaucracy will blur the real goals of a team.
- Physical separation: as mentioned before, development is a social activity in which communication is important. For DeMarco and Lister communication between developers is improved by physical proximity. However, in cases where distributed work is necessary or unavoidable communication software tools prove useful to counter distance (Handel and Herbsleb, 2002).
- Fragmentation of Time: when a developer is assigned to more than one team, it makes difficult for him to commit entirely to all the projects he is working in.
- The quality-reduced product: usually projects are reduced in budget. The practical consequence of this is a reduction in product quality which can detriment the morale of developers.
- Phony deadlines: similarly to the previous point. Making developers work impossible deadlines will lower their commitment.

Source: DeMarco and Lister (1999, 133-138)

Individual and group characteristics of developers influence the way they work and therefore the quality of the software they develop. Unfortunately, most of the focus in the study of software development is on its technical aspect and reject the human and social aspects. Where it considers human and social issues it tends to be of the user domain rather than the developers' domain. Considering developers' human and social aspects in methodologies and team formation may improve their performance. Making developers work in teams may help them to accept users working with them as well. This in the end will facilitate the recognition of social and human factors relevant to the software they are developing, an issue this thesis is concerned with.

3.3 Chapter Summary

In this chapter I have explored human and social issues surrounding software development. I have focused on issues emerging from software as a product and from the development environment. Business software, though a technical concept, resembles at different levels, some of the human and social characteristics of the organisation which is using it. For example, issues such as considering software to support more flexible processes needed by organisations (fudgeability), or making software to adapt to the style of managers and decision makers or considering the group (as opposed to individuals) as the user of software were discussed in this chapter. Other external factors also influence software and the way it is developed. Factors such as pressures from the market and limited perception of developers would affect the quality of the software. Other human and social issues within the development environment will also influence software. I have stated that software development is shaped by the characteristics of the development environment and the people who are going to develop it, in addition to the characteristics of the target organisation which is going to use it. For example, adaptability in software may depend on how developers are able to foresee future changes and how they are able to write code which can be changed

easily. Software development methodologies are evolving to take into account these characteristics. For example, the agile methodologies are designed to account for adaptation in business organisations by providing means for adaptation within the development environment. Thus I have presented three perspectives of software development which are shown in figure 3.7. Software development has long been seen as an engineering discipline which puts emphasis on processes and productivity. However, at considering more the human and social aspects of developers and the development environment, software development could be seen more as a craft that cannot be as structured as engineering. Finally, software development can also be seen as a cooperative game which puts emphasis on people rather than on processes.

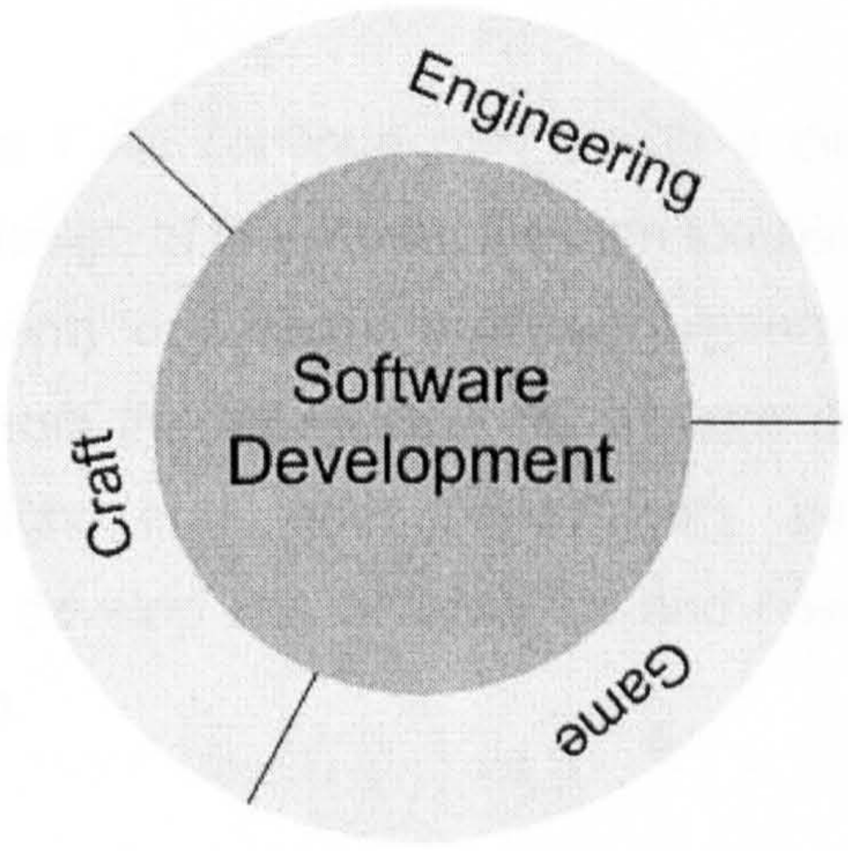


Figure 3.7 Three perspectives of Software Development

I have also explored some of the individual and team characteristics of developers, which affect how their work is organised. I have also implied that the development environment presents characteristics comparable to the ones in the model of complex organisations presented in section 1.1 of chapter 1. These characteristics are for example, that development environments are unpredictable environments due to the nuances of human nature (developers) and the mixture of different specialists. Development environments are also adaptable, as they have to change

according to the project in hand and the kind of software they are designing. Finally, group work is done within development environments. As explained software development is also a social activity and developers are organised in groups according to their expertise. These groups at the end have to be coordinated and commanded, possibly, as groups are coordinated in business organisations. The conclusion I would like to make here is that in addition to the nature of business organisations and the needs of their members, software could also be shaped by the characteristics of the people who are developing it. Thus this thesis wants to explore also how the human and social aspects of the software development environments affect software development, particularly in the cases in which the target organisation contains unpredictable, adaptable, and multi-group work processes like those ones contained in decision making processes, workflow and collaborative work.

In the next chapter I will continue part II with a discussion about the methodology and design of this thesis. As I am focusing on software (and software development) for complex business organisations I will explore this phenomenon from the perspective of software developers. This will give me an insight into what practitioners perceive about the organisations they develop the software for and how the software they develop fits in there.

Chapter 4: Research Methodology

In chapter 1 I exposed an information systems and software development problem. Human and social issues, such as unpredictability and adaptability, emerge from organisations where decision making, collaborative work and similar abound. These issues are difficult to understand by developers using their available frameworks and thus are not considered when developing software. In chapter 2 I explored a series of views on organisations and information systems which try to explain their unpredictable and adaptable nature. Chapter 2 showed how human and social aspects within business organisations shape the context where business software is used and software development happens. In chapter 3 I examined some literature on software and software development to see how human and social aspects of organisations are dealt with. In this examination I came across some human and social issues within the software development environment that also affect developers work. The concern of this research is to connect the issues discussed in chapters 2 and 3, and to see how software developers perceive human and social issues in complex organisations and how these perceptions are affected by their own environment and their way of working. To carry out an investigation of this issue I designed a research methodology which I will explain and justify in this final chapter of part II.

The research design is guided by a qualitative, interpretive paradigm, specifically by the phenomenological and hermeneutical perspectives. The proposed method is online ethnography of virtual communities of software developers. The interpretive approach allows me to access software developers' beliefs or interpretations about software development, the software they produce and the organisations they develop the software for. By using an interpretive paradigm I can then build up my own interpretations based on software developers' interpretations. Software developers' views about their jobs provide an

original and practical perspective on software development to this thesis. Software developers are the people who do the job. They are involved in the software development process; they possess knowledge about complex organisations, their users and their problems. Software developers experience first hand how their software is used and are able to assess its quality and the extent to which it is helpful to their users. Software developers possess the practical know-how of software development methodologies and know which techniques are successful in practice. The online ethnographic method allows me to contact software developers in their virtual communities. Virtual communities are "places" where real work happens, which are familiar to software developers and where I can easily approach them. Virtual communities of software developers are social gatherings where members conduct and exchange their working experiences in the software development field. In this thesis, the concept of communities of practice (CofP) is also used to explain how virtual communities are formed by the interactions and relationships their members are engaged in when they "practice their skills through tools, frameworks, idea sharing artefacts or documents" (Coakes and Clarke, 2006a, 92) (see section 4.2.3 of this chapter). Ethnographic interviews in the form of conversations within online forums, weblogs, instant messaging and e-mail, and archival research in online sources are used to observe developers and gain insight into their current practices, problems and personal opinions. Data collected from the online ethnography are analysed according to the interpretive paradigm. Two data analysis methods are combined to account for the nature of the online data and to guide their interpretation. The first method is content analysis which guides the classification of data in segments of text that represent the purposes of the research. The second method is conversation analysis which I use to make sense of the structure of online conversations as they follow different rules from written and oral speech. To give the data a comprehensive shape, an interpretive and inductive theory building strategy follows data analysis. The result of the theory building stage is a model that describes and explains human and social issues in software development for complex business organisations. This

model corresponds to the aim of the research and answers the research questions which are also presented in section 4.5 of this chapter.

The structure of this chapter is shown in figure 4.1. This chapter is divided into seven sections. The first, second and third sections introduce the interpretive approach as the paradigm for this thesis, virtual communities as the field in which this study is conducted and ethnography as the method to collect data. The fourth section describes the data analysis technique. The fifth section re-introduces the research aim and research questions which the theory building strategy, explained in section six, answers. Finally the chapter is summarised in section seven.

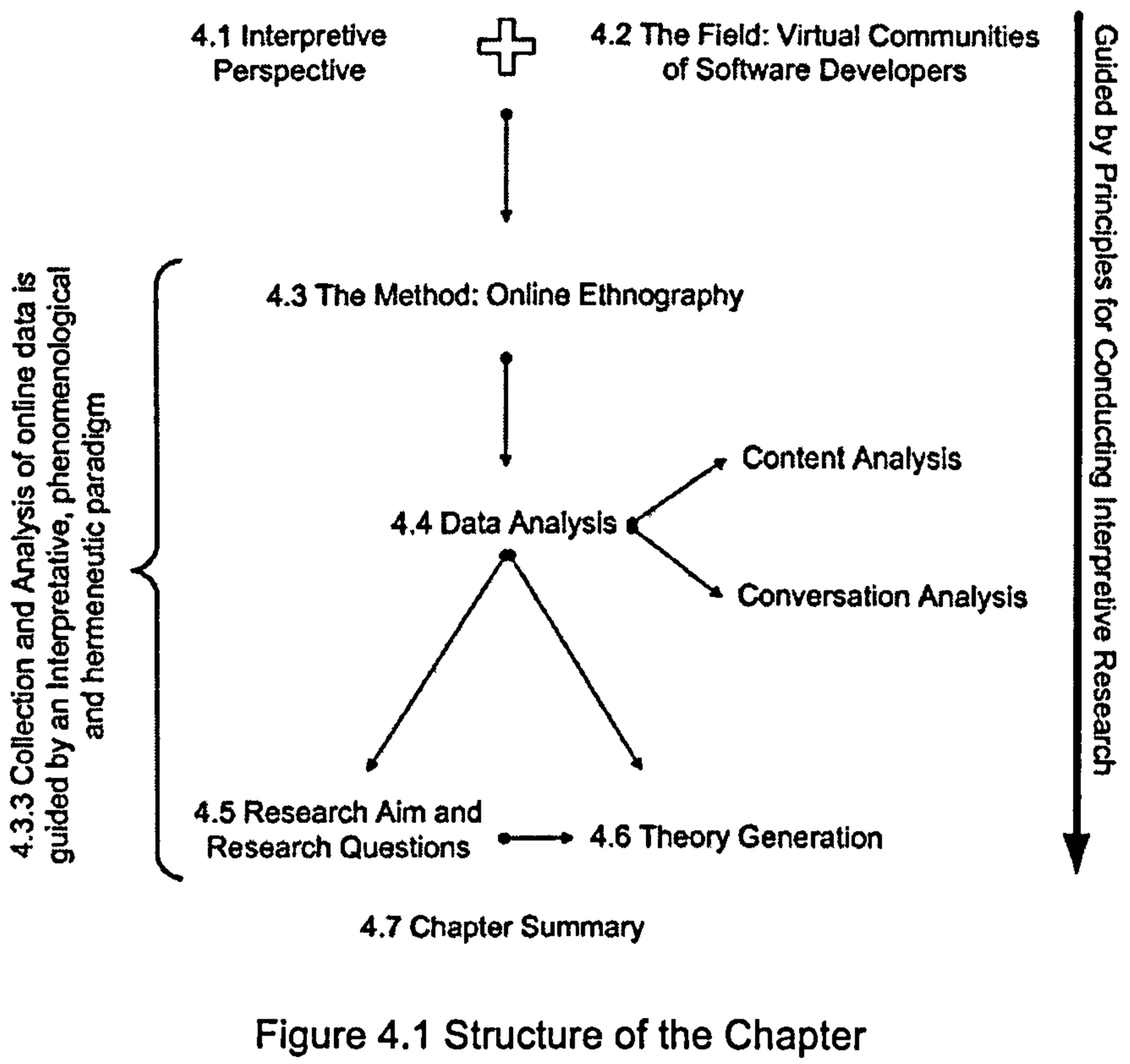


Figure 4.1 Structure of the Chapter

4.1 The Research Paradigm: The Interpretive Approach

What is a research paradigm?

Research is a "voyage of discovery during which the researcher learns much about research methodologies as well as about the subject being researched" (Remenyi et al., 1998, 27). A researcher chooses a variety of methods, techniques and tools to dig into a problem, collect data, analyse and interpret them and develop conclusions. These methods let the researcher approach the subject of study in such a way that its relevant features are uncovered and the credibility of the outcomes is assured (Williams and May, 1996, 7). The selection depends on the personal inclination of the researcher, the philosophical assumptions and the nature of the research problem (Yin (1981) quoted in Remenyi et al.(1998, 107)). Whatever the approach chosen is, the philosophical assumptions should be "clearly spelt out" (Remenyi et al., 1998, 30) to prove that "the approach to the research has been sound" (Remenyi et al., 1998, 27).

Philosophical assumptions comprise a set of concepts, such as ontology, epistemology and methodology, which explain how the researcher sees the subject of study and how he intends to study it. Ontology is the branch of philosophy which studies the being (Dictionary of theories, 1993, 835). Ontology "differentiates between 'real existence' and 'appearance' and investigates the different ways in which entities belonging to various logical categories (physical, objects, numbers, universal abstractions, etc.) may be said to exist" (A dictionary of philosophy, 1984, 255). Roughly, there are two opposite perspectives within ontology. The first one is realism, which states that "things in the world have a real existence independent of our thoughts about it" (Williams and May, 1996, 42). The other perspective is idealism which views the external world as a "product of the mind" (Williams and May, 1996, 42).

Epistemology studies the nature of knowledge (The Cambridge dictionary of philosophy, 1999). Epistemology raises questions about "who can be a knower" (Stanley and Wise, 1990, 26) and "how and when reliable knowledge is attainable" (Tice and Slavens, 1983, 313). Research epistemology could be classified within a spectrum of beliefs which goes from the positivist on one extreme to the interpretive or anti-positivist perspectives on the other. (This thesis' epistemology and ontology is explained in the next section called "The orientation of this research"). Positivist scientists seek to "explain and predict what happens in the world by searching for regularities and causal relationships between its constituent elements" (Burrell and Morgan, 1979, 5). For them the world is external and objective and the observers (i.e. the researchers) are independent (Carson et al., 2001, 1). From this point of view, observation of phenomena "must be neutral and uncorrupted by theory, if the verification principle is to hold" (Williams and May, 1996, 28). This point of view has been disputed within the social sciences. As for example, positivists "may argue that people react to their environment much as molecules which become 'excited' when heat is applied to a liquid" (May, 1997, 9). According to anti-positivist social scientists, human beings behave according not to causal relationships but socially constructed values. Also anti-positivists believe that "each situation is seen as unique and its meaning is a function of the circumstances and the individuals involved" (Remenyi et al., 1998, 33). Meanings and values need not to be described but understood in their contexts. In order to gain this understanding, researchers immerse themselves in the social world they are studying so as to "hear, see and begin to experience reality as the participant" (Marshall and Rossman, 1989, 106). This gives researchers first hand data which is then interpreted to comprehend social phenomena.

Methodology "refers to the procedural framework within which the research is conducted" (Remenyi et al., 1998, 30). A research methodology provides the researcher with guidelines to select methods

and tools and to justify this selection. A research paradigm is comprised by an epistemology and ontology which are the assumptions about the nature of the object of study and which ground the chosen methodology and back the outcomes of the research.

Postivist and anti-positivist perspectives can be and are used in Information Systems research. However the views on organisations and information systems, and the methods applied differ. For positivist IS researchers "organisations are understood to have a structure and reality beyond the actions of their members" and the research has to "discover" this reality by applying precise measures that will uncover the dimensions of reality that interest the researcher (Orlikowski and Baroudi, 1991, 9). An example of a positivist research in the field of IS is the one by Eom et al. (1998) which studies the tendency in Decision Support Systems (DSS) implementation and use within 1988 and 1994. For their study, a DSS is an objective, computer-based interactive system that supports decision-makers (Eom et al., 1998, 109). Eom et al. (1998) research uses quantitative surveys and statistical tools to conclude that creative applications of optimisation and suggestion model-based DSS are growing in popularity against simulation-based applications. On the other hand, the anti-positivist perspective assumes that the social world is not given. "The social world is produced and reinforced by humans through their action and interaction" and organisations do not exist apart from humans (Orlikowski and Baroudi, 1991, 14). The anti-positivist researcher cannot measure reality but only understand it by getting inside the world he is studying. An example of an anti-positivist research in IS is the one by Bandow (1998) which studies trust within dispersed groups of systems developers. For Bandow's study, systems development organisations are integrated work teams which are constrained by people. Tension, conflict and trust are components of the working relationships among the developers. The present research uses qualitative interviews to collect information. It then generates categories and themes from data which reflect the experiences of systems developers who work in dispersed groups and how trust develops and

shows up among them. Another example of an interpretive research stance is the study of CASE tools by Orlikowski (1993). For Orlikowski CASE tools are a form of "technology-based organisational change" (Orlikowski, 1993, 310). Orlikowski uses qualitative unstructured and semi-structured interviews to collect data from the staff of two organisations which have implemented CASE tools. Following an inductive theory building strategy, Orlikowski creates a theoretical framework that addresses the organisational issues that emerge from the adoption and use of CASE tools.

According to Mingers and Willlcocks (2004) most IS research stems from a positivist paradigm (Mingers and Willlcocks, 2004, XIV). This concurs with a study made by Orlikowski and Baroudi (1991) thirteen years before, which found that most IS research articles that they examined shared the same positivist philosophical background. (The articles were published between the period of 1983 and 1988 in four major information systems journals.) Thus the tendency of IS research towards positivism seems very slow and hard to change. Mingers and Willlcocks (2004) and Orlikowski and Baroudi (1991) agree in that this "narrow focus" (Mingers and Willlcocks, 2004, XIV) is "unnecessarily restrictive" (Orlikowski and Baroudi, 1991, 1) and argue for a wider range of philosophical assumptions to inform IS research. This thesis follows the recommendation of these authors and chooses an anti-positivist stand which is explained in the following sections.

The orientation of this research

The present thesis is located within the social sciences. This thesis believes that information systems are, "fundamentally, social rather than technical systems" (Hirschheim, 1985, 13). Similarly, the activity of software development is thought of as a social activity (Weinberg, 1971, 45) which is interpreted by the software developers who also interpret the reality of the information systems they are working with. Also, this thesis sees every single social setting (such as an information system or a

software development environment) as unique because of the changing nature of humans and their social interactions and the different interpretations that people give to it. Similarly, the ontology of this thesis is Idealist. That is that this thesis believes that reality is subjective in nature and that it is created by the interpretations of people. Following this, the present thesis' epistemology is **Interpretive** as I believe that phenomena can be known through the interpretations of people involved with those phenomena. This thesis is "devoted to developing an understanding of human systems" (Savenye and Robinson, 2001, 1172) and is done in "natural settings" with the researcher as the "main data gathering instrument" (Savenye and Robinson, 2001, 1175). Thus the outcome of this thesis will be a result of my interpretations of the interpretations that the subjects of study have of the problem of software development for complex business organisations. This strategy is aligned with the tendency within research in the Information Systems field which is switching from a focus on technological issues to a focus on human and social aspects in organisational settings, such as managerial and organisational issues (Myers, 2003, 1).

The interpretive epistemology addresses the world from "the point of view of the people studied" (Hammersley, 1992, 165). Beliefs about the external world being a "product of the mind" (Williams and May, 1996, 42) play an important role. Since the world is constructed in peoples' consciousness, including researchers' minds, the concern is on the "meanings that people give to their environment, not the environment itself" (May, 1997, 13). This is one of the most important distinctive aspects in social research. The interpretive approach is also concerned with gaining an in-depth understanding of the social world and with interpreting "the meaningful character of social action" (Travers, 2001, 8). In-depth understanding requires the researcher to immerse himself in the phenomena to be studied (Carson et al., 2001, 65). Instead of drawing a line between researcher and phenomena, interpretive researchers promote engagement between them and their subjects of study (May, 1997, 20). The engagement must be reported transparently so as to

describe the conditions in which the results were found. This adds a touch of subjectivity to the research which in the interpretive circle is welcome and appreciated. In the case of this thesis full immersion and engagement with subjects of study were achieved via participation in online communities of software developers and hence becoming *part* of their communities of practice.

According to Denzin and Lincoln (1998) "every researcher speaks from within a distinct interpretive community, which configures, in its special way, the multicultural, gendered components of the research act" (Denzin and Lincoln, 1998, 23). The interpretive perspective recognises that researchers carry with them a set of values which will guide their work. For May (1997) values "are a fundamental part of human condition" and cannot be eliminated (May, 1997, 44,45). Thus bias in interpretive research is not hidden but acknowledged. Researchers "seek to be aware of their own personal and cultural prejudices" so as to "become conscious of those things that are influencing them" (Spradley and McCurdy, 1972, 15).

Criteria for guiding and assessing the quality of Interpretive Research

Interpretive research is assessed via a variety of constructs from which the most important (and most used) are reliability, validity and credibility. For Perakyla (1997) "checking the reliability is closely related to assuring the quality of field notes and guaranteeing the public access to the process of their production" (Perakyla, 1997, 203). This study's data was collected from online public forums and weblogs where software development is discussed by software developers (see section 4.2 of this chapter for a discussion of virtual communities of software developers). Online data were collected when they exposed human and social aspects surrounding software developers' work (as opposed to pure technical discussions) which is the concern of this thesis as reflected by this thesis' research aim and questions (presented in chapter 1). The web addresses

for those online sources are provided in appendix 6. Some of the discussions in which I participated and which I collected may no longer exist online (probably due to storage limitations) at the time the reader is reading this thesis. Most online forums show their most recent contributions on top of their lists and a number of previous posts for some time available online. For this reason I kept an electronic copy of every online document I used for this study. However due to space limitations of this document this material is not included completely. Instead, segments of relevant data are presented in chapters 5 and 6 to explain the data collection and data analysis processes, and to provide an exposition of this study's findings. Wherever a segment of text is given the name of the online source, the title of conversation and the date in which the conversation took place is shown. This information can be used in an internet search engine (such as Google) or the online forum's search engine (web addresses of online sources are given in appendix 6) to find the complete discussion (if still available online).

Perakyla also states that "the validity of research concerns the interpretation of observations" (Perakyla, 1997, 207), in other words, it concerns "whether the researcher has gained full access to knowledge and meanings of respondents" (Remenyi et al., 1998, 115). Additionally, Mason (1996) states that judgements of validity are "judgements about whether you are 'measuring', or explaining what you claim to be measuring or explaining" (Mason, 1996, 146). This study collected data from virtual communities of software developers, i.e., online sources where software development issues are discussed. Virtual communities of software developers are places where actual software development work is done and discussed and thus are places to access valid data and knowledge about software development (see section 4.2 of this chapter for a discussion of virtual communities of software developers). The way in which software development work is shown is by the online interactions between participants. Human and social aspects of software development are expressed through online texts showing participants' thoughts, experiences, problems and successes at work. Although there are some

issues regarding the anonymity of participants and the lack of visual and social cues of the online media, virtual communities are places in which communication is possible and where people can convey and infer meaning (section 4.2.2 of this chapter discusses these and other issues in virtual communities). Also, the process of interpretation of the online meanings that I carried out is explained throughout chapter 5, 6 and 7. Chapter 5 explains how I did categorisation of data arriving at interpretive generalisation. My interpretations are shown in chapter 6 as a descriptive picture showing the current state of affairs (as it was during the fieldwork) in software development and in chapter 7 as a model of social issues of software development. In addition, section 5.1.2 of chapter 5 discusses my use of probe questions to establish the relevance of some of my interpretations. Probe questions are asked to different people to see if there is consistency with their understanding of a particular issue and with my interpretation of that understanding. Probe questions provide validity and reliability to this study's data.

The credibility of a research is measured by the extent to which it was designed in a manner that identifies and describes the phenomenon (Remenyi et al., 1998, 115). Credibility is gained when the researcher manages to prove that he has identified and thoroughly described the subject matter e.g. human and social aspects of software development for complex organisations. It indicates the level of immersion and involvement of the researcher in the fieldwork. Therefore, researchers have to prove their credibility by ensuring "that the subject was accurately identified and described" (Marshall and Rossman, 1989, 145). The online ethnographic method (discussed in section 4.3 of this chapter) provided full immersion and involvement in virtual communities' interactions. The topics of the discussions were all related to software development and reflected their human and social aspects, as shown in part III of this thesis.

There are more constructs that are used to measure qualitative research and which inform this thesis. These are transferability, dependability and

confirmability (Lincoln and Gubas (1985) quoted in (Marshall and Rossman, 1989, 145)). The transferability of a research indicates how one investigator can make use of results from previous inquiries. This is a difficult task, as the changing character of the social world makes it almost impossible to have two similar settings from which to collect data. Nevertheless, this can be done, under the responsibility of the researcher, who has to be able to justify the decisions satisfactorily. Transferability can be also understood as the ability to generalise the findings in a small setting of population to a bigger one. However, in the interpretive perspective, generalization is achieved not by the statistical representativeness of the data collected but by the logical reasoning used to relate the results. A theoretical discussion of the interpretive process of generalisation is provided in section 4.6 of this chapter. The actual process of generalisation is shown throughout chapter 5, by explaining the way I carried out the data collection and analysis and by providing examples showing how fieldwork data were transformed into abstract concepts and propositions. This study's final outcome is a result of this process of interpretive generalisation and therefore this thesis claims that the Model of Social Issues in Software Development (see chapter 7 for a discussion of the model) could be transferable to other software development situations different from decision-making and collaborative software.

Dependability is established when the researcher accounts for "changing conditions in the phenomenon chosen for study as well as changes in the design ... of the setting" (Marshall and Rossman, 1989, 147). As every setting is unique and as the engagement and the values of the researcher contribute to make it different from any other, a detailed and methodical account of every event has to be provided. This thesis' setting – virtual communities of software developers - is described in section 4.1 of this chapter. Section 4.1 also discusses the peculiarities of the online media and stresses the advantages and limitations of CMC (computer mediated communication) for this study so as to justify the design of the data analysis strategy.

Confirmability, stresses whether the findings of the study could be confirmed by another. This again is a very tricky issue as "each situation is seemed as unique and its meaning is a function of the circumstances and the individuals involved" (Williams and May, 1996, 34). Therefore a different kind of assessment is needed. The examination of the philosophical foundations, the selection of methodologies, the accuracy of the interpretations and transparency of the whole process are vital. In section 8.4-b of chapter 8 I propose directions for future research by transferring and confirming the results of this thesis.

In addition to these constructs Klein and Myers (1999) have designed a set of seven principles which can be used to guide and assess the conduct of interpretive researches, especially those guided by the phenomenological and hermeneutic perspectives and those which use ethnography as the method for enquiry. For Klein and Myers (1999) these principles are "fundamental ideas" because they are derived from philosophical writings considered as classical contributions to the interpretive perspective (Klein and Myers, 1999, 70, 71). The purpose for creating these principles is to respond to the call to discuss explicitly the criteria for judging interpretive research (Klein and Myers, 1999, 68). As opposite to positive research in which there are long accepted guidelines for conducting and assessing research (Klein and Myers, 1999, 68) interpretive researchers have lacked of this kind of guidance. Klein and Myers' (1999) seven principles are designed to guide interpretive researchers in IS and to assess the quality of their research (Klein and Myers, 1999, 67). Klein and Myers (1999) created these principles for the phenomenological and hermeneutical branches of the interpretive approach and to target field studies such as ethnography. These principles are suitable to assess this thesis as it follows an interpretive, phenomenological, and hermeneutical paradigm (in section 4.3.3 of this chapter I explain the process of interpretation of social issues for this thesis which is consistent with the phenomenological and hermeneutical perspectives), and uses an ethnographic method to collect data. Also

according to Klein and Myers (1999) the fulfilment of these principles helps the researcher to justify the implementation and outcomes of the researchers' thesis (Klein and Myers, 1999, 87). Also Klein and Myers (1999) state that their principles should not be followed blindly to the line but that the researcher should select from these principles the ones that suit his research. The principles are shown in table 4.1.

1. The Fundamental Principle of the Hermeneutic Circle
This principle suggests that all human understanding is achieved by iterating between considering the interdependent meaning of parts and the whole that they form. This principle of human understanding is fundamental to all the other principles.
2. The Principle of Contextualization
Requires critical reflection of the social and historical background of the research setting, so that the intended audience can see how the current situation under investigation emerged.
3. The Principle of Interaction Between the Researchers and the Subjects
Requires critical reflection on how the research materials (or "data") were socially constructed through the interaction between the researchers and participants.
4. The Principle of Abstraction and Generalization
Requires relating the idiographic details revealed by the data interpretation through the application of principles one and two to theoretical, general concepts that describe the nature of human understanding and social action.
5. The Principle of Dialogical Reasoning
Requires sensitivity to possible contradictions between the theoretical preconceptions guiding the research design and actual findings ("the story which the data tell") with subsequent cycles of revision.
6. The Principle of Multiple Interpretations
Requires sensitivity to possible differences in interpretations among the participants as are typically expressed in multiple narratives or stories of the same sequence of events under study. Similar to multiple witness accounts even if all tell it as they saw it.
7. The Principle of Suspicion
Requires sensitivity to possible "biases" and systematic "distortions" in the narratives collected from the participants.

Table 4.1 Seven principles for Interpretive Field Research
Source: Klein and Myers (1999, 72)

An evaluation of this research using these principles is shown in section 8.3 of chapter 8 of this thesis.

4.1.1 Justification for the selection of approach

Having briefly exposed the philosophical assumptions of this thesis, and therefore set the scene of how it will be performed and then evaluated, in this section, I will summarise and justify the reasons for choosing the interpretive approach. The following list is complementary to the ontological assumptions presented earlier in this chapter (sub-section entitled "the orientation of this research").

- The ontology of this thesis (and hence mine) is idealist. For this research, business organisations, information systems, software development and software are products of people's interpretations and are investigated through in-depth fieldwork, such as ethnography, in the places where the people involved with them interact like for example virtual communities.
- Interpretive research allows the researcher to investigate a phenomenon from the point of view of a (group of) stakeholder(s). From the many perspectives available (in this case, for example, managers in business organisations, staff in business organisations, software project managers, etc.) I have chosen to investigate software development for complex organisations from the perspective of software developers. They are part of complex environments and as I am studying software development I believe that their practical perspectives give original insights to my investigation. Figure 4.2 illustrates this.

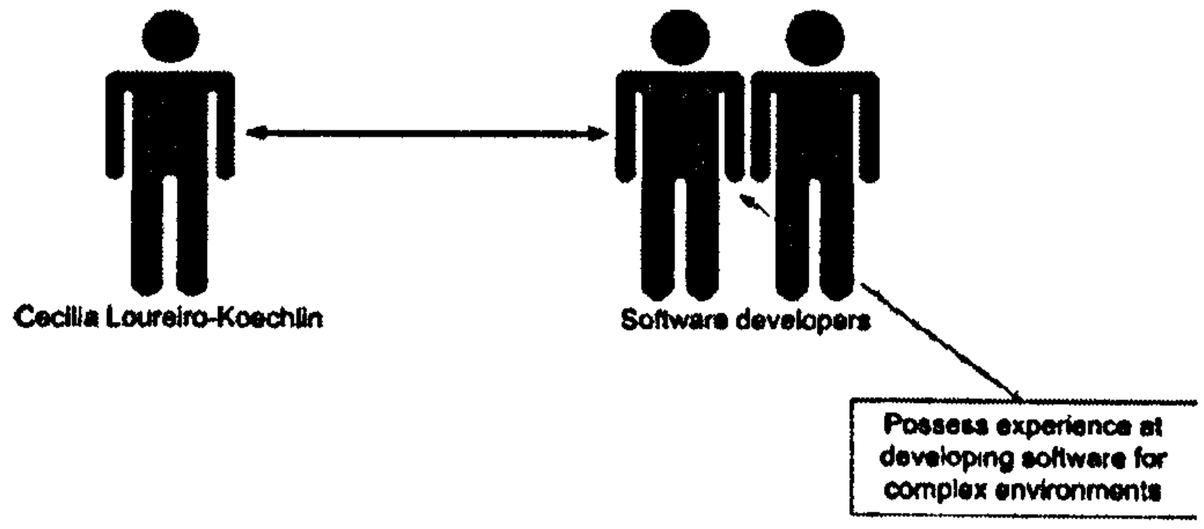


Figure 4.2 Interpretive Research from the Perspective of Software Developers

- Hence an interpretive perspective allows the researcher to explore how developers see and deal with complex situations (i.e. what they actually do and why) and how they perceive their work in relation to the complexity in the organisations they develop the software for.
- An interpretive approach accounts for a subjective perspective. The outcome of the interpretive research is a result of my views of the views of software developers about software development for complex situations.
- The interpretive approach lets (and requires) the researcher expose his intuitive, religious, political and philosophical views so as to justify and clarify design decisions and the outcome of the research. Thus a declaration of values is presented in the next section.

4.1.2 Declaration of Values

> "If the sociologist manages to produce any truth, he does so not despite the interest he has in producing that truth but because he has an interest in doing so - which is the exact opposite of the usual somewhat factous discourse about neutrality"
>
> Pierre Bourdieu quoted in May (1997, 45).

An interpretive approach requires the researcher to immerse himself and be conscious of his own values. This is summarised in section 4.1.1. The above quotation suggests that in every research undertaking (not only in the social sciences) value judgements are embedded. This is also valid in situations where researchers declare themselves neutral. Neutrality is also a value judgement. The interpretive approach recognises the value content of research. It encourages researchers to expose their values and to be aware of them during the process so as to be more conscious of their own biases. In the light of this I am now presenting my values and views of the research presented in this thesis.

In chapter 1 I presented the motivations that led me to this thesis. These are professional and academic interests and delineate my research. My background is in industrial engineering. I also studied a master in systems engineering as I was interested in the industry of software development. I enjoyed very much modelling and working with databases. As my career developed, I became responsible for managing projects of information systems development. My interests broadened into understanding the organisational aspects that surrounded the implementation of these projects. I then went to do a MBA in Information Management which complemented my views of information systems projects and shifted my perception from being concerned with the technical aspects of information systems towards an interest in the human aspects of the organisations for which the information systems are designed. After some further professional experience in Information Systems Development I decided to study a PhD in order to address the following interests:

- To learn and understand development of information systems and in particular software from the human and social perspectives
- To contribute to the body of knowledge of the IS field by providing insights into the work of software developers
- To establish the foundations for future professional career as a consultant and researcher in software for information systems

As a researcher, I consider that my role was to explore and participate to a certain extent in discussions about software development for information systems. My intention was to not become engaged in altering the course of any software development endeavour by influencing those who are responsible for projects. Neither did I want to assume some responsibility for the delivery of any product. Thus the focus of my research was on enquiring about how people make sense of their work as it currently is. However, the hope of this thesis is that the results of this

research will lead to improvements in future software development projects. To guide this process of enquiry I came across two possible strategies. First to make sense myself of the complexity of environments with which systems designers have to deal with; second, by understanding such complexities through the perceptions of designers. I have chosen the second one because of the following reasons:

- Software designers are part of these complex situations and therefore their views are highly valuable
- I can appreciate the strengths and weaknesses of these views in achieving success or failure
- My own perspective is closely related to the practice of Software and IS development
- This would make an original contribution to my work

Additionally, during the last few years of my professional experience, I used online tools as part of my work to communicate with peers. I was part of a network of systems analysts and developers within and outside the organisation. We were always connected by e-mail and instant messaging. Although some of us were not physically far away from each other, in some situations we preferred to talk online. The reason is that most of the topics we talked about were related to the things we were doing in our computers. Online tools allowed us to share programme codes, error messages and programme outcomes; and work at the same time. We also used to contact users and software providers through this media. This was a fast and efficient way to approaching them, at least for most of the everyday-activities. Of course, sometimes we met with people in person for more formal conversations.

Having discussed that online environments are places where IS and software work is done (i.e.: online environments correspond to online communities of practice) I considered this and decided to carry out my interpretive research within virtual communities of software developers.

The method for this inquiry was online ethnography which shares some similarities with traditional ethnographies but possesses some special features which are addressed in section 4.3. Both virtual communities of software developers and online ethnography are discussed further in the next sections.

4.2 Virtual Communities of Software Developers

Having laid out the main interpretive tenets of my research approach, I now proceed to explore the concept of virtual communities since it is essential for my methodology. This section explores the characteristics of the online computer environments (the hard side) and stresses the motivations and social interactions among information systems developers (the soft side). There are two definitions which reflect both sides of the virtual communities in which I did my fieldwork. The first definition is by Lee et al. (2003) and emphasises the use of computers:

> "a cyberspace supported by computer-based information technology, centered upon communication and interaction of participants to generate member-driven contents, resulting in a relationship being built up" (Lee et al., 2003, 51)

The second definition is by Rheingold (1994) and focuses on the emotions and relationships of the participants:

> "Virtual communities are social aggregations that emerge from the Net when enough people carry on those public discussions long enough with sufficient human feeling, to form webs of personal relationships in cyberspace." (Rheingold, 1994, 5)

From the above definitions, the "virtual" component in virtual community represents the use of networked computers and software tools used to communicate. The second part, "community" represents people with similar interests, who gather for entertainment, to discuss or ask for help.

However, the mere act of gathering online does not imply that a community exists. There are more factors that are needed for a community to be created. There needs to be a "desire and need to share problems, experiences, insights, templates, tools and best practices" (APQC (2004) quoted in Coakes and Clarke (2006b, 92)). Also time is needed to allow people to know each other. Because online participants do not meet face-to-face it takes longer to develop relationships. Nevertheless, relationships are built and thought as genuine by online members (Parks, 1996, 2). As in real societies norms are created to show the members of virtual communities how things are done there (Elliott and Scacchi, 2003, 21). Social evolution speeds when these norms are challenged and changed as new people join the group (Rheingold, 1994, 2).

There are two main elements that keep virtual communities alive. The first one is a shared interest or common purpose, such as talking about computer games. Virtual communities of information systems or software developers are grounded on their "occupational communities" as they share "similar goals, work practices, beliefs interests, and value systems" (Elliott and Scacchi, 2003, 24). The purposes of these communities would be to provide or ask for technical support or to discuss practices. The second element is a sense of membership shown by people's commitment to participation, solidarity of the group and a unique identity. For example, the number of replies that an enquiry receives would give an idea of the level of commitment and solidarity of the group with its members. The way outsiders are treated and the differences between them and insiders reveal the identity of the group.

Despite the scepticism of some scholars who think that online gatherings are a "socially-impoverished domain" (Baym, 1995, 26) people are using the net more to engage in virtual relationships. There is evidence to prove this. In her study about humour Baym (1995) shows how people create a virtual community to talk about soap operas. Rheingold (1994) provides too a rich picture of a virtual community devoted to share

medical and family problems. There is also evidence of communities created for work purposes. A virtual community of journalists is studied by Millen and Dray (2000) and a group of public relations professionals is researched by Thomsen (1996). Communities of information systems developers have also been investigated. Elliott and Scacchi (2003) study a virtual organization of Free Software developers that successfully collaborate and resolve conflicts online. The members of this community share the same values about the use of free software and freedom of choice which are manifested in their online conversations and influence their practices. Since virtual communities are places where actual work is done (as opposite to leisure or personal interests) the concept of communities of practice can be applied. Communities of practice are defined by the practices their members are engaged in. More on communities of practices is discussed in section 4.2.3 of this chapter.

I have classified virtual communities of software developers in a) open and b) close communities. Open communities are created upon personal, work or professional interests (Millen and Dray, 2000, 167). Members are geographically dispersed and sometimes never meet each other. To communicate, these communities use free, public online settings such as bulletin boards, online forums or mailing lists which are sponsored by public or private organisations and which usually run in a browser (see glossary). The motivations of these people to take part are often centred on technical support and theoretical discussions about technologies, software tools, programming languages, etc. The access to these environments is not difficult as they welcome new members who share their interests. The difficulties however arise at looking for the appropriate communities for the fieldwork. Once found, the challenge is to get known and establish rapport with the other members. Closed virtual communities possess members who only interact with people they already know (in person). Although members could be geographically dispersed, these communities are grounded mostly on physical organisations (Millen and Dray, 2000, 166) like for example software development houses, and their membership is restricted to its members.

Access to these environments could be difficult as permission will be needed by the organisation or a member of the community. Communication is done by e-mail, instant messaging or systems provided by the organisation. The purposes of these communities can be divided in three: 1) to perform or coordinate task related activities, 2) to broadcast information and 3) to socialise (Kettinger and Grover, 1997, 517).

4.2.1 Selection Criteria for Virtual Communities

For this research I opted for open communities as their public nature facilitates access to a wider cross organisational and culturally varied people (software developers with different professional backgrounds who work at different organisations and who practice under different paradigms). See figure 4.3. Also information obtained through these types of communities was not only technical but provided insights into the human and social aspects of software development within different contexts such as the software development environment and the organisational settings.

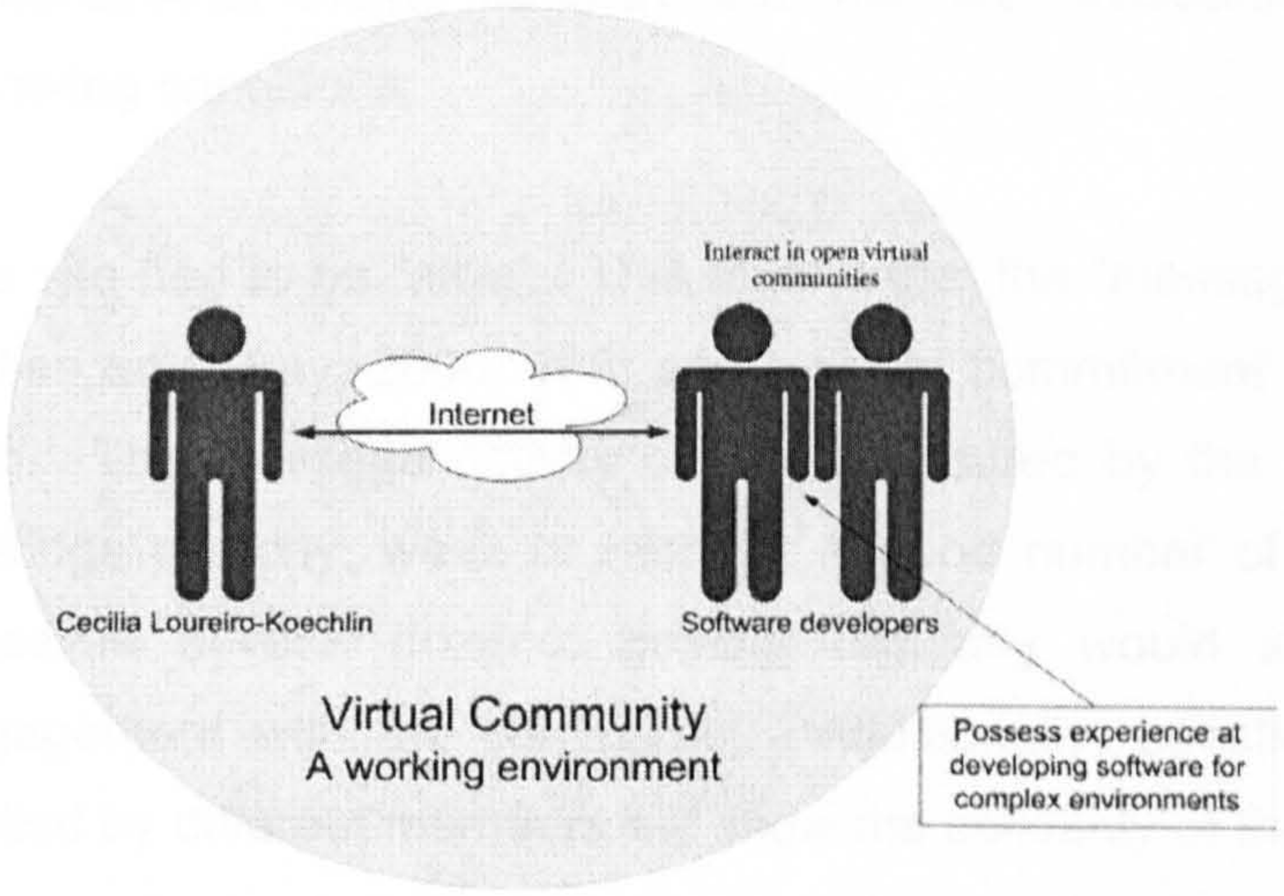


Figure 4.3 Interpretive Research of Virtual Communities of Software Developers

To search for open, virtual communities I designed a system which comprised two search criteria. The first criterion was the kind of web site which supports online conversations. Some of the keywords I used were Newsgroups, Discussion Forums, Mailing Lists, Blogs/Weblogs and Wikis. The second search criterion was the topic of discussion. Topics had to be related to the topic of this thesis: the development of software for adaptable, unpredictable, multi-group environments. As I was targeting software developers I chose to use technical terms (most of which I refer to in chapter 2) like for example:

- Software /Systems development
- Software /Systems programming
- Decision support systems (DSS)
- Group decision support systems (GDSS)
- Collaborative systems
- Collaborative work
- Workflow

I also looked for specific brands of products that develop these kinds of technologies such as Microsoft. In addition to this, I also used some filtering criteria. After finding the online communities I evaluated the sites for the following conditions:

- The site had to be "alive". This means that the "message activity" (Millen and Dray, 2000, 168) and level of commitment should be high. The message activity can be measured by the density of postings per day, week or month. A good number of members (possible several dozens) posting regularly would show their engagement with the community. Additionally, questions being replied by different members will show the solidarity of the group.
- The content of the postings should be related to software development and its human and social aspects. Site with members who only post job or conference advertisements or are

looking for jobs were discarded (mostly because there were no follow ups to those kinds of posts). In extract 4.1 an anonymous online participant of the JoS[2] forum using the name "Bugs Bunny" talks about the kind of topics he thinks attract most of the online interaction. For him it is the human and social related aspects of software development which keep the JoS forum alive.

- The members of the communities must be practitioners. As stated before, the approach of this thesis is from the point of view of the software developers (see section 4.1.1 of this chapter).

Extract 4.1

Thread title: Moding this board must be 'a lot of work'	**Date: 16th July 2004**	**Source: JoS**

"The things that motivate people to speak don't really seem all that software related. Instead they're related more to the 'human' side of things:

Why does HR treat me this way?
What about that Google billboard ad?
How do the old/young people differ?
How did my interview go?
Should I work at this place?
Why do software engineers have such a mentality?
What makes a good hire?
Why is my manager treating me like this?

...and so on and so forth. Most of the technical questions get short answers quickly but the questions dealing with human feelings and human relationships tend to get many times more responses that the pure technical questions. Even technical questions with a 'human' spin get more responses than pure technical questions."

Bugs Bunny

The reason for using the above filters was that I needed a "place" where developers were working and discussing software development topics so I could collect data from there. There are many advantages of using online communities to gather information. The online nature of these communities makes them accessible from any part of the world. I did not need to physically move to meet other people. The public nature of the communities chosen facilitated my introduction to other people as another peer. As there are hundreds of online communities around the world I

[2] JoS: Joel on Software one of the online forums where I carried out the fieldwork. The text segment provided corresponds to data category [Virtual Communities/Motivations] which is not part of the analysis results of this thesis but helped me to make sense of the nature of virtual communities. This is explained further in section 5.2.1 of chapter 5.

was able to select discussions from a wide variety of topics. I also met people with different professional backgrounds, i.e., practitioners that work in different companies and that use different tools and development methodologies. Finally, no special software was required as most of these communities can be accessed through a browser. Thus the access was relatively easy to gain.

The complete list of collected online sources (i.e. elements of virtual communities such as forums and weblogs), 13 communities in total, used in this study is shown in appendix 6. This list includes the online interviews I carried out in MSN messenger. From these 13 online sources, I gathered 107 online conversations whose length varied from only two contributions (1 question and 1 answer) to 60 contributions (see section 5.2.1 of chapter 5 for a discussion about the volume of data collected). The data collected is only a small fraction of the several hundreds of online conversations I read during my fieldwork.

4.2.2 Pragmatic and Linguistic Issues in Virtual Communities: Computer Mediated Communication

Online communication is different from oral and written communication. The online media is also known as "computer mediated communication" (CMC) and possesses linguistic and pragmatic peculiarities which are addressed in this section. A linguistic and pragmatic analysis of the online medium is relevant at this point to emphasise its potential as a medium in which communication is possible and where people can convey and infer meaning. Also it is important to stress the advantages and limitations of CMC for this study so as to justify the design of the data analysis strategy which will be presented later in section 4.4 of this chapter.

Linguistics studies the structure and systems of language. Pragmatics is the branch of linguistics which is concerned with "the study of meaning as communicated by the speaker and interpreted by the listener" (Yule,

1996, 3). Pragmatics studies the relationship between linguistic forms and the users of those forms. This includes people's intentions, people's assumptions, the kind of speech acts that people perform and the knowledge that sender and receiver of messages have about the physical and social world. A speech act is an action performed via an utterance. (For example the sentence "I fire you" represents the action of firing someone. Speech act is a concept introduced by J.L. Austin (1962) in his book "How to do things with words".) Both linguistics and pragmatics are relevant to this study as they address the different ways in which people who communicate manage to understand each other.

CMC can be defined from different perspectives. From a technical point of view, CMC is "communication patterns mediated through the computer" (Metz, 1994, 32). From an organizational point of view CMC is a tool that is used for "distributing information", "increasing organizational efficiency", "creating electronic democracy" and "challenging traditional hierarchies" (Baym, 1995, 2). However, as I am using CMC as a medium through which I want to interpret other people's interpretations, a more relevant, linguistic and pragmatic definition of CMC is "an hybrid form of interaction" or "written conversation" (Marcoccia, 2004, 2). According to this, CMC could be understood as a new kind of register (a variation of language used in a specific kind of situation or in other words "a description of the linguistic forms which generally occur in a particular situation" (Thomas, 1995, 154)), which possesses characteristics of both written and spoken speech. These characteristics appear in different degrees depending on the online setting that is being used. In order to explore better these characteristics I have divided these online settings as three CMC-using situations. The term CMC-using situation has been adapted from Crystal's (2001) Internet-using situations. Crystal has identified five internet-using situations: e-mail, synchronous and asynchronous chatgroups, the World Wide Web (WWW) and virtual worlds (imaginary worlds created by the online participants), which "are sufficiently different to mean that the language they contain is likely to be significantly distinctive" (Crystal, 2001, 10). In this study the term CMC is

used instead of Internet to not limit the scope of the analysis to the internet but to any other network through which online communication is done. Additionally, I have discarded virtual worlds from this analysis because their imaginary nature make them not appropriate settings for real IS and software work. Finally, I have combined synchronous and asynchronous chatgroups as one category, but I address their differences, when relevant. Following, I will explore the linguistic and pragmatic characteristics, first of CMC and then for each CMC-using situation.

Linguistic Characteristics of CMC

From the linguistic point of view, CMC is seen as a having some characteristics of oral speech and writing. E-mail is closer to writing and chatgroups are closer to spoken conversation (McDaniel et al., 1996, 39). Oral speech is "time-bound" and "spontaneous" (Crystal, 2001, 26), CMC allows more time to the speaker to think what he is saying, hence it is less spontaneous (Folkman Curasi, 2001, 6). Oral speech is prosodically rich (see glossary for a definition of prosody), CMC lacks visual, audio and social cues. Additionally, there are also some differences with written speech. Writing is "contrived" (Crystal, 2001, 26), CMC is less formal. Writing is more structured and linear; CMC allows more dimensions (e.g. many conversations at the same time, links and bookmarks) so texts do not have to be read in sequence but in the order the reader prefers.

CMC's lack of visual and social cues, often present in face-to-face conversations, could limit the interpretation of texts to the written statements (Sweet, 2001, 40). "Cues signal the nature of the context", they give the participants an idea of who and where are the others, their characteristics and the relationships between them (Jacobson, 1996, 463). In CMC indications about the participants' physical appearance, age, gender, position and the physical appearance of the setting are lost (Sproull and Kiesler, 1986, 1497). Furthermore, visual signals like nods

from an addressee to indicate understanding, agreement or disagreement are missing. There are three perspectives that address the consequences of this. The first one states that without visual and social cues CMC becomes an "impersonal medium" (Parks, 1996, 2) and a "socially-impoverished domain" (Baym, 1995, 26). The second perspective proposes that the lack of cues enhances social dimensions such as "cultural beliefs, values and norms" (Elliott and Scacchi, 2003, 28). According to this perspective, CMC "liberates interpersonal relations from the confines of physical locality" (Parks, 1996, 1). Thus relationships that arise through CMC have more quality as people get to know each other more. The third perspective states that alternatives to visual cues are created and used in CMC. These are conventions that allow people to understand each other and the context in which conversations are carried out.

CMC allows the creation of "conventions of communication" (Jacobson, 1996, 465) or "chat codes" (Greenfield and Subrahmanyam, 2003, 732) which are limited to the written medium. These are rules that people have to follow to make communication possible. Some conventions are well spread through the cyberspace whereas others are kept within the virtual communities that created it. Some conventions mimic existent conventions from oral or written speech, but others are created especially for CMC. For example, to stress an utterance it can be highlighted by using a different colour. To call the attention of a specific person (in a group conversation) his name can be indicated within brackets, as in "Cecilia: [José] what is your age?" People can raise their voices by using CAPS. Feelings can be expressed by emoticons (e.g.) or other signals (e.g. <<Cecilia is happy>>). Common expressions can be abbreviated (e.g. YMMV, which means Your Mileage May Vary (or your experience could be different)). Newcomers in virtual communities need to acquire pragmatic competence i.e., learn these codes and learn when and how to use them, to become members and to be accepted by the others. For example, I created a dictionary of online codes which helped

me to familiarise with the conventions used in some of the forums. The dictionary is shown in appendix 4.

CMC allows different kinds of speech. First, it is said that CMC fosters direct speech (Parrish, 2002, 2). The anonymity of the medium causes the participants to speak baldy without redressive action (i.e., they do not pay attention to other people's feelings). A negative consequence would be violent behaviour like verbal aggression or "flaming" (Parks, 1996, 2). On the other hand, a more positive consequence would be the ease of communication by reducing ambiguities and misunderstandings. However, there will always exist a potential for misinterpretation, misinformation and misunderstanding in the written world (Willis and Coakes, 2000, 122). Second, CMC also allows indirect speech. People can "communicate meaning via implicatures" (Yule, 1996, 40). Implicatures contain additional conveyed meaning. For example, the utterance "C++ is difficult" apart from its semantic meaning can also imply that the speaker is asking for help. Hearers have to infer the meaning from what is said and from the visual context of the conversation. As CMC lacks visual cues the strategies for interpretation have to be different from the ones used in writing and oral speech. It is possible therefore, that different interpretations can arise from one single utterance (which is not that bad as it could encourage richer conversations searching for clarification.)

Conversations in CMC follow different ordering rules than in normal conversations. In face-to-face conversation, people negotiate their turns with cues showing the others when they can have control of the floor (i.e., the right to speak). Turns are also organised by adjacency pairs which are logical sequences that people expect to be followed (e.g. a question followed by an answer). In some instances of CMC many people can be typing at the same time and without seeing the new messages that are arriving. Their messages are posted in the order in which the server receives the messages (this depends on speed and broadband (Smith et al., 2000, 98)). Additionally, there could be several parallel conversations

taking place at the same time (Parrish, 2002, 10). The result is several threads of conversations overlapping each other. The overlap causes a turn that should have followed another one to appear several lines after. In between there can be turns from other people participating in the same conversation or in others. It is difficult then for the inexperienced, to deduce to which conversation a particular turn belongs and what is its place (order) in that conversation. Although this phenomenon appears more often in synchronous conversations it is also true for asynchronous conversations. For example, replies of e-mails can carry copies of the previous turns which are not necessarily in order.

The lack of visual cues, ambiguity in texts, conventions of communication and a messy conversation structure carried some implications for this study. The nature and structure of online conversations was considered at the time of analysing the texts for the study. For example, the interpretation of turns in long threads of conversation was done methodically to identify the relevant sub-threads to which the turn was related. The implications of the nature and structure of online conversations to this thesis' methodology is discussed in section 4.4.1 of this chapter where I justify my selection of conversation analysis as part of the data analysis approach.

Pragmatic characteristics of CMC

The CMC characteristics that stand out from other types of communication are the reduction of time and financial constraints and the loss of geographical boundaries (Folkman Curasi (2001, 367); Parrish (2002, 3); Sweet (2001, 2)). CMC allows people to communicate regardless of the time zone in which they reside and in the case of asynchronous settings at the time most convenient for them. CMC is less expensive than for example, flying to another part of the world to meet with people. There are no geographical boundaries as one can meet with people in any part of the world.

As it is yet a new channel of communication, CMC has been compared with other more conventional channels, such as telephone calls and face-to-face conversations. Some studies have addressed CMC from the point of view of these other channels of communication assuming that the differences are in fact deficiencies of CMC. There are other studies, however, that see CMC as a different phenomenon which deserves a different framework of analysis. Taking into account both perspectives we can identify some advantages and disadvantages in using CMC.

Participation in some types of CMC is anonymous (Parrish, 2002, 2). In most types people can use false names or nicknames to participate in virtual communities. Consequently, people in CMC are likely to be more self-disclosed than in face-to-face conversation as "there is very little chance of anyone ever linking them with their statements" (Parrish, 2002, 2). Self-disclosure is also enhanced in other types of CMC which are not always anonymous. In e-mails, for example, people tend to focus on themselves in message salutations and closings (Sproull and Kiesler, 1986, 1509). This is an advantage for this study as it was possible, through CMC, to "stimulate [...] disclosure from informants" (Folkman Curasi, 2001, 368) although the lack of face-to-face contact made it difficult to establish rapport. CMC mitigates influencing factors in communication such as status differences, gender, ethnicity and others (Sweet, 2001, 6). Hence, the field is even for all the participants. In organizational environments, for example, everyone regardless of their position sees e-mails and bulletin boards. Messages are sent by subordinates and superiors equally. Anyone within the organisation has access to information "that formerly would have been difficult if not impossible for them to get" (Sproull and Kiesler, 1986, 1510). Through CMC, gender, ethnicity, accents and physical appearance do not play a role (or play a different one that is mitigated), this eliminates predisposition against the others. The focus is then only on the content of the messages.

Although CMC provides an even and democratic field for members of online communities, not all of them participate in the same way. Members can be classified according to their role and level of participation. Based on Millen and Dray (2000) and McDaniel et al (1996) typologies of participants I am proposing the following Types of Participants:

1. Frequent contributors, they usually dominate (McDaniel et al., 1996, 46) the conversations by initiating more conversations and by sending more replies than the others.
2. Sporadic contributors, participate when "a topic of personal interest is discussed" (Millen and Dray, 2000, 170).
3. Marginal contributors, they have low levels of participation.
4. Lurkers, they read without responding (Parks, 1996, 15).

Additionally, members play different roles in the communication process. According to Marcoccia (2004) participants can be senders and/or readers (Marcoccia, 2004, 17). A type of sender, the host, possesses more attributions than the rest. They act as the moderators of conversations and have power over the other members (e.g., they can ban members). The consequences of these differences in online behaviour are that very few members, the more influential ones, dominate topics of conversations. Similarly, it could be said that an opinion or perspective about a topic cannot be associated to all the virtual community but only to the few members that participate in conversations. Unfortunately, due to their lack of participation, lurker's perspective and behaviour cannot be observed. However, their role could be thought as supportive of the community as no one forces them to remain listening in the background.

The ability of participants to read and reread texts before sending them influences the quality and nature of the messages. For example e-mails contain better grammatical use than oral speech (Folkman Curasi, 2001, 7). Warschauer (1997) confirms this when stating that CMC "promotes

more complex sentence structures" (Biesenbach-Lucas and Weasenforth, 2002, 149). CMC also allows more reflectivity (McDaniel et al., 1996, 40) as people have more time to think about their messages.

More characteristics of linguistic and pragmatic characteristics of CMC can be identified by focusing on the individual characteristics of the CMC-using situations:

a. E-mail

E-mail is "a computer system for exchange of messages and other information" (Kettinger and Grover, 1997, 514). E-mails are fast and asynchronous (Sproull and Kiesler, 1986, 1493). Although a message can arrive in seconds to its destination, senders and receivers do not have to be present at the same time. E-mails can be reviewed whenever is convenient for people. As mentioned before, e-mails share some characteristics with written and oral speech. McDaniel et al (1996) believes that e-mails are more like writing (McDaniel et al., 1996, 39), Willis and Coakes (2000) state that e-mails resemble speech patterns (Willis and Coakes, 2000, 121), whereas Biesenbach-Lucas and Weasenforth (2001) state that this depends on the situation (Biesenbach-Lucas and Weasenforth, 2001, 11). For example, a quick note to a relative would look more like oral speech and a notification to a superior would look more like writing.

Despite the lack of visual cues, in e-mails textual identity cues can be included in the form of message headers and signatures, and other message features (Baym, 1995, 2) thus breaking the anonymity of the medium. Other contextual cues might not be found though. Biesenbach-Lucas and Weasenforth (2001) found that unlike word-processed documents, e-mailed documents lacked background or contextual information as senders assume that this information is "being shared with the audience" (Biesenbach-Lucas and Weasenforth, 2001, 16).

E-mails usually contain multiple messages addressing multiple topics. The reason is that because a reply might not come in the near future (e-mails are asynchronous), senders feel the need to anticipate some movements and save some time. Unlike this, in face-to-face conversations, topics are addressed one by one. Additionally, improvement in technology facilitates this as mailboxes and individual e-mails can contain big amounts of information. Consequences for the study could be that sometimes an e-mail will contain turns belonging to more than one conversation and that e-mail files could not be catalogued only under one topic but many. However, as access to e-mail becomes more pervasive and continuous, one could assume that e-mail will move closer to one topic exchanges as it gets closer in style to chatgroups. During the fieldwork I exchanged a few e-mails with some of the online participants. As the e-mails were about topics already discussed in the online forums they remained short and focused.

b. Chatgroups

Chatgroups are "continuous discussions on a particular topic, organised in rooms" (Crystal, 2001, 11). Popular forms of chatgroups are online forums, mailing lists, weblogs and instant messaging. Chatgroups allow multiple users to engage in synchronous or asynchronous conversations. In synchronous situations, users interact in real-time in conversations that have a short life. In asynchronous situations, there are longer gaps between messages and conversations tend to be considerable longer. For both types of chatgroups, logs of conversations can be recorded and reviewed as well as resumed later. This characteristic of CMC is called "persistent conversation" (Erickson (1999) quoted in Marcoccia (2004, 3)).

In addition to the characteristics mentioned before about the different type of organization in online conversations (differences in turn-taking, floor control and adjacency pairs are more evident in chatgroups), and lack of visual and contextual cues, there are other features that need to be

considered. The first one is that having multiple threads of conversations and members participating in more than one conversation results in lack of "collective focusing" (Marcoccia, 2004, 4). That is, in chatgroups, not all the members are discussing the same topic nor are aware of all the topics that are being discussed by others. Instead, sub-groups with "varied focuses" are created. Second, not all the conversations are long in length. A great percentage of initiated conversations are truncated, i.e., are not followed by responses, or are short as the topic of discussion is not of interest for participants. Members of virtual communities would "not feel obliged to answer a message which remains unanswered for a long time" (Marcoccia, 2004, 7). Hence the levels of interests of online members in particular topics are displayed very explicitly.

Implications for the study were, first, that in order to understand some conversations, a reading and segmenting strategy had to be developed. Relevant threads of conversations were picked from the whole body of data. Most truncated exchanges had to be discarded because they did not provide valuable information to the study. From those relevant threads, turns had to be linked to their adjacency pairs so as to give coherence and context to an idea or topic. Individual turns had to be analysed thoroughly to find contributions to more than one conversation and hence more than one topic.

c. The World Wide Web

The World Wide Web or WWW is "the full collection of all the computers linked to the internet which hold documents that are mutually accessible through the use of a standard protocol (the hypertext transfer protocol, or http)" (Crystal, 2001, 13). The WWW is also a device for delivering chatgroups which have been already mentioned in section (b). However, under the classification of WWW I am only considering information or documents in which no interaction between users happens. From the three CMC-using situations, the WWW is the medium which shows more characteristics of writing than of speaking. The WWW is also different

from the other two situations in the sense that communication interaction is unidirectional. The author of the Web pages does not know his audience who can be indeed massive.

With regard to this thesis, two issues are worth mentioning. The first one is the nature of the texts in web pages. As opposite to normal, written texts which follow a linear structure or "unidimensional flow of speech" (Crystal, 2001, 196), texts on the WWW are non-linear and can be read in a "multidimensional way". Hyperlinks are used to jump to other pages. Web pages can be divided into sections; texts can be grouped in boxes or tables. The order in which a user reads a web page depends on his need and on the ability of the designer to attract the attention to some parts of the page. The second issue is the validity of the information posted on the web. As there are no moderators, there is no control over what is published. In searching for a keyword, one can find enormous loads of outdated information. Along with this there is also a large amount of information that might be plagiarised from other sources (Crystal, 2001, 206). It is important therefore to exercise caution when accessing online sources. In this study, part of this work was done by the subjects of study revealing the self-disclosing nature of the CMC media. The web sites analysed were the ones proposed by participants as they had read or written them and were able to recommend them.

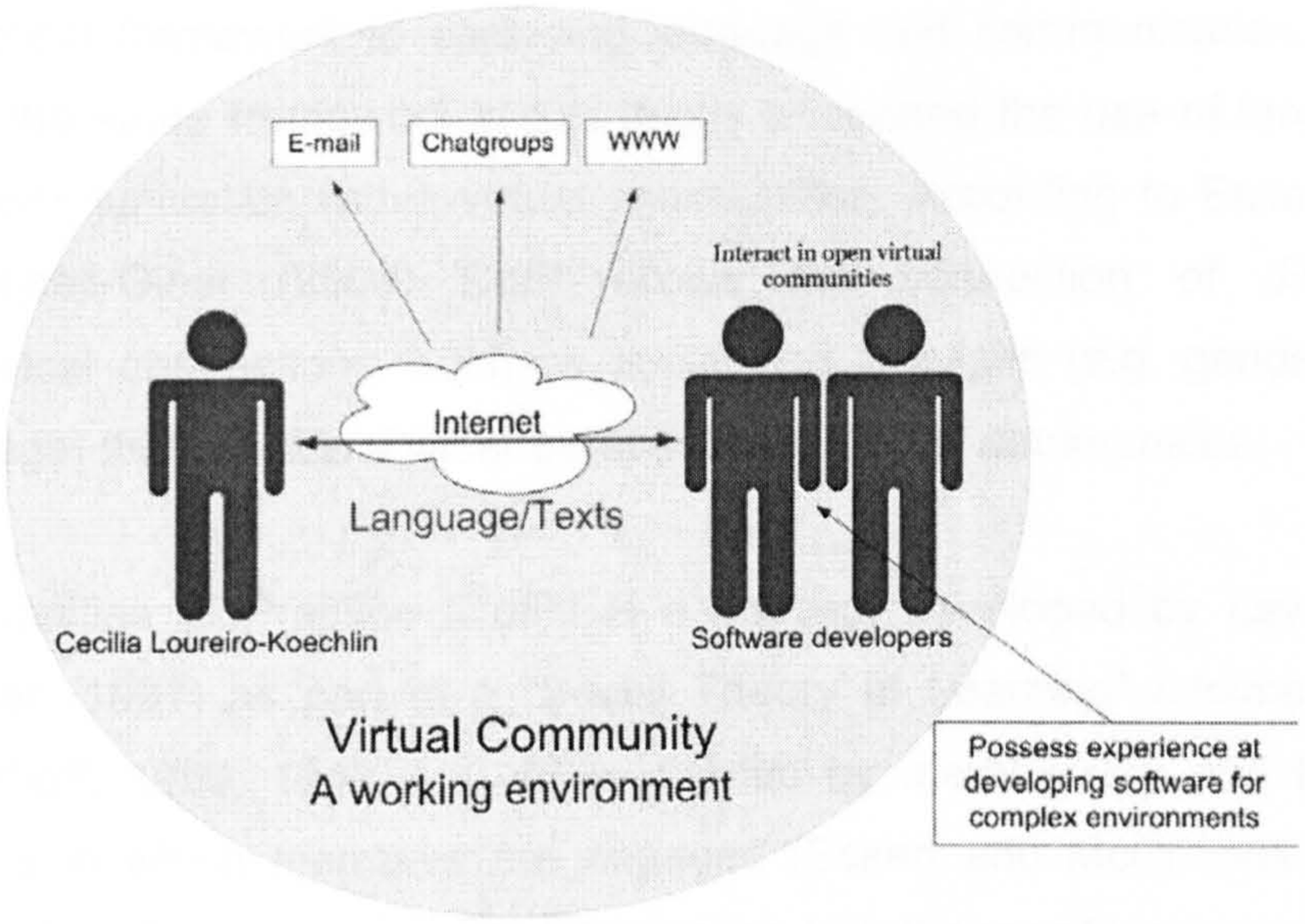


Figure 4.4 CMC as a medium through which to carry out interpretive research

Figure 4.4 summarises the exposition of CMC as the medium through which I carried out the fieldwork. After presenting the characteristics of CMC it is evident that this is a very different medium from more conventional ones like written and oral speech. However, there is evidence that more people are participating in online communities to communicate or look for information (Parks, 1996, 2). It becomes unclear though, how people learn to communicate and perform activities online. As a result some questions arise: how do online participants understand each other? How do online conversations convey meaning? How are goals achieved and how are relationships built in online environments?

4.2.3 Virtual Communities as Communities of Practice

The answers to the above questions are not restricted to linguistic terms. There are other factors playing a role in the construction of linguistic forms within online environments, as for example, the personal interests of each individual, personal backgrounds, IT literacy, age, etc. Eckert and McConnell-Ginet (2000) in their paper on language, gender and power propose the use the concept of Communities of Practice (CofP) as a

theoretical framework to analysing language and communication. I am using the same framework in this thesis to expand the use of language and communication within virtual communities. According to Eckert and McConnell-Ginet (2000) CofP allows the connection of different theoretical abstractions, such as social and linguistic (e.g. gender and language) that provide a wider perspective of virtual communities.

Communities of Practice (CofP) is a concept developed by Lave and Wegner (1991) as part of a "Social Theory of Learning" (Holmes and Meyerhoff, 1999, 174). A CofP is defined by membership and by the practice in which members are engaged (Eckert and McConnell-Ginet, 2000, 490). The purpose of its introduction to a theory of learning was to explore the concept of learning as a social process. Members of a CofP learn while they perform community activities and while they negotiate the way those activities are to be performed. As language is also a "social practice" (Bucholtz, 1999, 210) we can presume that online members also acquire "sociolinguistic competence" (Holmes and Meyerhoff, 1999, 174) as they participate in online activities. Ahmad and Al-Sayed (2006) state that special language in CofPs is consciously created to foster a sense of common purpose amongst a group of people and some times use to exclude" (Ahmad and Al-Sayed, 2006, 77) non members from the CofPs. CofP connects language use with other social practices. In the case of online communities these social practices could be engaging in software development, helping or asking for help, solving conflicts, etc.

Conceptualising virtual communities as CofP allows us to understand how online members perform activities through the use of language. The following characteristics of CofP give us a light in this regard:

a. Diversity and conflict among CofP members: there are core members who are totally engaged with community's practices and peripheral members who are in the process of becoming full members

b. Ambiguity as a condition of negotiability: members do not always interpret meanings in the same way. Social practices become "dynamic, always open-ended and generative of new meaning"
c. Mutual engagement: quantity and quality of interaction, it is the "basis for the relationships that make the CofP possible"
d. Joint Enterprise: a process of negotiating practices and goals
e. Shared Repertoire: of resources for negotiating meaning, this includes linguistic resources

Source: Adapted from Holmes and Meyerhoff (1999, 174-176) and Wenger (2003, 84)

As CofP acknowledge differences within communities it is a useful tool to explain how the linguistic styles of the online members blend to create a style for a whole virtual community. Mutual engagement allows members to learn the linguistic repertoire and to participate in the negotiation of new ways of expressing themselves. CofPs ambiguity and diversity is dealt with by negotiation. The concept of joint negotiation helps to understand how such new ways of chatting with multiple, parallel, overlapping and slow conversations and a lack of visual cues, is not wrong but different, as it was negotiated by online members. Despite the lack of visual cues, senders can convey meaning in a message that can be inferred by the receivers. Personal and working relationships can be built up through virtual communities. In fact, work can be done through this medium. However, due to the different nature of CMC, especial strategies have to be designed which might be different from a strategy prepared for face-to-face interviews for example. Having said this, in the next section I will explain how this study was carried out by using CMC – Virtual Communities as the source of data.

4.3 The Online Ethnographic Method

Online ethnography is the selected method for this research. It is grounded in the interpretive paradigm and "demands first hand involvement in the social world chosen for study" (Marshall and Rossman, 1989, 106), i.e., virtual communities of practice. An ethnographer has to live or work with the people he is studying. These experiences give him a perspective of the phenomenon that is closer to the one that the subjects of study have. The researcher then, is able to interpret the phenomenon the way the others do (Gill and Johnson, 1991, 97). The result is in-depth understanding of the phenomenon or verstehen (Gill and Johnson, 1991, 133). This immersion into the field, however, does not imply that the researcher becomes a full member of that community. On the contrary, as the researcher is doing fieldwork he has an additional role and different intentions which distinguishes him from the rest. This is the "essence of the fieldwork" (Wolcott, 1995, 66).

"Ethnography has recently been adopted by the information systems field as a valid means of interpretive inquiry and research methodology" (Walsham (1995) quoted in Coakes and Coakes (2000, 104)). Online ethnography is not a common practice in IS research yet but it is growing in popularity. There are many researches that use online tools (CMC) to communicate with subjects of study; however, not all of them can be catalogued as ethnography. Ethnographies require full immersion into the field. For example, a study by Chou (2001) about internet addiction that uses chat rooms to carry out interviews is not an online ethnography as the researchers are not involved with the subjects in their activities on the internet. Also Madge and O'Connor (2004) do not claim having done an online ethnography although they used web based questionnaires and online synchronous interviews to research new expectant parents. Research that uses online tools can also study the characteristics of virtual communities and not be online ethnographies. For example Greenfield and Subrahmanyam (2003) study online discourse

(conversation) by obtaining "transcripts" from an online "chatroom". Researchers doing real ethnographies claim deep insight into subject's experiences; for instance Browne (2003) who studies online learning using "cyber-ethnography". Online ethnographies can be classified according to their purpose. Some online ethnographies study virtual communities. These researches focus mostly in the sociological and linguistic characteristics of these communities. For example Baym (1995) explores the development of solidarity and identity of online groups through the exploration of humour. Figure 4.5 summarises the classification of online research and locates this research within this framework. My thesis focuses on software development but is also concerned with the characteristics of virtual communities, particularly with their linguistic and pragmatic aspects, as they are relevant to this thesis' methodology.

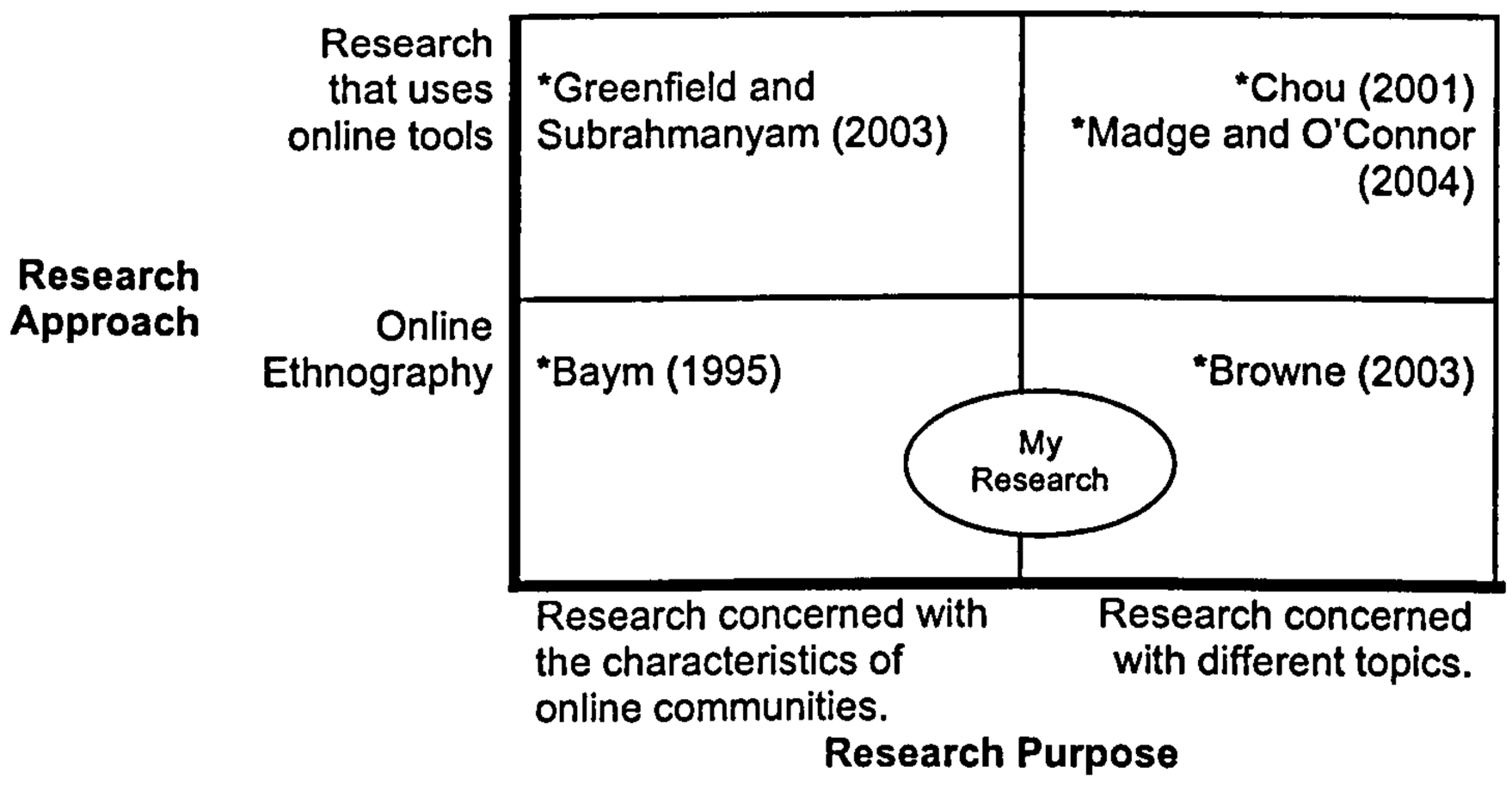


Figure 4.5 Classification of Online Research

Participant observation is the "central ethnographic data collection technique" (Punch, 1998, 188) that allows observation and participation in people's daily activities. The kind of participant observation chosen determines the level of immersion of the researcher in the field. There are four levels of involvement as shown in figure 4.6. The researcher can choose to participate in the activities or only observe as a simple spectator. He can also decide whether it is convenient to let the others

know what his real intentions are. From this classification I find that my role in this study was as "participant as observer" in virtual communities. I participated as a member of virtual communities but my aims were not entirely work related or directly problem solving as theirs. My objectives were to look for information about software development for my research. The level of intrusiveness was low as the purpose of these communities is to discuss and share opinions and software development problems and because most of them are open to new participants to collaborate with them.

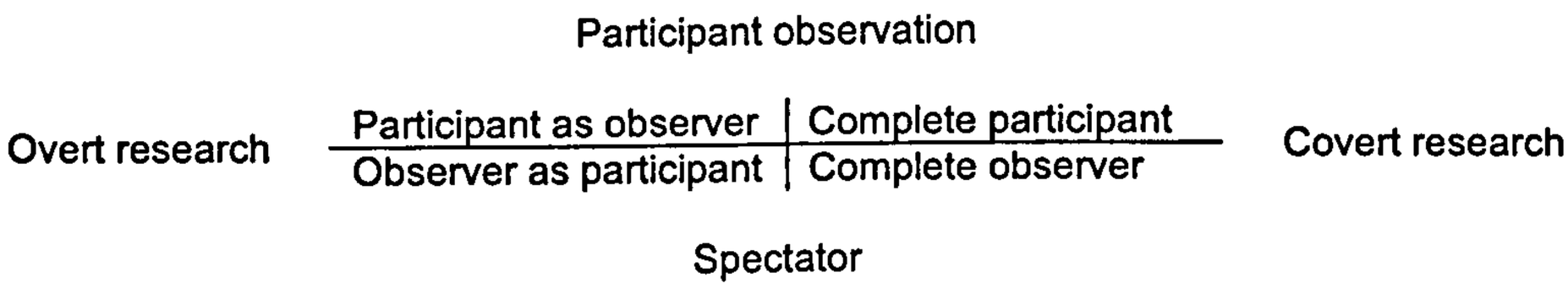


Figure 4.6 Levels of involvement in participant observation
Source: Gill and Johnson (1991, 117)

The role of complete participant in participant observation can be regarded as a reflective practitioner and not as a researcher. In fact, I was part of that segment when I was working as a systems analyst and moved across to participant as observer when I entered the PhD.

4.3.1 Data collection techniques

Data was collected through participant observation in online communities. Environments like discussion forums, chat rooms, weblogs and e-mails were the main settings for the fieldwork. During online participant observation, there were two main activities that I performed: participating in conversations and reviewing information. These activities correspond to two ethnographic techniques: ethnographic interviews and archival research.

a. Ethnographic Interviews

Ethnographic interviews are open-ended, in-depth interviews (Punch, 1998, 178). They are a combination of unstructured interviews and standard conversations. On the one hand they are like unstructured interviews because the researcher may have to explicitly request to hold the interview (Flick, 1998, 93), follow an outline of questions to ask and guide the respondents to a certain area of interest. On the other hand they are like conversations because of their friendly nature (the researcher has been accepted in the community and is known by its members) and because they can occur at any time during participant observation, for example, when a good topic is proposed by someone. There are three kinds of questions that can be asked in these kinds of interviews (Flick, 1998, 93):

- Descriptive questions: informants can describe the tools they use to work or describe the activities they perform. (See an example in extract 6.23 in section 6.3 of chapter 6.)
- Structural questions: "answering them should show how informants organise their knowledge about the issue" (Flick, 1998, 93). (See an example in extract 6.8 of section 6.1 of chapter 6.)
- Contrast questions: "provide information about the meaning dimensions used by informants to differentiate objects and events in their world" (Flick, 1998, 93). (See an example in extract 5.4 of section 5.12 of chapter 5.)

Ethnographic interviews through conversations by e-mails and chatgroups, were important sources of information of this study. Other conversations in which I participated, but which I did not initiated are also considered under this category. Although the term interview implies that the researcher has the control of the interaction, the online ethnographic nature of my study allows the researcher to participate in other people's conversations and obtain valuable information from them.

b. Archival research

Another activity that was performed within the framework of online participant observation was searching information in the WWW. For the purpose of this study the sources of information were classified in two: logs of past conversations in a specific community in which I did not participate or web-sites with relevant information. Both forms of information had one thing in common; they were pointed to by a member of the virtual community as an important piece of information that was related to a specific conversation (an aspect of online participants that reveals their self-disclosure). For example, if a particular respondent referred to a previous conversation in the same group or if he referred to a technology or software development methodology published in a particular web-site.

4.3.2 Justification of the Method

In addition to those in the discussion about virtual communities, I can add more reasons for choosing online ethnography as the method for this research. First, in my experience the internet and other networks are familiar environments to information systems and software developers, especially for discussing work related tasks. Online tools, such as e-mail or chat rooms, are sometimes compulsory within working environments. Therefore as I wanted to study software development from the point of view of developers I needed to approach them in an environment that is familiar to them: the internet. Second, doing online ethnography placed me in the "same critical plane" (Harding, 1987, 9) as the information systems developers I wanted to approach. I shared the same epistemological grounds as them (Parrish, 2002, 8). My participation in online conversations was as theirs. I received the same information simultaneously and exactly as other members did I was able to review previous conversations and post new messages. A drawback of online

ethnography though emerges from its written nature (as explained in the Linguistic and pragmatic characteristics of CMC section). Online ethnography does not provide enough tools to judge the validity or veracity of the online participants' statements and to obtain background information to capture the "social world behind the words of the actors" (Klein and Myers, 1999, 78). For example, in online ethnography it is not possible to verify whether the depiction of developer's users is accurate or if his description of a work problem is not biased due to problems with his colleagues. However, in the case of the present thesis, what is important is the perception and point of view of the participants, so the drawback becomes a secondary issue.

4.3.3 Interpreting phenomena through online media

Doing online ethnography with an interpretive perspective involves a process of interpreting and understanding social phenomena through the texts that are generated by human interactions. Texts in this sense mean language, the media through which In philosophy phenomena and texts are studied by phenomenology and hermeneutics respectively. Both are the most important foundations of the interpretive paradigm (Klein and Myers, 1999, 70; Myers, 2003). "Phenomenology is the intuition of essences" (Boland, 1985, 193). "An essence is defined as that which is necessary for something to be recognized as that thing" (Hirschheim, 1985, 25). We grasp the essences of things and experiences subjectively. As these essences are not physical entities, they can only be perceived by intuitions. Through scrupulous inspections of their own conscious (A dictionary of philosophy, 1984, 266) researchers are able to uncover the essence of the objects of study from their assumptions and prejudices. It is in this process in which meaning is provided to the experience (Hirschheim, 1985, 25). "Phenomena bear their meaning with themselves" (Gadamer and Linge, 1976, 131). Hence phenomenology is concerned with the search for meanings of experiences. We find these meanings by entering into a dialogue among the researchers and the

subjects (Boland, 1985, 195, 200). This is an hermeneutic problem, "the problem of translation and interpretation of texts" (Boland, 1985, 194) or "text-analogues" (Myers, 2003) in which language plays an important role. ("An example of a text-analogue is an organization which the researcher comes to understand through oral or written text". (Myers, 2003).) In fact qualitative, interpretive research "is generally language intense" (Campbell, 1997, 125). Gadamer, an advocate of hermeneutics, sees language as the "transmitter between actual experiences, traditions, etc. and the process of understanding" (Hirschheim, 1985, 131). Understanding is a "dialectic process" (Boland, 1985, 195) "between the understanding of the text as a whole and the interpretation of its parts" (Myers, 2003). This is a circular relationship called the hermeneutic circle.

These concepts are put together in figure 4.7 to explain the process of interpretation of Software Development for Complex Environments through the perspective of software developers. Once asked subjects in a virtual community consciously explore their subjective experiences and communicated them to the online community of practice. Additionally, I was able to grasp more ideas by watching the interactions of others. Language was the media through which experiences were communicated and were transmitted as texts by electronic tools. Although there was no face-to-face contact, other kind of expressions like feelings or facial expressions were also transmitted as emoticons or text-analogues. Being online texts the only medium through which I collected data, hermeneutics played an important role. During the online ethnography, I grasped the subjects' messages in a subjective experience. Then I inspected my conscious and uncovered the essence of my experience so as to provide my own meaning to it. This understanding was gained by interpreting the texts sent by the subject while we, the subjects and I, were immersed in a virtual community.

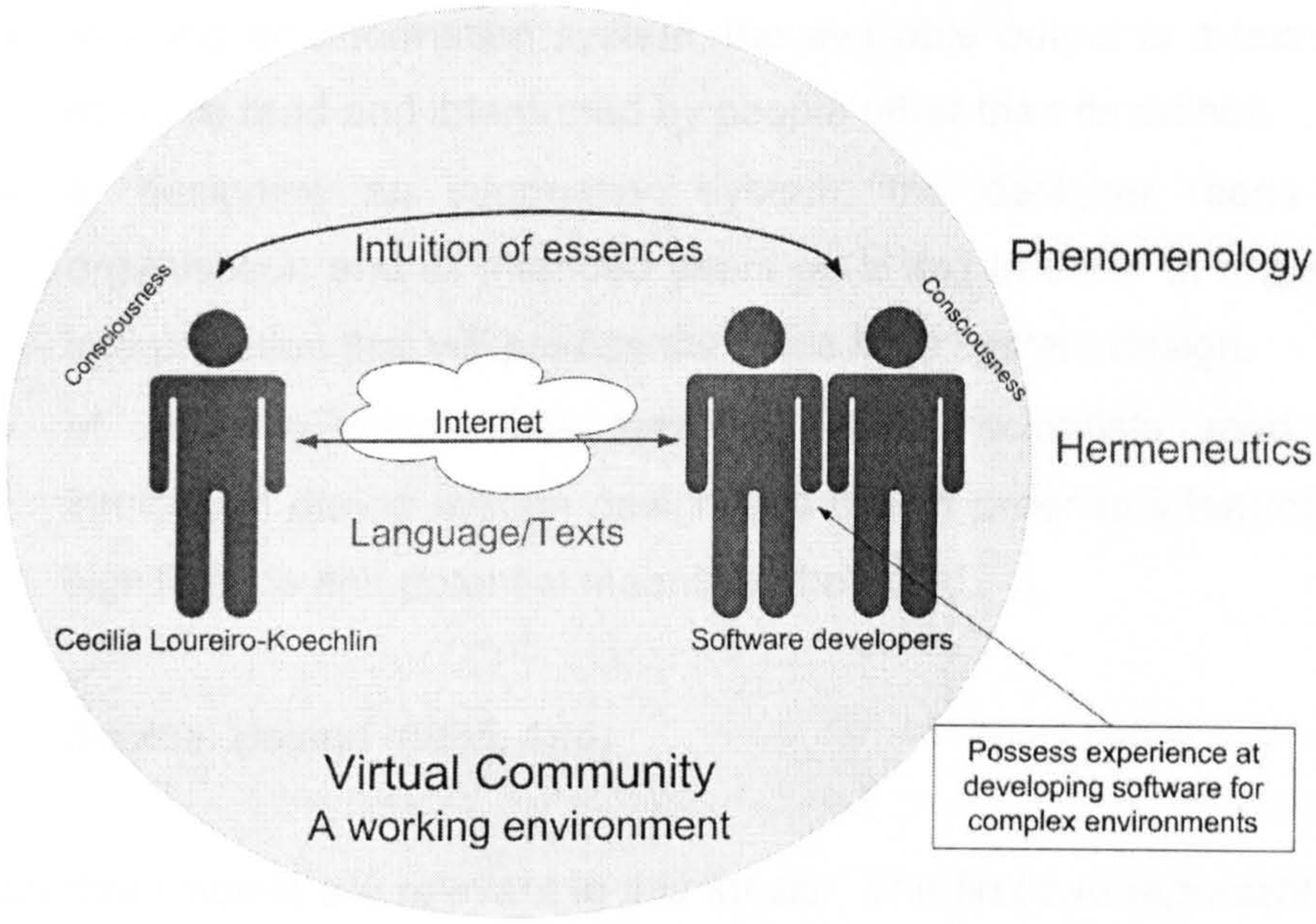


Figure 4.7 Interpreting Software Development with Online Ethnography

Boland (1985) defends the use of phenomenology and hermeneutics in information systems research. For Boland the understanding of the hermeneutic problem enables the researcher to see the importance of phenomenology in information systems. He argues that "the design and use of information systems is the text that we must try to understand. It is through a hermeneutic process that we can approach this task and develop an interpretive description of information systems" (Boland, 1985, 194). According to Boland (1985) there are three hermeneutic tasks in information systems which justify the use of hermeneutics in IS research. The tasks are:

- In using an information system, the available output is a text that must be read and interpreted by people other than its author.
- In designing an information system, the designer reads the organisation and its intended users as a text in order to make an interpretation that will provide the basis for a system design.
- In studying information systems, social scientists read the interaction during system design and use in order to interpret the significance and potential meanings they hold.

Source: Boland (1985, 196)

These three points are relevant to this thesis. The first two represent the area of application of this thesis and the third one reflects the process of researching. In seeing IS practices and research as a hermeneutic task I was able to give meaning to the texts and text-analogues generated by the accounts of the people studied. Also within the process of understanding I was able to answer the research questions and generate knowledge by consciously interpreting my experiences in the fieldwork.

4.3.4 Ethical Issues in Online Ethnographic research

Researching in virtual communities brings with it some ethical issues that might affect the process of investigation. These issues arise from the relationship between the researcher and the subjects of study (Gill and Johnson, 1991, 126). They have to be considered before, during and after the research is finished because they provide credibility, reliability and validity to the research. Brownlow and O'Dell (2002) discuss some ethical issues which should be addressed in any online research. For them there are three issues that should be considered. The first issue is privacy and confidentiality which raise the question about the "privacy" of messages posted on online forums or any web site. A researcher could be a outsider who interferes in the communication between the sender of the message and the intended receiver. In online forums, the answer

relies very much on who the intended audience is. For example, in open public forums messages are read by anyone who can access the site. However in self-advocatory groups, messages are intended only to the members of the forums. In these cases the researcher should introduce himself and state his purposes at participating in the online group (a critical step as it would facilitate access to the online community). In the case of the WWW in most of the cases the audience is anyone who reads the content.

The second issue concerns the ownership of the data. Is the data from online forums public or private property? Again this depends on who the intended audience is. If the researcher is dealing with private data, he should ask for permission to use and publish quotes from posts in online forums or web sites.

The third issue is informed consent. It is the condition whereby a person gives consent to participate in an investigation based upon an understanding of the research objectives and implications. In the case of interviews, for example, informed consent is given by people who make a conscious decision to participate in them. However, if the researcher is doing online ethnography, online participants would not have the same control over consent. This is especially true in the cases of covert research. The principle of informed consent states that in order to get access to virtual communities I have to present my project and aims to the other online members in an open and transparent way. Norris (1993) states two conditions to achieve informed consent: the awareness of the subjects of study and their freedom to decide whether to participate or not.

Due to the nature of my study, I needed access to public online information for most of my data collection. My initial perception was that I could use public information available on the internet freely. However I sent a message to one of the main forums where I did my fieldwork to confirm this. In this message I used my real name and I stated my

intentions as PhD researcher within the community. I also asked for permission to quote some of the participations in the thesis. The whole thread resulting from this message is shown in appendix 5. The outcome of this online conversation was very positive. Most of the respondents agreed in that due to the public nature of that forum there was no need to ask for permission to quote any post and that I should carry on with my study. Also where I conducted online interviews via MSN messenger I had to explain the topic of my thesis and the purpose of those interviews. However, in other forums and weblogs where I collected data I did not reveal my identity as a PhD researcher. I participated as any other online member. The reason I had for doing this was to explore the differences between researching covertly and overtly. The results did not differ much as the content and quality of the conversations were concerned. However, there was a difference at a personal level. Where people did not know my real identity I felt more relaxed in participating. Where I revealed my identity I always felt I was being examined and subject to criticisms. As my subjects of study were practitioners, my position as an academic was not a familiar one for them (at least that is how I saw it). I was in a dilemma. On the one hand I needed to conduct the online ethnography as a PhD researcher. On the other hand I felt identified with the online participants who did not have the need to analyse the data and write a thesis but just participate in the online forums for software development work reasons.

4.4 Data Analysis Methods

Analysis is the "detailed examination of anything complex made in order to understand its nature or to determine its essential features" (Webster's, (1964) quoted in Tesch (1990, 113)). In qualitative research, data about a certain phenomena is collected in the form of texts (e.g. written data, audio and video tapes). The objective of qualitative analysis therefore, is a thorough examination of those texts. Depending on the intentions of the researcher, interpretive analysis can be descriptive or theory building.

In descriptive data analysis, the objective is to describe the elements and characteristics of a phenomenon. Theory Building data analysis goes one step beyond description and aims at developing abstract concepts that explain the nature and behaviour of the phenomenon studied, and that can be applied to other similar cases (Tesch, 1990, 114). As the topic of my research is not of concern only to a single organisation, but something that may be applied to a variety of them, I aimed at theory building, specifically a theory based model of social issues in software development. Thus the answer to the research questions is a model grounded on the understanding of software developers about their work with complex processes. This is explained more in section 4.6.

The data analysis strategy developed for this research is aligned with the methodological guidelines mentioned above. As I explained, online ethnography guides the data collection and phenomenology and hermeneutics guide the interpretation of the data. According to these philosophical perspectives, the data collected through online conversations and other sources contain information about the essences of the phenomena studied as seen by software developers. My job as a researcher was to thoroughly read the texts. Following the hermeneutic perspective, I segmented the texts in parts, and each part was considered in relationship to the whole (Tesch, 1990, 94). A better understanding of the whole was gained when every segment was analysed. According to the phenomenological perspective, this is a subjective process. The experience of reading the texts and recalling the conversations repeatedly was equivalent to moving the information about the essence of the phenomena from the minds of the members of the online communities to mine. At the end I had to uncover the essence of the phenomenon studied by a scrupulous inspection of my own consciousness (A dictionary of philosophy, 1984, 266).

Unlike quantitative data analysis, there are no formal or "cookbook" (Hycner, 1999, 143) procedures to follow for analysing qualitative data (Marshall and Rossman, 1989, 200). There are however different

approaches which give some orientation to the researcher. These approaches are differentiated by their perceptions about data, by how they organise data and by their aims (e.g., description or theory). For this study, I decided to combine two approaches to data analysis: conversation and content analysis, which I will describe and justify in the following sections.

4.4.1 Conversation Analysis

Conversation analysis (also known as discourse analysis) is interested in the formal procedures through which conversations are mediated and certain situations are produced (Flick, 1998, 200). Discourse in this context "usually means instances of communication in the medium of language" (Johnstone, 2002, 2). Examples are an essay, a letter, an oral conversation, a transcript of a conversation, and an online conversation. "Conversation analysis holds that talk is an orderly affair" (Parrish, 2002, 6). Utterances in a conversation are indexed according to their order of mention. The interpretation of an utterance depends on the previous statements i.e. its context. According to conversation analysts, the participants create indexes interactively during their participations to create a coherent conversation. This means that their contribution is relevant to the previous ones and because of this, makes sense to the others. To do this participants follow rules such as turn-taking and adjacency pairs.

In online research, conversation analysis is concerned with features of online conversations like turn-taking, adjacency pairs, visual and contextual cues and conversation structuring (multiple conversations, truncated exchanges, etc.), which are different to the ones from face-to-face conversations. Conversation analysis tries to establish whether online conversations convey meaning as face-to-face conversations do. In this research, conversation analysis was used with content analysis for the analysis of online conversations and web sites. Conversation

analysis also gives light to the strategies of big contributors for controlling the topics of conversations. For example, it may be that the others members respect and are identified with what a popular member says; or it may be that the others are not interested at all and therefore do not intervene. Summarising, conversation analysis was used in this research:

- To guide the segmentation of data in content analysis. Conversation analysis helps to identify threads of conversations and turns. This is not an easy task as turns may come in many lines and juxtaposed by the participation of other people in between. The identification of threads depends on the identification of turns with contributions to a single topic.
- To guide the interpretation of data in content analysis as for example the analysis of textual cues that might convey more information than their semantic meaning. The thread of conversations that are identified also might give context to some assertions and might clarify ambiguities due to indirect speech.

4.4.2 Content Analysis

Content analysis is "a procedure for analysing textual material, no matter where this material comes from" (Flick, 1998, 193). It "facilitates the production of constructs from textual data through a systematic method of reduction and analysis" (Priest et al., 2002, 36). Unlike conversation analysis content analysis does not deal with structure but with subject matter. Content analysis can be done in two ways (Tesch, 1990, 79,80). The first way is by carrying out an inventory of words and by arriving at conclusions from the number of times a word is mentioned. The second way is by classifying segments of texts in categories which "should reflect the purposes of the research" (Holsti, 1969, 95). As my aim is to interpret texts and build theory (and not quantify words in a statistical sense), I

chose to develop categories, which are refined into abstract concepts that shape the final theory.

The first stage in content analysis is to define the type of segments of data to be analysed. These segments are called recording units or units of general meaning (Hycner, 1999, 145) and can be words, sentences, paragraphs, etc. which "express a unique and coherent meaning [...] clearly differentiated from that which precedes and follows". Recording units can be classified in coding units and contextual units (Flick, 1998, 193). Coding units are the "minimal part of the text which may fall under a category" and contextual units are the "largest element [...] which may fall under a category". For the purpose of this study these were the units selected:

For online conversations (i.e. chatgroups, e-mails)

- Recording unit: a contribution (or turn)
- Contextual unit: a thread of conversation (a thread of conversation as a discussion about one topic)

For the analysis of other sources of information such as web sites:

- Recording unit: themes: "a single assertion about some subject" (Holsti, 1969, 116).
- Contextual unit: a web-site.

The next stage in content analysis is to classify the segments of data into categories and then assigning codes to categories. Miles and Huberman (1994) suggest three approaches to creating categories:

- Creating a "start list" of codes prior to fieldwork. That list comes from the conceptual framework, list of research questions, hypotheses, problem areas, and key variables that the researcher brings into the study.

- A more inductive... do not precode any datum until data collection, see how it functions or nests in its context, and determined how many varieties of it there are.
- ... between the two approaches... creating a general accounting scheme for codes that is not content-specific but that points to the general domains in which codes will have to be inductively developed.

Source: Miles and Huberman (1994, 57)

The approach for this study is the third one. Based on the research questions and some relevant themes from within the literature review, I delineated certain areas of enquiry which were the basis for starting conversations (see chapter 5 for a discussion about how the data categories were developed). The creation of the real categories though was done inductively from data which were relevant to the topic of the research. The creation of codes was an aid to retrieve categories and the texts belonging to them.

The final stages in content analysis include the grouping of material belonging to the same categories and making sense of them as wholes. The researcher then describes the parts and characteristics of each category separately. Finally, all the material belonging to all the categories is collected. From this material, the researcher writes summaries in the form of memos containing descriptions and explanations of all the categories and the associations among them, as they explain the nature of the phenomenon studied. The final stage of data analysis of the present research is discussed in section 5.2 poof chapter 5.

The overall approach

In summary, the approach to data analysis was content analysis with help from conversation analysis. As the aim of this thesis is theory building the extraction of relevant concepts from the data is needed. In content analysis, these concepts are classified in categories and then associations between them are identified. Conversation analysis is used to address some problems with the form or structure of conversations and to aid interpretation of data. The overall approach is complemented by the use of a computer application NUD*IST which aids the management of data.

4.4.3 The role of NUD*IST in this research

NUD*IST is a computer application that works with non-numerical, unstructured data. It provides data management but does not do data analysis. Unlike quantitative data analysis, which can be automated, qualitative data analysis requires the intervention of a human mind. Qualitative data is full of meaning which needs to be uncovered, inferred and interpreted within a context. This is a job that cannot (not yet) be performed by a machine. NUD*IST is suitable for data analysis techniques such as, content analysis, grounded theory, or narrative analysis which aim to extract themes from data. Texts can come in the form of field notes, transcripts, minutes or pictures. As a data management application, NUD*IST helped me to perform the following tasks:

- Group all the files containing conversations and web-sites in one research file project.
- Create codes representing categories or sub-categories, and link those codes to segment(s) of texts. (The complete tree of data categories is presented in appendix 2.)

- Create annotations and link those to codes or segments in texts. (See an example in extract 5.6 in section 5.2.2 of chapter 5.)
- Search for:
  - Keywords within the documents in the project
  - Texts linked to a particular code
  - Codes linked to a particular segment of text.

There are some risks in using software tools for data analysis. It is said that it "may cause distancing between researchers and data" (Beck, 2003, 232) a risk that has a big impact in qualitative analysis. However, this risk is minimised in NUD*IST as the creation of codes and annotations, and their association to segments of the texts is always done by the researcher. Also this risk is overtaken by the following advantages of using software tools:

- Decreasing the time required to perform tedious manual tasks.
- Enabling researchers to handle large amounts of qualitative data.
- Enhancing the flexibility and comprehensiveness of data handling
- Permitting more rigorous data analysis
- Allowing for a more visible audit trial

Source Beck (2003, 232)

Using a software package does not always save the researcher from some manual work. As I was using NUD*IST v4.0, a rather old version, I needed to adapt the online conversations to a "text" format as it does not accept word or html documents (see glossary for a definition of html). The following are the steps that were necessary to convert online data into a format suitable for NUD*IST v4.0:

- Create Text Files with the text content of web pages.
- Mark a header for the document (title that explains the content for Nudist). The lines that belong to the header have to be marked with asterisks * at the beginning.
- Divide the document in sections. Each section starts with an asterisk at the beginning of the first line. Sections are:
  - Turns in conversations
  - Subsections or paragraphs in readings
- When converting to text files some graphics get lost. To maintain the closeness to data, a description of those graphics should be added. This helps the researcher to not distance himself from his data. Graphics are:
  - Emoticons in some forums
  - Diagrams in readings

4.4.4 The Organising System of Data

Based on the discussion of the approaches for data analysis I designed a data organising system. The organising system of data is a strategy for data management and data analysis. The concept of an organising system is employed by Tesch (1990) to describe in detail plans for data segmentation, data categorisation, data coding and analysis. An organising system has to be guided by the methodological approach to the research and the data analysis technique. My organising system comprises six steps. In step one to five texts are analysed one after the other in an iterative and cumulative process, where categories are extracted, described and added to the developing theory. Step six is the final step and summarises the previous ones.

Step One: The storing system

This step is carried out immediately after data collection. It involves the naming and storage of texts according to their natural properties. Files were named according to the following criteria:

- Date of Collection: (dd-mm-yy)
- Sender or Source: Name of sender or name of chatgroup or name of web site.

And were stored in folders whose names followed this criterion:

- Name of the: Chatgroup/E-mail/Web Site/(Month-Year)

I included the files in the NUD*IST Project created for this research. As NUD*IST also allows me to add comments to files I included brief descriptions of the content of each document. Figures (a) and (b) in Appendix 7 show screenshots of the data files stored in my computer and as part of the NUD*IST project.

Step Two: Segmenting Data

This step is concerned with the identification of recording or coding units which are relevant to the research question and their classification within a system of categories. In NUD*IST this task is performed by selecting the segments of text and assigning codes to them. Figure (c) in Appendix 7 shows how a segment of text was assigned to a code (i.e.: data category). The names for the codes and their definition are tentative at the beginning and are refined later. Chapter 5 explains how the process of category creation evolved.

Step Three: Defining categories

After coding each document, the texts belonging to each category are reread to look for the elements and characteristics of each category. Descriptions and comments on each category are written as annotations and linked to the codes. (See an example in extract 5.6 in section 5.2.2 of chapter 5). If necessary, the categories are divided in sub-categories and the texts re-linked to them. This process is shown in more detail in section 5.1 of chapter 5. NUD*IST creates categories within a tree structure so categories can contain several levels of meanings. Each subcategory therefore was assigned a code within NUD*IST. The complete tree of categories for this study can be seen in Appendix 2.

Step Four: Refining Categories

As the reading of documents advanced, the number of categories increased. It was then worth analysing the themes covered by each category and ensuring that the category was relevant to the research questions. In this process, some categories were found to be redundant. For example, if two different names (codes) for the same concept were assigned, these categories had to be merged and the description had to be adjusted to explain the material covered. Other categories were clustered into other themes. This means that some categories became subcategories of another category. Also some categories were split because they covered many themes. Since this was the heart of the data analysis this step is further explained in section 5.2 of chapter 5.

Step Five: Developing abstract concepts

This step corresponds to the theory building strategy where categories are transformed into abstract concepts. The process started by looking at the elements or dimensions of the themes covered by each category. When these were identified, they were described in more general terms,

i.e. not describing them in relation to the data collected but as a concept on their own. This was a task of interpreting the appropriate descriptions and interactions of the fieldwork respondents (Halfpenny, 1979, 807). Again this is furthered explained in section 5.2 of chapter 5.

Step Six: Shaping of the Model

I stopped adding new data when I got to the point of "theoretical saturation" (Carroll and Swatman, 2000, 240). I then refined categories and the concepts and identified the relationships between them. Then the final step was to put all the material together and give it a more comprehensible shape. This and the previous stage are described more in-depth in section 4.6 of this chapter. As it is the aim of the research the result of the theory building process is presented in chapter 7.

4.5 Research Aim and Research Questions

Having presented the paradigm and method for this thesis, it is pertinent at this point to re-present the research aim and questions as a way to link the approach with the theory building strategy and contribution of this thesis. In the exposition of the research problem in chapter 1, I describe a situation in which software developers have to develop software for adaptable, unpredictable and multi-group settings. These kinds of situations are the ones that arise in decision-making and collaborative environments. The human and social aspects emerging from such environments should be considered along with the technical implications of IS and software development, installing data warehouse databases for example. Similarly the human and social aspects occurring in the development environment should be considered as they affect the perceptions of developers of the complex organisational settings and the development of the software. Thus the research aim is:

> To identify and explain how the social issues within the software development environment and the social aspects of complex organisations shape software and the process of software development.
> (Repeated from section 1.2 of chapter 1)

In the light of this aim I designed the following research questions (see table 4.2, repeated from table 1.1 of chapter 1) each of which addresses human and social aspects from within a different context:

Context	Question
Social Issues in Complex Business Settings	Q1: What are the human and social aspects of business environments that make software development complex and difficult?
Social Issues in Development Environment	Q2: What are the social aspects in the development environment that allow or hinder the identification of social issues in business settings during the development of software?
Social Issues considered in Development Practices	Q3: What aspects of current software development methods address or neglect social issues in business settings? What is social software and does it have a role to play in improving these practices?

Table 4.2 Research Questions

The nature of research aim and research questions presented above determined the overall research paradigm of this thesis, the data collection method and the theory generation strategy which will be presented in the next section.

4.6 Theory Generation Strategy

Carroll and Swatman (2000) state that theory building is a "poorly understood path of moving from broadly ill-defined research themes, collecting masses of data, analysing and interpreting them to build theory" (Carroll and Swatman, 2000, 236). In this thesis theory building is part of the analysis stage and follows the category generation. Concepts and propositions emerge from categories by de-contextualising the data from the original online conversations and thus making them more abstract.

Concepts are "abstract ideas generalized from empirical facts" (Taylor and Bogolan, 1984, 133) and include refined, abstract versions of the descriptions of the categories and of the comments from the fieldwork data under each category. As these concepts are a higher level of abstraction, they may stand for other similar pieces of data (Tesch, 1990, 138). This means that they "may be valuable in the future to explain other organizations and contexts" (Walsham, 1995, 79). Propositions about the abstract concepts are "general statements grounded in the data" (Taylor and Bogolan, 1984, 134). Propositions explain or theorize the relationships between concepts (Tesch, 1990, 85) and are the foundations for theory building.

The set of abstract concepts and propositions comprise a theory. Theory is a "coherent description or explanation of observed or experienced phenomena" (Gioia and Pitre, 1990, 587). The phenomenon is, in this case, the development of software for complex business settings. By adopting an interpretive perspective, that defends that social phenomena can only be understood from the point of view of the people involved in the development activities (Burrell and Morgan, 1979, 5), the software development phenomena have been observed in this thesis through the views of software developers in virtual communities.

The process of creating concepts and propositions and thus a theory is called generalization. The notion of generalization in the interpretive perspective is different from the one understood by quantitative researchers. Generalization, as perceived by the positivist paradigm, relies on the representativeness of the sample with regard to a particular population. However, the validity of generalization in the interpretive paradigm depends on "the plausibility and cogency of the logical reasoning used in describing the results from the cases and drawing the conclusions from them" (Walsham, 1993, 15). Although my thesis is documenting a limited number of cases, i.e. the views of a number of software developers about software development, as they have

experienced it, the interpretive paradigm (as mentioned above) provides a philosophical grounding for generalization and abstraction.

Having developed a set of concepts and propositions from the online data in the data analysis stage, the next step is to present these concepts and propositions in a comprehensible shape. This final shape would represent the generalisation of the findings from the online communities. According to Walsham (1995) there are four ways in which one can generalize interpretive, qualitative data:

- The development of concepts
- The generation of theory (like the model of social issues in software development developed in this thesis, (my addition))
- The drawing of specific implications
- The contribution of rich insight into developers' work

Source: Walsham (1995, 79-80)

Following the aim and research questions of this thesis, the objective of the process of data analysis and theory building was a theory based model which showed how developers perceive and are affected by the human and social issues that surround them when they develop software for complex, adaptive, unpredictable and multi-group settings. As the concern of this thesis can be applied in many organisational settings the theory building approach was the more suitable to generalise the findings. The process of creation of this model is shown in chapters 5 to 7 of this thesis. Also an analysis of the theoretical and methodological contributions of this thesis (the model and the process of its creation) is provided in section 8.2 of chapter 8. This analysis follows Phillips and Pugh (2000) ideas about original contributions to knowledge.

4.7 Chapter Summary

In this chapter I have described and justified the selection of the research approach. The research approach is qualitative and interpretive chosen because it provides access to insights to developers' perceptions about the human and social issues within the software development environment. The method for this research is online ethnography of virtual communities of software developers, and the methods for data analysis are conversation and content analysis. The aim of the data analysis stage is theory building valid because the phenomena studied could represent more than one situation. Theory building is part of the process of analysis of the data collected from these online communities. Theory building is performed iteratively until the representation of the phenomenon, i.e., the model, is tested and refined (Gioia and Pitre, 1990, 587). Abstract concepts are generated from the categories generated in step 4 of the organising system of data and connect to the research aim and research questions. To do this, I chose a theory building strategy that corresponds to the research approach and the nature of the phenomena studied (Gioia and Pitre, 1990, 587). This strategy is inductive in nature. Having collected fieldwork data, I let the model emerge from the data with few a priori ideas. The theory based model emerged in the form of general propositions and relationships describing and explaining my view of the interpretations (Walsham, 1995, 78) of software developers about the nature of the process of software development. Because the subjective nature of interpretive research, the fieldwork data show, not the phenomenon itself, but the interpretations of the phenomenon by software developer or as Lee (1991) puts it, the phenomenon is shown "in terms of what it means to the observed people" (Lee, 1991, 347). Similarly, the subjective nature of the analysis and theory building processes generates concepts based on my interpretations of the interpretations of the subjects of study.

Next, in part III of the thesis, I will discus the process and results of data collection, data analysis and theory building which follow the strategy explained in this chapter. Figure 4.8 explains the purposes and relationships between these three chapters. Chapter 5 discusses the practicalities of the process of data management during the online ethnography. It also explains the process of data categorisation. Chapter 6 contains a description and discussion of the findings organised by categories. Finally chapter 7 shows the moulding of those categories into a theory based model.

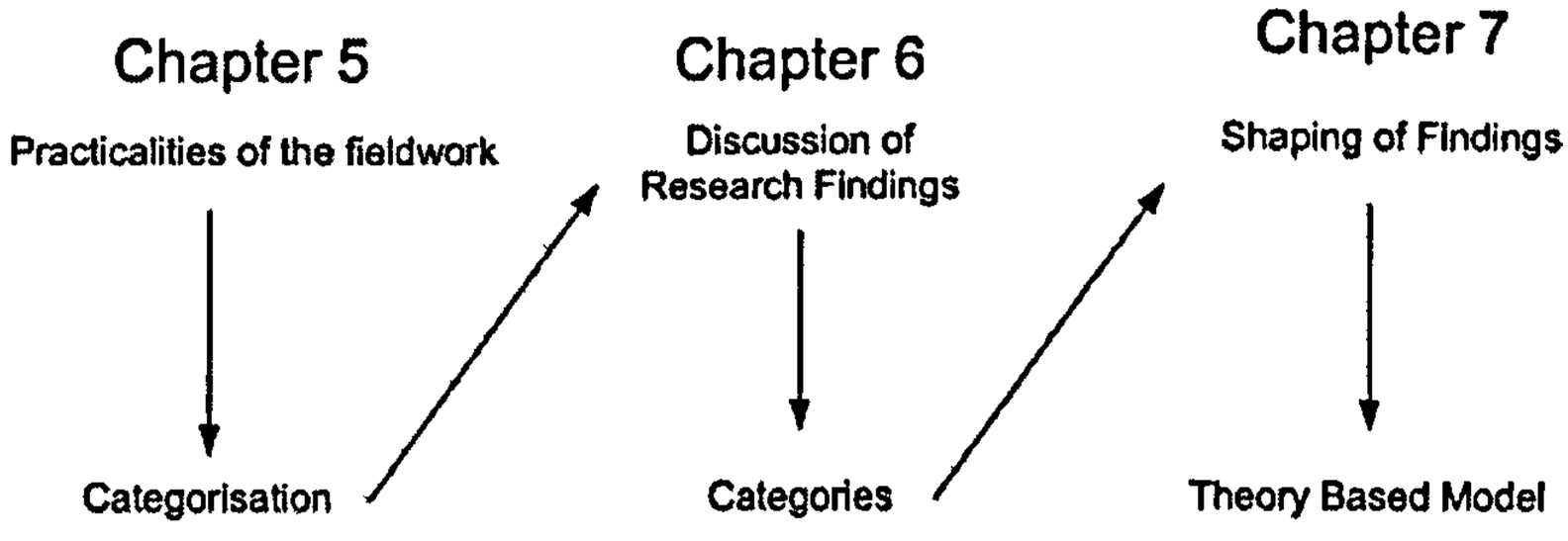


Figure 4.8 Presentation of Findings

PART III: FINDINGS AND CONCLUSIONS

Part III of this thesis focuses on describing and explaining the fieldwork, data analysis and theory generation as described and justified in part II. It presents the results of this thesis in the form of a descriptive, theory based model, which complements Orlikowski's model of enactment of technologies-in-practice. It finally presents the conclusions of this thesis.

Chapter 5: Description of the Fieldwork

In this chapter I discuss the process of data collection and analysis and focus on the practicalities of the online ethnography. As in qualitative research "there are no clearly agreed rules" (Spencer et al., 2003, 200) the researcher needs to design her own method for collecting data that provides reliability and validity to the research. Therefore here I explain how I designed and carried out the fieldwork for this study. I have divided this chapter in two sections; each one corresponding to one of the two interrelated aspects I found in online ethnography: participation in online communities and analysis of data. I conceive online ethnography as one endeavour consisting of two parallel activities which follow the phenomenological and hermeneutical approaches to interpretive research. The first activity is my participation in the virtual communities of software developers, my interactions with online members and collecting data relevant to the study. These data are the thoughts or interpretations of software developers about complex business situations and software development. I concentrate on the adaptive nature of online ethnography (Hine, 2000, 65), that is, the way I decided to do the ethnography on a daily basis, the kind of topics in vogue, how I decided on the relevancy of these topics, what additional reading I had to do to catch up with current topics, etc. Thus this section 5.1 is called the management of the online ethnography.

The second activity is analysing the data and developing concepts and propositions that respond to the aims of the research. That is, interpreting the phenomenon: software development for complex business situations in terms of the meanings developers bring to it (Denzin and Lincoln, 1998, 3). As the data analysis progressed the online interactions became more focused. Having started with an open-ended perspective, themes emerging from the data helped me to refine the research questions and to concentrate on the online interactions that were more relevant to them. I explain in section 5.2 how I developed

categories and descriptions out of these themes from the data collected. These two activities are mutually dependant as progress in the analysis of data guides further online interactions and data collection; and the continuous interactions in the field provide more material for analysis and give the researcher a sense of attachment not only to the virtual field but to the topics being discussed. The cycle ended when I reached theoretical saturation, that is, when new additions of data contributed little or nothing to the understanding of the topic (Carroll and Swatman, 2000, 240).

5.1 Management of the Online Ethnography

The online ethnography was a long endeavour with a variety of activities needing frequent management. Depending on the particularities of the day-to-day experiences – the kind and number of interactions happening online - I had to arrange my activities for the day or week ahead. Some days interactions were numerous 7 and interesting and I could hardly read all the conversations and participate in them. Some other days there were very few of them if not any. In this section I give an account of the adaptive routine I undertook at different stages of the fieldwork. In the early stages I had a more exploratory routine, in which I was open to any topic that could rise in a forum. In later stages my routine depended on the analysis and themes arising from data.

5.1.1 Ethnography at the early stages.

I started the ethnography with data collection concentrated on four online forums: Joel on Software (JoS), Codeguru, VB Forums and Microsoft discussion groups (see chapter 4 for a discussion of virtual communities and selection criteria). In these 4 forums I found a wide variety of themes or "recurring ideas" (Marshall and Rossman, 1989, 116) being discussed, from the very technical ones (e.g. programming code) to the social ones,

and from the software development related ones to the leisure ones. I read hundreds of online documents; however, I only collected the relevant ones for this research, as I will explain in this chapter. The actual volume of data collected is discussed in section 5.2.1. From this richness of data, I began to organise the themes and to relate them to the characteristics of the problem situation that motivated this research. This is a combined approach suggested by Miles and Huberman (1994, 57). Themes are created inductively from the data but the process is informed by the description of the problem situation presented in the introduction of this thesis. The result of this was a map which helped me to build a picture of the kind of material I had in hand as well as the kind of topics I would find being discussed elsewhere on the web. In this way I had an initial criteria to assess the relevance of the conversations and did not lose track of the aim of this research:

> to identify and explain how the social issues within the software development environment and the social aspects of complex organisations shape software and the process of software development.

Later I used this map to develop the categories for the analysis phase (discussed in section 5.2 of this chapter) as it provided a natural flow from the initial stages of data collection to the development of ideas to answer the research questions. The map of themes is shown in figure 5.1. (This map acted as the original tree of categories in NUD*IST).

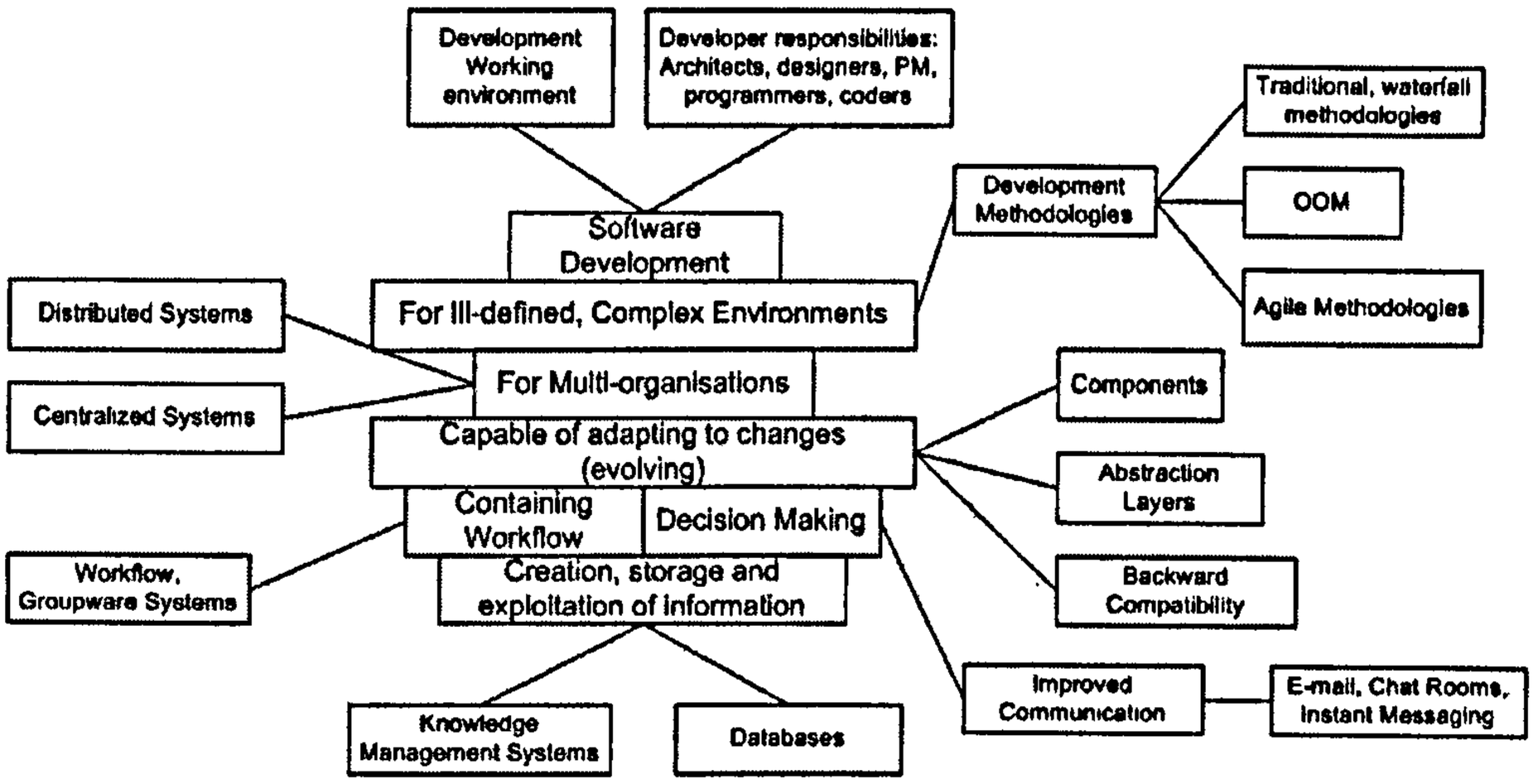


Figure 5.1 Map of Themes from Online Forums at early stages of the fieldwork

In the centre of the map, the most important characteristics of the Model of Complex Organisations (see figure 1.2 in chapter 1), as presented in the introduction of this thesis, are connected to the themes that emerged from the online conversations. To understand the data it may be helpful to give a brief examination of these initial themes as they were found:

Software Development: topics discussed in this area were about how developers feel at work, how they deal with project managers and colleagues, how they improve their IT skills, what they think about themselves and what they think are their responsibilities (designing, documenting, coding, etc.)

Ill-defined, complex environments: discussions here flowed around development methodologies which support better specifications, better coding, and hence better products. "Agile methodologies" was a burning topic in this area. Such methodologies are designed for rapid, incremental development which can adapt rapidly to changes.

Multi-organisations: As this thesis concerns multiple locations for information systems (multi-organisation, multi-group) I also

considered discussions about centralised and distributed systems. My preliminary thought was that distributed systems resemble multi-participant processes and can propagate information and support functionalities to the edges of the network, where the users reside. Distributed systems have their functions dispersed across many servers and are not dependent on one source. Centralised systems concentrate functions in one location. For example applications and databases are stored in one server.

Adaptation to changes: discussions in this area flow around two areas: software maintenance and user customizable software. Most developers think of adaptation and evolution of software as consecutive updates of software or software maintenance. There was little interest in allowing users to customize their software to fit changes, perhaps because developers are more interested in programming activities (either because it they think it is the core of their work). Developers compare the programming languages with high and low levels of abstraction in terms of how difficult it is to develop and maintain software. Level of abstraction is the extent to which a programming language provides pre-coded low-level functionalities allowing developers to focus on high level functional specifications. The higher the level of abstraction the more pre-built functionality (code, components) that is available to the developer. However at lower levels of abstraction developers can customise more the code to fit clients' needs.

Workflow and decision making: themes focused on the characteristics and use of web technologies, such as chat rooms, instant messaging, and some Microsoft products like SharePoint, Exchange and Outlook for collaboration. There was very little about design techniques for these kinds of software.

Creation, storage and exploitation of information: very little. Themes are knowledge management tools and relational

databases. In both cases attention is paid to the technical issues such as platform compatibility and performance.

This map of themes grew and gained consistency when my interactions in online forums led me to additional communications. These were materials that the online members conversed about but which were not published in the forums but elsewhere in the web or books. Developers recommended other forums, weblogs, or web sites that they found interesting and that were related to the topic of conversation. Some others would recommend books that they liked at university or that they found useful for their jobs. This horizontal integration inherent in online ethnography did help me to make more sense of the field. Below there are two examples of additional material being recommended to me. Extract of data 5.1 (extracts are presented in their original form, spelling and grammatical constructions have not been altered.) belongs to a conversation in JoS in which I asked the online members about hiring consultants to develop customised development methodologies. One member sent me a link to an entry in a weblog with a funny story about a developer having to design an optical scanning system for a clerk who needed to file a dozen pages every month or two[3] . This was a sarcastic way of saying that developers' skills are wasted when organisations do not know what they have and what they need.

Extract 5.1

Thread title: Five Worlds	Date: 18th October 2004	Source: JoS

"May I recommend http://mthollywood.blogspot.com/ ? He has some interesting stories to tell about the real world. Some of his stories eerily mirror some of my experiences."

Peter

"Peter, thanks for the link... that story about the optical scanning system is fantastic. It reminded me some of my own experiences."

Cecilia Loureiro

[3] Peter recommended the whole weblog to me as it contained interesting stories. However, at the time of the conversation, I read only the story that was on the top of the web site and made reference to it when I thanked him for the link. The link in the extract is a link to the weblog's main page which always shows the last entry and to which I came back some times to read other stories. As a different home page may be shown at the time of writing this thesis I am providing here the permanent link to that entry http://mthollywood.blogspot.com/2004/10/captain-ahab-xvii-i-got-assignment.html

The second text extract is from a thread in Microsoft discussion groups in which I asked members about their experiences at dealing with users and management. One of the members recommended some books to me, one of which, he thought, provides a useful methodology to gather requirements and hence deal with users. The original text is in Spanish (left) and I have translated it into English (right).

Extract 5.2

Thread title: Factores humanos que afectan nuestro trabajo (Human factors that affect our work)	**Date: 9th February 2005**	**Source: Microsoft discussion groups (in Spanish)**
"Tal vez ya hayas caminado por UP, no lo se, te recomiendo que leas EL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE de Jacobson, Booch y Rumbauch. ISBN84-7829-036-2. Es un libro para leer lentamente, repasando y reconfirmando lo leido. (Es un poco tedioso.) Te recomiendo el capitulo 6 Captura de requisitos: de la visión a los requisitos." Daniel A. Calvin MCP	"Perhaps you have already gone through UP, I don't know, I recommend you to read "THE UNIFIED PROCESS OF SOFTWARE DEVELOPMENT" by Booch and Rumbauch. ISBN84-7829-036-2 It is a book to read slowly, rereading and confirming all what you've read. (It is a bit tedious.) I recommend you chapter 6, Capturing requisites: from the vision to the requisites." Daniel A. Calvin MCP	
"Daniel, muchas gracias por las referencias. No, no he leído los libros. Pero si son como dices creo que tendré que echarles una miradita. ;)" Cecilia Loureiro	"Daniel, thanks very much for the references. No, I have not read the books. But, if they are as you say I think I will have a look at them ;)" Cecilia Loureiro	

5.1.2 Ethnography at the later stages

After five months of concentration on data gathering I began analysis alongside further data collection. I entered my data into NUD*IST v4.0 and started the segmentation of texts and generation of categories. Both segmentation of texts and category segmentation were informed by conversation and content analysis. Segments of data correspond to recording units, discussed in section 4.4.2 of chapter 4. Record units were identified by using both data analysis approaches. For instance, a

segment of data from an online conversation is a turn or contribution in that conversation whose content describes a theme. However, according to conversation analysis a turn may not make sense if the context of the conversation is not given, for example by a previous turn. In extract 5.2, there are two turns belonging to the same conversation. The second turn (my answer) would not make sense if I have not provided the first segment.

As mentioned in section 5.1.1, the map of themes was used as a starting point. However the final result is different from it (see findings in chapter 6). Data collection ended when I reached data saturation, that is, when data analysis told me that no new themes were adding to the understanding of the topic. My strategy during the later stages of data collection was to be more selective with topics of conversation and to search for probe topics or to ask probe questions (Goode and Hatt, 1952, 201) about themes I was not sure about. As categories were developing sometimes I needed more material to better understand and describe them. Probe questions were a good tool to establish the relevance of some categories to the concern of software development for complex organisations and to increase the validity of this study's data (see section 4.1 of chapter 4 for a discussion of validity of data). By using probe questions, I discarded some themes and confirmed others. In NUD*IST this was done by removing the node (i.e., category) from the list of nodes in the coding facility (see a snapshot in figure (c) of appendix 7). The resulting categories and descriptions represented a consensus view (among the online participants) on human and social aspects of software development.

The following example shows how I discarded the themes "centralised and distributed systems" which were in the initial map of themes. The extract of data shows part of a conversation that I started in which I compared distributed technologies with adaptive, multi-organisation configurations and asked for feedback. This online conversation took part at the beginning of the fieldwork but the decision for withdrawing

"distributed systems" as a theme was made months later during analysis. While the conversation was longer, I am showing only the comments made by "bah humbug" and "Just me (Sir to you)" as they summarise most of the replies. Although at the beginning I believed that distributed technologies as opposed to centralised ones could resemble multi-participant processes, this turned out not to be necessarily true. "Distributed systems", as used by software developers, is a very technical term. It does not represent or emulate multi-organisation business processes but the processing of objects in different computers. The reason for choosing a distributed technology is not only the physical distribution of localities but the availability of technical resources and architecture (as in web servers). Even centralised technology solutions are often preferred for multi-group kinds of organisational configurations. So my focus on people being distributed was not represented by processing being distributed. The assertion made by bah humbug in extract 5.2: "Just because you _can_ do something (or might be able to) doesn't mean you _should_" is the one which made me realise that technology is not and should not mimic human systems (i.e. organisations) if there are other simpler, cheaper alternatives. The words "can" and "should" were emphasised by the poster (by using the "_" symbol) making it a stronger argument (conversation analysis).

Extract 5.3

Thread title: Distributed systems	Date: 24th May 2004	Source: JoS

"…I am thinking about a software framework of decision components that can be chosen on-the-fly by users as they are dealing with a situation. There would be components for communication, to access data, documents, models, to do calculations, to build reports, etc. …

I am thinking more about a federation of pc's connected by the system, each of them containing a component of the decision system. …

Is that possible? Are distributed technologies going in that direction?..."

Cecilia Loureiro

"I'm stuck with the question of 'why bother?' I can't see that this would actually do anything useful in a significantly better fashion than existing systems, and would add lots of bizarre complexities.

Just because you _can_ do something (or might be able to) doesn't mean you _should_. The PC itself was an example of this: 'let's break away from centralised management and control and put anarchy on the desktops'. Now there is a definite trend back to centralised management and control."

bah humbug

"Now the need for distribution can be a non-technical requirement. There might be operational restrictions, aquisition-control issues, policies, autonomy ... many things that force one into a "distributed" scenario where technically a central sytem would suffice. There can also be technical factors: offline scenarios, unique infrastructure demands, scalability ... that would necessitate distribution. But distribution is often a costly thing. You add a whole extra layer of complexity to your system that does not come cheap. Simply guessing "it might emulate multi-organisations better" does not in itself seem to support a good ROI for this."

Just me (Sir to you)

The next example shows how I used probe questions to confirm the relevancy of the developers and developers' responsibilities themes. The text extract shows my question and some of the responses I got when I asked the other developers to define themselves. These replies showed me that there are different levels of responsibilities (coders, analysts, architects, etc.) for developers, and that in some instances only analysts would know the target organisations and information systems for which they are designing the software. Programmers or coders will only dedicate their time to code, and will be unaware of the particular complexities happening in organisations. Sometimes analysts would not talk to coders or there will not be analysts at all and only coders will be in charge of software design and programming. This and similar conversations led me to think that having situations in which no consistent analysis and design of software in line with organisational

needs is done may be a cause for faulty software in the industry. Additionally, these kinds of conversations also showed me that there are other social issues within the development environment that may have some impact on how software development is carried out.

Extract 5.4

Thread title: What is a developer?	Date: 17th June 2004	Source: VB Forums

"Hi,
I am a bit confused about what a developer does and doesn't do. I've heard of analysis of requirements, technical spec. and programming. I've also heard about types of applications as for example, ERP, Business Intelligence, intranets, operating systems, finally I've heard about project managers, analysts, programmers, etc.... what is a software developer? What should a developer know to develop software?..."

Cecilia Loureiro

"I've been in the computer programming field for 25 years. I started as just a coder. I took direction from the CTO of the company, on what his vision was and then delivered just that.
After a few years, the company grew and I became the "R & D Group Leader". That meant I had 6 programmers working for me and I pretty much was an "analyst" - but giving me that title probably would have rubbed the CTO wrong...
Beside the 6 programmers I even had a librarian - very helpful in the group - kept our code in libraries - did our backups.
I've been on my own now for 15+ years. That means I'm both the analyst and the coder. I have some high school kids working for me - they code for me as well - since we have so much code to deliver.
I've worked on larges teams at insurance companies where the structure was very strict.
Analysts met to decide what the coders were going to do.
Coders did some work.
Coders met weekly to "peer review" each others work
After a routine/program was deamed to be done, it was "peer reviewed" for release and then delivered to the "production team"
We never even met the "production team" - they were on a different floor.
If they found a problem - you were toast, and the whole cycle started again.
But that paid $150/hr - so I would jump through hoops for that."

Szlamany

"Yes! Exactly.
Once you get down to the developer level. You are pretty much working either in a mop closet or stinky horse stable.
They feed you every other day and you must meet a daily coding quota. Mine is 600 lines of code a day. Or else I am trampled by the owners stallion in the next stall.
Occasionally, they let us outside for some exercise, but it ususally consists of running in circles with a rope tied to around our throats.
It isn't a bad existence. It could be worse. I could be a database adminstrator!....arrggghhhh...I get chills just thinking about their job...It's just too gruesome to discuss.
Anyway, my master...err...boss is coming and I must leave, before he whips me again..."

Memnoch1207

5.2 Data Analysis Process

In this section I will describe the process I followed to analyse the data from the online forums. My approach to data analysis was exploratory. I thoroughly examined the texts from online conversations, weblogs and web sites. I looked for recurring themes that could lead me to the generation of insights into the interpretations of software developers on software development for complex settings. The results from this analysis are presented in section 5.2. Data analysis throughout during all the ethnographic intervention either as a background process (at the early stages) or as a main activity (at the later stages). However, for explanatory purposes I am defining the data analysis stage as starting with data coding in NUD*IST and ending with the development of concepts and propositions for each category and research question. Further developments of these concepts and propositions into a descriptive and explanatory theory are presented in chapter 6. Following the strategy defined in chapter 4, the process of analysis was carried out in an iterative and incremental fashion (see figure 5.2). Texts were examined repeatedly in the search for themes. The context of analysis changed with every interaction as the focus switched from the online conversations to the research questions.

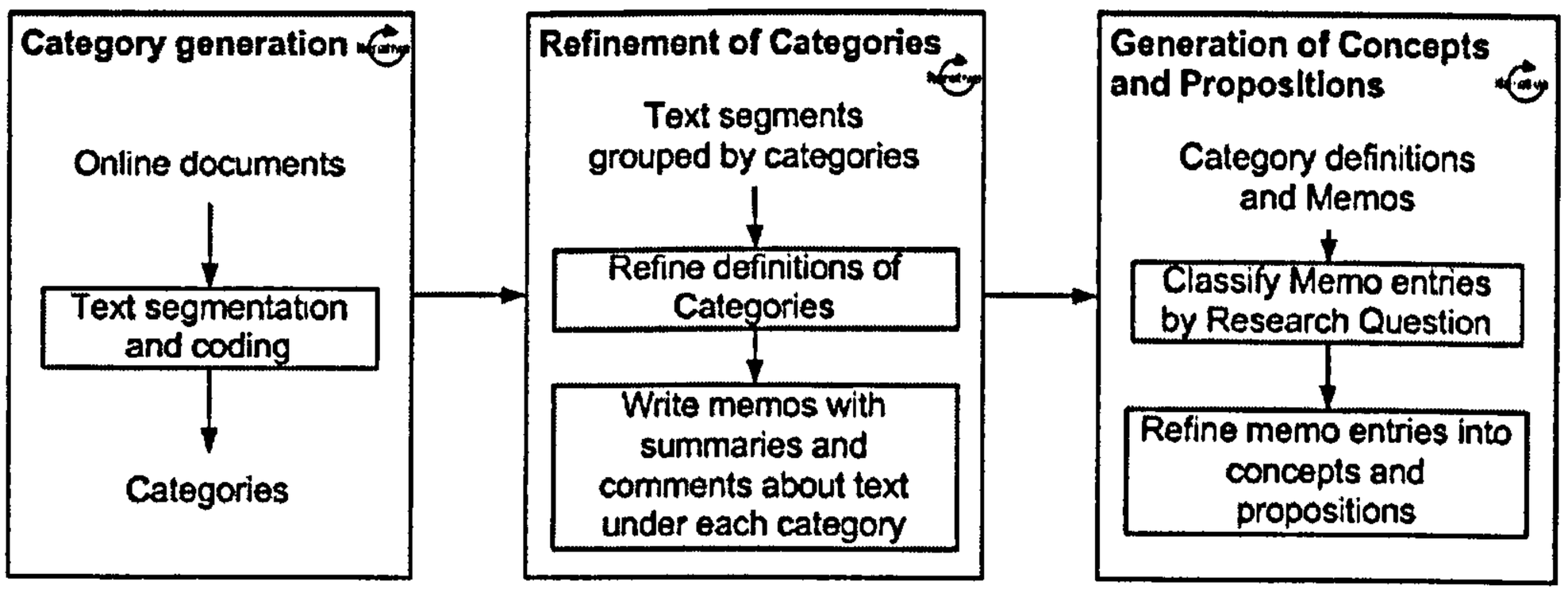


Figure 5.2 Data Analysis Process

5.2.1 Generation of categories

I entered 107 online documents into NUD*IST, the shortest one being an online conversation with two contributions (1 question and 1 answer) and the longest one a conversation with 60 contributions. The data collected is only a small fraction of the several hundreds of online conversations I read during my fieldwork. The number of people involved in those conversations is difficult to calculate as sometimes some online participants can contribute more than one time in one conversation. Also participants in virtual communities can use different names every time they contribute with a post (see section 4.2.2 of chapter 2 for a discussion of linguistic and pragmatic issues including the anonymity of the online media). However, setting this aside, I would say that 500 is a good estimate of the number of people who participated in the online conversations I collected in this study.

From the 107 documents, I "isolated certain themes and expressions" (Miles and Huberman, 1994, 6) that were relevant to the research purpose (Tesch, 1990, 79). Segments of texts corresponding to these themes were coded in NUD*IST under 72 categories (see category tree in appendix 2). Categorisation in NUD*IST was done by selecting one document at a time from the Document Explorer facility (see a screenshot of the Document Explorer in figure (b) in Appendix 7). Once selected, I opened the document in the Document Browser (see a screenshot of the Document Browser in figure (c) in Appendix 7) and read their content. Whenever I found a segment of text which was relevant to the concern of this thesis, (see recording unit in section 4.4.2 of chapter 4), I selected that segment and accessed the coding facility (as shown in figure (c) in Appendix 7). If the category already existed I selected it from the list of nodes (i.e., categories) provided. If the category was not in the list, NUD*IST allowed me to create a new one from the same coding facility.

Some segments were coded in more than one category. The next extract of data shows two segments of text taken from one online conversation about documentation. Both segments were coded at two categories: [SW-development/ Process/ Communication] and [SW-development/ Stages/ Documentation]. The online participant expresses his opinion about oral or verbal documentation. In the context of this conversation, oral or verbal documentation mean oral or verbal communication (conversation analysis states that in order to understand a term or an utterance within a conversation the context of the whole conversation must be considered). According to him oral documentation does not guarantee good communication especially in the long term when maintenance work needs to be done, probably by a developer who was not in the original team. By connecting this matter to the concern of complex organisations I find that documentation of processes and information, and communication between developers and users are relevant themes to the aim of the research. They involve the establishment of a relationship between developers and users, the understanding and modelling of business processes and the transmission of that knowledge to other members of the development team.

Extract 5.5

Thread title: Oral Documentation - Not Worth the Paper It's Written On	Date: 10th June 2004	Source: Hacknot

"Forgive me for questioning a holy proclamation, but isn't it rather well established that verbal communication is often incomplete and ambiguous, and that human memory is inaccurate and prone to confabulation? The plethora of psychological research in such areas as false memories, the veracity of eyewitness testimony, and the effect of predisposition on the interpretation of sensory data has surely given us a big hint that our perceptual and communicative capabilities are erratic and dubitable?

So where comes the apparently wide spread acceptance of (or at least, lack of challenge to) such outrageous Agile sophistry? For my part, it is difficult to ignore the manifest problems associated with a development team's reliance upon face-to-face communication. Over the last 3 or 4 months, as the inheritor of a code base whose authors preferred the "verbal tradition" style of documentation, I suffer daily from the flow-on effects of this laziness. Let me illustrate by providing you with a summary [the summary is in appendix 1] of a typical day for me in recent months, so you too can marvel at the feel-good richness and super-duper efficiency of face to face communication amongst software developers."

Mr, Ed

As the example above shows, categories were defined in the context of the online conversations from which the segments were extracted. Every new addition of texts to the system of categories involved a revision of the definition of current categories or the creation of a new one. Categories were arranged in branches some of which had more sub-levels of branches. This tree of categories went through several changes during the analysis process, starting from the map of themes (see figure 5.1) to the final version which is shown and discussed in chapter 6. The top level categories and their definitions are shown below:

1. *Software Development*: creating and maintaining software applications. (This research concentrates on the software applications that serve information systems.)
2. *Software Engineering*: the application of a systematic, disciplined, quantifiable approach to the development of software.
3. *Software*: Computer program designed to support people at organisations.
4. *Developers*: anyone involved in any of the stages of software development (architect, business and system analyst, coder, etc.)
5. *Methodologies*: set of practices that are carried out to produce software.
6. *Business Issues*: aspects of organisations that software developers should learn about before or during software development.
7. *Social Software*: software that allows or encourages social interactions.
8. *Virtual communities*: online places where software developers share working experiences.

Note: the data from the category [Virtual Communities] are not part of the analysis results. I organised this category to help me make sense of the nature of the virtual communities that I was exploring. I also use this data in chapter 4 to describe virtual communities of software developers.

5.2.2 Refinement of categories

The outcome of the previous stage was a set of categories with provisional definitions. The definitions were created in line with the contexts of the online conversations from which the segments were extracted. During the refinement of categories, the context of analysis changed from the online documents to the categories themselves. Therefore, from here, content analysis began to play a relatively major role over conversation analysis. I read again all coded texts grouped by categories and refined the definition of those categories at a more abstract level. NUD*IST provides a Node Browser which shows all the segments of text coded under a category. I concentrated on the meaning of the categories and their relationship to the concern of this thesis, regardless of the original context of the online documents. Additionally, I summarised the category texts and commented on the implications to the research in the Memo facility of NUD*IST. (In NUD*IST both, documents and nodes can have memos attached, however I found writing memos for nodes more useful to refine categories than commenting on the documents.) Below I present an example of this. The text extract contains the definition of the category [Software Development/ Process/ Improvement], one of the ten comments from its memo and two of the segments of text that correspond to that entry.

Extract 5.6

Category [Software Development/Process/Improvement]		
Definition: Tips on how to improve practices in software development. **Comment from memo**: Improvements in software development practices, like with new methodologies, need to consider the actual practices to make change effective.		
Thread title: UI Prototyping vs. Use Case Mode	**Date: 3rd Sept 2004**	**Source: JoS**
"But to address your situation... it sounds like you're in a situation where things are done a certain way "because." Because they always have been done that way; because someone decided they would be done that way; because nobody knows any other way; because they read a book by Booch and went nuts. The only thing they're NOT doing is thinking about why they're doing what they're doing. What can be done? It depends on your position. If you're well respected in that group and it's not highly political, you can get things done by advocating them. In a political organization or one in which you're less senior or have less clout, the more you suggest change, the more frustrated you'll get!" Joel Spolsky		
Thread title: What design methodology(TM) do you use?	**Date: 2nd March 2004**	**Source: JoS**
"This being said, the first thing you should do is understand your current process (yes, even if informal and undocumented you _are_ following a process)! Once you know how you are currently doing your business (probably software development), you can start looking at how the practices described in the RUP can help improve your process. Too many people go the other way and try to simply look at RUP and define what their process _should_ be. This is a recipe for disaster. It is difficult to get somewhere when you do not know where you are!" Selrahc		

The category [Software Development/Process/Improvement] addresses the actual issues that concern developers and the measures they take to deal with them. I defined this category as "tips" because in my opinion this is the most common way of giving advice online. One developer will give informal advices to the others mainly in the form of: "this worked for me in these circumstances." In the case of the example above (extract 5.6) as I see it, both developers have learned from bad experiences with people who wanted to change their practices. The problem according to them was that before attempting to change or improve their practices developers should be able to articulate how and why things are being done in the present. Only in this way they would know the problems that need to be addressed and how the new practices may cope with them. (This is an explicit example that shows how the present research is a result of my interpretations of software developers' interpretations).

5.2.3 Generation of concepts and propositions

Having defined and written memos for all the categories I went through all of them again to compare them with the three research questions. The purpose of this cycle was to refine the comments into more abstract and generalisable statements that could be used to build a model of software development (shown in chapter 7). I classified the comments in the memos by the research question(s) they were related to. This part of the analysis process is not supported by NUD*IST 4, (memos cannot be coded or classified) and I had to do it manually. I went through all the memos in NUD*IST one by one and wrote next to each comment the research question numbers to which the comment was related. Then I copied and pasted the memos in a word document and organised them by research question. I reviewed all the comments, this time in the context of the research questions and research aim. As the comments originated from the online conversations, the purpose of this stage was to accommodate and focus the comments on the research aim. This made them less specific to the particular contexts of the online conversations and more applicable to explain more situations. The outcome of this stage was a more refined set of comments under each research question containing "propositions" about "concepts" related to software development for complex situations. The propositions are "general statements grounded in the data" (Taylor and Bogolan, 1984, 134). They explain or theorize the relationships between concepts (Tesch, 1990, 85) and are the foundations for theory building. Concepts are "abstract ideas generalized from empirical facts" (Taylor and Bogolan, 1984, 133) and include abstract versions of the descriptions of the categories themselves, and of the comments from the fieldwork data under each category. As these concepts are at a higher level of abstraction, they may stand for other similar pieces of data (Tesch, 1990, 138), i.e. they "may be valuable in the future to explain other organizations and contexts" (Walsham, 1995, 79).

Below there is an example of concept and proposition generation. The extract contains data about the concept "free social interactions among users" from the analysis stage. Freedom of interaction and power of decision within complex organisations are matters mentioned in the introduction of this thesis. The model of complex organisations shows that processes can take any shape or number of stages depending on how users of software decide to act. Processes might be started or ended by any actor in the organisation(s) and any number of people or groups may be involved. Data from virtual communities explain this concern from six different points of view. As freedom of interactions is a characteristic of complex organisations, this concept was connected to the first research question.

Extract 5.7

Research Question	**What are the human and social aspects of business environments that make software development complex and difficult?**
Concept	**Free (social) interactions among users**
Informal, covert practices	[SW-Dev/Characteristics/Complexity]
Users who do not follow standard procedures	[SW-Dev/Characteristics/Standardability]
Users cannot be controlled as computers	[SW-Dev/Characteristics/Controllability] [Software/Characteristics/Flexibility]
Different people doing the same jobs but in different ways	[SW-Dev/Characteristics/Complexity]
Difficulty of knowing key users, people who control the information, people who possess the domain knowledge, and their relationships.	[SW-Dev/Characteristics/Controllability]
Help users to adapt to any software and make a good use of it even if it's not 100% fit to their needs.	[SW-Dev/Characteristics/Complexity] [Software/Characteristics/Complexity] [Software/Characteristics/Flexibility] [Software/Characteristics/Adaptability] [Software/Characteristics/Social interface] [Social Software/Design]

The complete set of concepts and propositions under each research question is presented in appendix 3.

5.3 Chapter Summary

In this chapter I have described the practicalities of the online ethnography I carried out. I have explained the data collection and data analysis processes, as informed by content and conversation analysis, and described how abstract and general concepts and propositions were created from data. Figure 5.3 summarises the process of evolution of the findings of this research. I used a mixed approach to creating categories (Miles and Huberman, 1994, 57). Categories were created inductively from data. However, the process was also informed by the issues presented as part of the model of complex organisations and in the literature review. These issues informed also the research aim and questions. Category comments were created from the category definitions and their data and had the online conversations as their contexts. The final set of concepts and propositions (question comments) emerged from the category comments by considering the research questions as a new context for refinement.

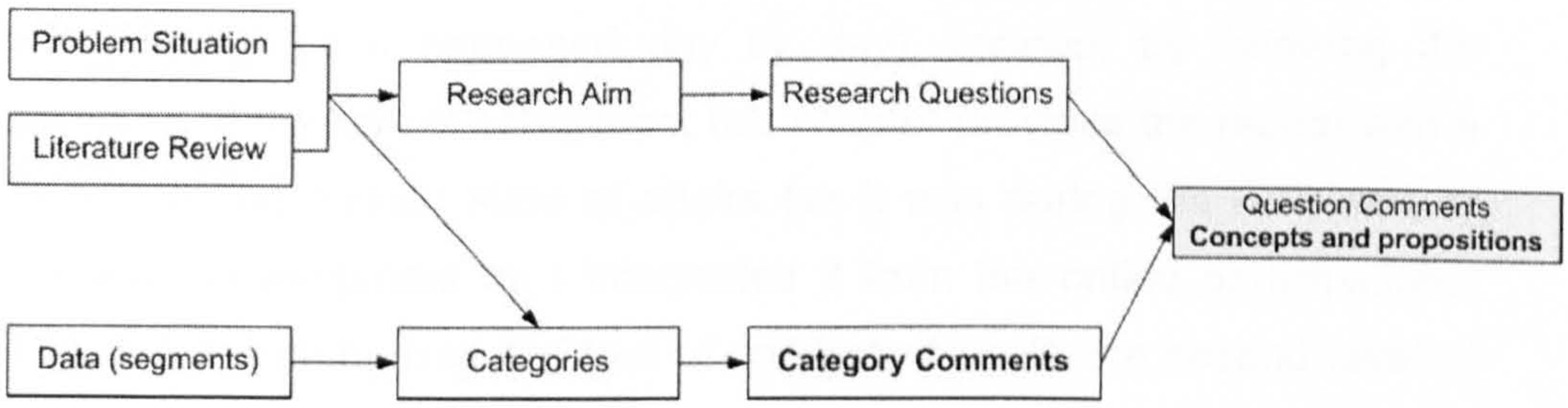


Figure 5.3 Generation of concepts and propositions

Further developments of these concepts and propositions into a descriptive and explanatory theory are presented in chapter 7. Chapter 6 will present a descriptive account of the findings organised by categories. The aim of this account is to show my interpretations of the interpretations that software developers have about software development. This descriptive account will connect data categorisation presented in this chapter and the theory based model presented in chapter 7.

Chapter 6: Research Findings: Software developers views about Software Development for complex business organisations

In this chapter I will present the findings from the online ethnography of software developers. As the approach of this thesis is qualitative interpretive, the findings presented here show my interpretations of software developers' interpretations of the field of software development. The account in this chapter tells the "story" of the online ethnography I carried out. A more detailed analysis of these findings comes in chapter 7 where a descriptive, theory based model built up from this chapter's data is presented. The descriptive material of this chapter (chapter 6) forms a bridge from data categorisation explained in chapter 5 into the explanatory model developed in chapter 7. The material collected is vast (107 online documents organised in 72 categories, approximately 1000 A4 pages in total) so I am arranging the results in 7 groups by the top-level categories defined in the data analysis phase. Therefore this account does not follow a chronological order (i.e. does not relate the ethnography as it happened day by day). Instead, by following the structure of the tree of categories, this chapter provides the reader with a picture of the current state of affairs (as it was during the fieldwork) in software development as I interpreted it from the online participations. Figure 6.1 summarises the tree of categories up to the second level of branches (36 categories). The complete tree of categories containing the three levels of branches and 72 categories is shown in appendix 2. This chapter follows the structure of figure 6.1. Starting from the category C1: Software Development, each of the top-level categories will be briefly described in the light of the online data and its relevance to the model of complex organisations. The order in which these categories are presented corresponds roughly to the order in which they emerged from data. In the next chapter, these results are discussed in line with each research question and shaped into a model of social issues in software development for complex organisations.

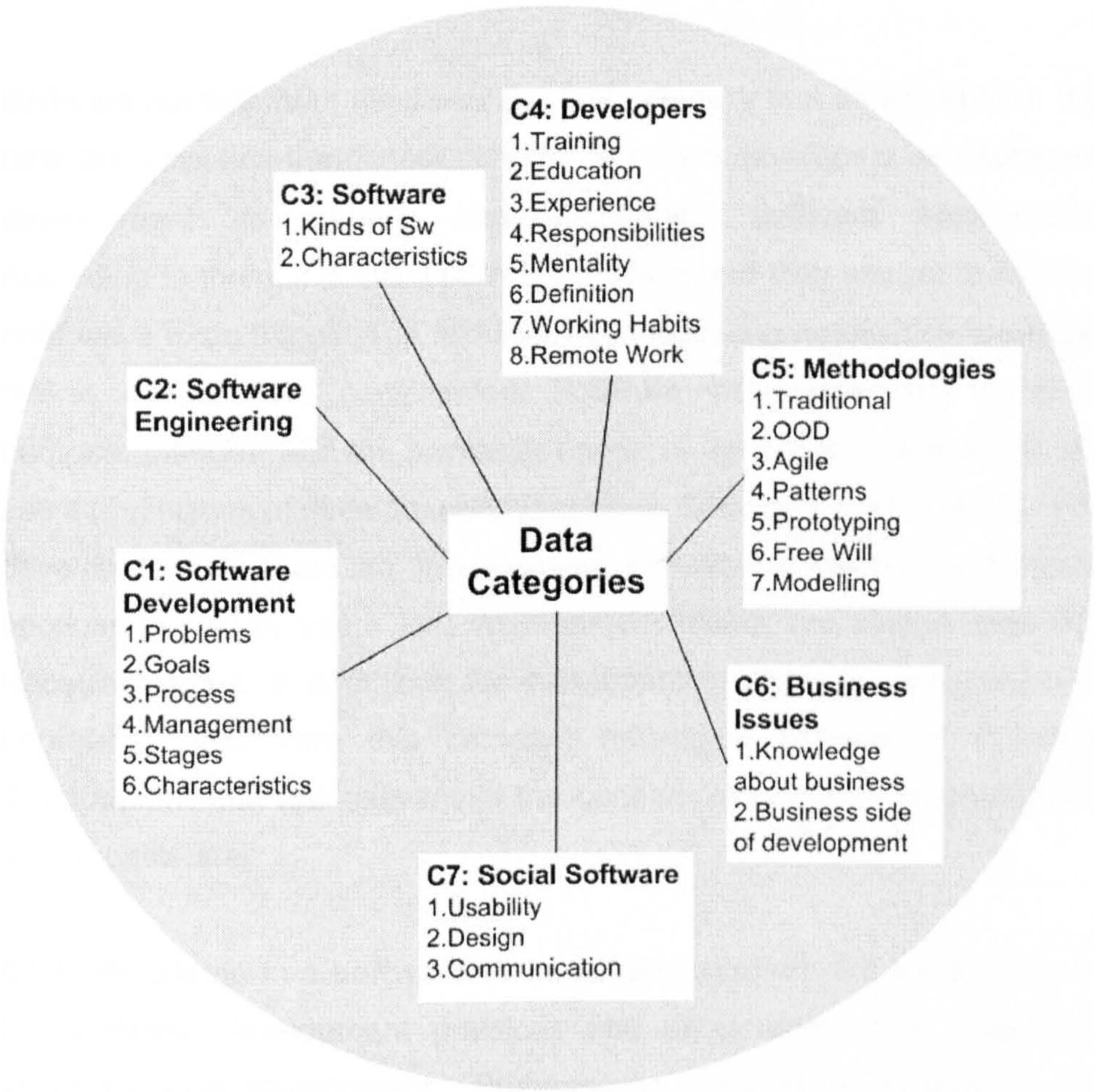


Figure 6.1 Data Categories

This first exposition of the findings is intended to provide a picture of software development (which includes views on the organisations and users they develop the software for) from the perspective of developers. This will lead to the development of the descriptive and explanatory theory in chapter 7. Seven aspects of software development are presented as relevant to complex business situations and discussed in the context of the online conversations that I collected.

6.1 C1: Software Development

Software development itself was the first category that emerged from the data and contains the richest content. Software developers see software development as creating and maintaining software applications. According to them, it is still a primitive practice and they are yet to find the right ways to do things. The fact that every software application is unique makes things more complicated. Software vary depending on their purpose, platform and the particular needs of the organisation which will use it (definitions of software and types were discussed in chapter 3). For developers, there are no two identical software development projects because software, users and development teams are always different. Because of this, it is difficult for developers to agree in one particular approach. Data from this category reflects 6 aspects of software development that are relevant to the concern of complex organisations. The aspects are:

C1-1: Problems in a software development project: are the aspects in the software development practices and environment that negatively affect software development. Problems as reported by developers are repetitive work, redundant work, lack of stability and lack of rest. Repetitive work comprises the kinds of activities that are repeated frequently in the same manner and which do not require innovative additions or modifications. Repetitive work requires low level skills and is something that could be automated by class libraries. Repetitive work is described by developers as "grunt work". For developers, development should be creative and challenging so they can apply and build better programming skills. Redundant work is work repeated or duplicated unnecessarily. Redundant work is usually the result of bad planning due to management error or user's pressure. Extract 6.1 (extracts are presented in their original form. Spelling and grammatical constructions have not been altered) provides an example of this. The online participant relates a story in which a developer was assigned a task which was not

planned or thought through by management. As it happens in these kinds of situations the developer had his software specifications changed many times and as a consequence he ended working more than was necessary.

Extract 6.1

Thread title: Worse problems with people	Date: 16th February 2004	Source: VB Forums

"Meeting, discuss what they want.
Go away develop a nice MS Access DB to their specs, do around 4 days work on it (four or five screens with validation etc).
They have a meeting on the fifth day and decide they don't want that section you've been working on.
So you sigh, comply with their wishes.
Next week another meeting, and they decide that actually they do want that section after all, but with some more information which wasn't in the original spec.

This came from a team that just had to have a database. They didn't really care at first what it held or what it did, just that all the other teams appeared to have a db of their own and in most cases it helped get the data in/reports etc."

Ecniv

The lack of stability in their jobs also affects developers' performance negatively. Many developers, especially in the United States, complain of unfair competition from off shore outsourcing. Outsourcing is growing faster and many American and European businesses are hiring Asian (mostly Indians) developers to programme their software. Western developers believe that Indian software development is cheaper but of a lower quality. For them software houses should not trust off shore skills as they do not comply with the higher western standards. In some cases these worries are transformed into a fear of outsourcing which drives the software projects. Some developers would do extra hours to keep their jobs but probably at the cost of a lower performance in the long term. Finally, developers also complain of lack of rest and working extra hours, which they also associate either with bad management or low development skills (in addition to external factors like outsourcing). Development is a much specialised job which needs lots of concentration and needs rested developers.

C1-2: Goals of a software development project: the objectives within a software development team vary. According to developers, it is unlikely that all stake holders will coincide in one set of objectives. There are always different agendas, among developers, users and management. However, after being directly asked about legitimate goals of a project, developers stated they perceived the following as legitimate goals (in alphabetical order):

To develop software:

- on time and budget
- that adds value to the business (value helps the flow of the business)
- that fits the business rule mission of organisation
- that is bug free
- that is maintainable
- that is user friendly
- that meets the customer's priorities (i.e., the person who is paying the bill)
- that works

This list shows that software developers work to satisfy many objectives. The overt objectives like the ones listed are not always the only ones that have to be accounted as there could also be hidden agendas. Hidden agendas could include making more money for management, making some users indispensable (so they cannot be fired), lengthen the project so developers can charge more billable hours, etc. This diversity of goals would influence software development in different ways. Developers would have to build the skill to recognise these different agendas and consider them when they design and programme the software. In a complex organisation, where many users or groups of users can put pressure on developers to favour their particular perspective the ability to discern how to deal with multi-objectives is invaluable.

It seems that developers perceive the issues of time and money as primary goals in the field (see extract 6.2). These issues when compared with any other one usually get favoured and technology solutions are adjusted to these constraints.

Extract 6.2

Thread title: Software lifecycle models	**Date: 27th October 2004**	**Source: JoS**
"Cecilia since time=$$$, time is always an issue. At least it is in my world." Dino		
Thread title: Worse problems with people	**Date: 16th February 2005**	**Source: VB Forums**
"My deadlines dont seem to change though and I am getting more and more major project pilled onto me [cry]" BodwadUK		

C1-3: Division of labour, coordination, communication and improvement of the development process: this category comprises four aspects of the development process which are interconnected. From the online data I can identify two general approaches to dealing with software development projects. The first approach emphasises division of labour. The development process is divided into tasks and each task is allocated to different people. Division of labour is often preferred by big, traditional methodologies which are usually used in big software houses and bureaucratic organisations. Developers specialise in some tasks of development and become experts in that area. However, this division, developers say, is an extra burden to their jobs. It causes problems of coordination and poor communication between team members. Developers tend to focus on the function they are involved with and lose track of the big picture. The second approach is to treat development as one big task and to assign the same group of developers to do the planning, design and coding as well as the testing. The advantage of this strategy is that communication and coordination of activities become easier and most importantly, it provides all the developers with exposure to the target organisation and user needs and allows them to make informed decisions at coding time. The disadvantage of this strategy is that it requires developers to have proficiency in many areas of development, something which is difficult to get. In line with this, modern

agile development methodologies focus on the improvement of coordination and communication in software development and emphasise the transmission of expertise between software development team members. Agile methodologies promote oral communication between developers and users and encourage pair programming between developers to speed up development and reduce errors. Data extract 6.3 shows the opinion of an online member who is in favour of improving development by improving communication.

Extract 6.3

Thread title: What is "extereme programming" exactly?	Date: 24th June 2004	Source: CodeGuru

"I think that we can further increase the quality of software by increasing communication between programmers and hence make it more cost effective. With increased communication, you will be better at exchanging ideas and reduce errors (assuming that there is more peer code evaluation with increased communication)."

YourSurrogateGod

C1-4: Management of a software development project: in general developers relate more their problems with management than their good experiences. Developers think of managers as ignorant of the nuances of development work. Managers urge them to finish software on time and budget and ignore other kinds of problems they have. For developers, problems like ambiguous information obtained from users, disagreements among the development team, or technical constraints also affect their performance and need to be addressed accordingly.

As noted earlier an important source of problems is redundant work which is forced by management (redundant work is the result of bad planning). Some developers albeit a small minority say they have a proactive attitude. They prefer to plan in advance some of their activities (not necessarily following a formal methodology) to avoid working on things that will not be needed. However, for these developers, management usually shows a "fire fighter" or reactive attitude. For example, they would force developers to start programming software that is not well specified yet. When the specifications are complete managers will urge developers

to do the pertinent modifications, even if it means to reprogram most of the application. Developers think that it is wasted time and resources. They would like managers to know how to prevent problems instead of being caught by surprise by them.

In connection with the previous ideas, managers' limited view of software development makes them see only the final product and the actual process of production. Managers frequently ignore the work developers have to do to obtain information, design and assess the quality of software. According to developers, managers do not really know the kind of tasks they perform probably because they haven't experienced or practiced software development themselves.

From my readings of online conversations I find that these problems could also be caused by the developers themselves, who do not acknowledge or understand the kinds of pressures and responsibilities the managers have. I noted this in many online discussions between developers and managers. It seems that the root of some of the problems could be the difference between managers' and developers' working styles and opinions (probably obtained from previous jobs). Extract 6.4 shows an example of this. An online member explains to the original poster that the cause of his problems at work was probably the mismatched views him and his manager have. Joe's text suggests to me that there is a need for managers to be more informed about developers' practices and to acknowledge different kinds of problems that could happen apart from time and budget problems. The opposite is also true, developers should be aware that managers' views would not necessarily concur with theirs.

Extract 6.4

Thread title: Monumental Incompetency- Part TWO	Date: 08th June 2004	Source: JoS

"In this case, it's going both ways. You and your PM have mismatched views of the world. To you, it's common sense that he should provide you with all the details. To him, it's common sense that you should be able to figure out the details.

The answer here lies in finding a middle ground. If you try to make him responsible for telling you every little parameter of a project up front, he'll feel like he's babysitting you, which won't put you on his good side. But if you don't get enough info, you'll inevitabely do something "wrong" because you weren't aware of how it should be done."

Joe

C1-5: Stages of a project: this category shows, following the division of labour perspective, the different stages developers have to go through to develop software. Although the great majority of developers describe themselves as programmers or as concerned only with programming, there is consent that there is more work to software development than just programming. From the data in forums I have divided software development in 5 stages, however, this division does not necessarily match all developers' views:

- **Planning**: for developers a plan is a scheme, programme or method worked out beforehand for the accomplishment of a development project. Developers think that planning is a delicate business. In general developers who have to plan have to be on the safe side. That is, they believe they should never offer any estimate before they gather all the requirements and spend some time with them.
- **Analysis**: is the examination and study of the existing state of affairs in an organisational area. The analysis stage is usually used to clarify ambiguities between users and developers and to make sure that everybody is speaking the same language. Analysis includes gathering requirements from users and other stakeholders and developing the functional specification, which is a blueprint of the functionality that the software should have. In some cases the analysis stage is driven by the customers as they

know more about the problem situation. Generally developers feel uncomfortable with analysis when they do not speak the business language. In some projects these responsibilities are assigned to systems or business analysts and coding is done by programmers. However, some developers stated that it was useful for them to do analysis and to "be in the users' shoes". That would give them a more global vision of the business. Being only on the technical side will limit their ability to develop the final product.

- **Design**: is preparing a prescriptive statement about a software application. I found that there are two perspectives about design. The first one sees design as an activity that needs to be done before coding the software. Developers following this perspective would advocate a particular design methodology or the use of a modelling language. According to this perspective the outcome of the design stage is a detailed document specifying how programmers should code the software. Programmers will then use these specifications to programme. A problem with this perspective is that the documentation usually gets outdated as requirements from users change. The second perspective sees design as part of the coding process (see data extract 6.5). Developers who follow this perspective consider the source code as the detailed design of the software. Developers who advocate this approach would start programming the software with just a few functional specifications. They will make design decisions during the coding and then seek confirmation from analysts and users. One of the most popular ways of doing this is with prototypes. Prototypes are good to show the users how the system will look and behave. However the danger is that some of the users would think that the prototype is the final product and will therefore expect their software will be ready before the developers think it will.

Extract 6.5

Thread title: Coding vs. design	Date: 30th November 2002	Source: JoS

"I am one of those rogues that doesn't really believe in "software design" as a separate activity, or as a document, or whatever. I believe the design of a piece of software is implicit from the very first line of code. Coding can't go without design, and design can't go without coding. Successful designers are those who can code, and successful coders are those who can design. The phrase "coding vs. design" is a contradiction in terms, at least in my philosophy."

Beka Pantone

- **Coding**: is the writing of a software programme. The transformation of the user's requirements and functional specification into a tangible solution. This job is preferred to the other stages of software development by most developers and unfortunately it is the centre of attention of most development projects. Unfortunately because (I think that) the other stages of software development deserve equal attention. Many developers start coding without having any written specification by users. These developers believe that the software will take a shape as they progress with the programming and ask users to test it. Other developers prefer to be given detailed design specifications so they do not make mistakes. The original poster to whom Joe in Extract 6.4 replies seems to be one of these. He did something wrong because his manager did not give him all the information he needed. However his manager thought he had more initiative and would find all the information on his own.
- **Testing**: is the process by which developers verify the correctness, completeness and quality of software. Unit testing and usability tests are the methods most mentioned by developers. Unit testing is focused on the technical aspect of software. It testes the correctness of algorithms and results thrown by the applications. In usability tests users are asked to use the application and testers assess the users' ability to learn how to use the graphic user interface. Testing is not concerned with verifying the correctness of the requirements gathering and functional

specifications. Problems in these areas are expected to be found in previous stages or by the users when it is too late.

In addition to these five stages I found another aspect of software development which is treated by some as process that runs in parallel with the previous stages:

- **Documentation** describes the use, analysis, design and coding of software. It is an agreement, between customer and developers, that confirms that they all share the same understanding (see extract 6.6). In traditional methodologies documentation is frequently done in a written format. However, in some modern development methodologies (like the Agile) documentation is replaced by other practices. Oral communication is preferred over written documentation between developers and between developers and users to speed up development. Unit tests written by developers and functional tests written by customers replace documentation in the testing phase. Generally, all kinds of documentations are needed. Oral documentation is needed in the development stage to improve communication and coordination among users and the development team. Written documentation is needed at maintenance stage, especially if the developer who is doing the changes did not participate in the original development team. Finally unit and functional testing are important for the technical testing at the last stages of development to assess the quality of the product. Although most developers agree that documentation is essential in all stages of development, very few of them are happy with writing it.

Extract 6.6

Thread title: What design methodology(TM) do you use?	**Date: 27th February 2004**	**Source: JoS**
"Your documentation is simply a memo of understanding between developer, analyst and customer/sales manager. Just put whatever in it so that all 3 can say "Yes, this is what we are doing". Is it enough to make it clear to the client what you are doing? Is it enough for a developer to get on with it?" Patrick FitzGerald		

C1-6: Characteristics of Software Development: aspects of the process of software development which address some of the characteristics of the model organisations as explained in chapter 1.

- **Controllability of software development.** There are two aspects of control explored in this category which are interconnected. The first one is how developers perceive control is exercised in target organisations and the second one is the extent to which developers can control the development process. Data from this study shows that there are two opposite views regarding developers' views of control in organisations. On the one hand, some developers believe that excessive or unnecessary control of information hinders software development. Extract 6.7 shows an example. An online participant explains how including more control over a website in the form of authentication makes his job more complicated. The way I understand this is that when developers design software they build a mental picture of the business processes the software is going to serve. This picture is often different from the one they are asked to model by users who like to control information. Developers think that control of information is not part of the real process but only a device to gain power and gain. As a consequence unnecessary complexities are forced into the software if the wrong picture of business processes is taken into account.

On the other hand other developers think that a system that controls information (excessively or not) offers simpler grounds to develop software as work in organising and formalising procedures has been done. According to them, clearer and formal procedures are easier to transform into software. The level of control the users have over their processes affects the control developers have over theirs. According to the first perspective of control, the level of control that developers have over the development process is

indirectly dependent on the level of control the users have over the information. The more control the users have the less control developers will have on their jobs. However, the situation is different from the other perspective. As developers will feel they are more in control of the process if the user has organised and formalised (and controlled) the relevant business processes.

Extract 6.7

Thread title: Are people complex?	Date: 12th January 2005	Source: JoS

"I would contend that a business is a totalitarian social structure where information is controlled for power and gain. It is therefor this need to control information that makes the design of software more complicated as the number of interaction paths increases.

For example, the www & Internet is used for very complex social interaction but due to the lack of information control it is relatively simple. If I had to get authorisation to view a page from my manager, obtain a view reciept, store my history for billing purposes, fill in a timesheet of activity for cross reference etc it would get very complicated. And that assumes all countries had the same rules. As soon as an individual web site requires authentication is complexity rises. Adding pages doesn't make it more complicated in itself."

Kim

- **Standardability of software development**: in this category I gathered developers' opinions about the extent to which processes and practices in organisations and in software development could be standardized. Standards in organisations are procedures or rules about how to perform business processes. Standards in development are agreed rules on how to model, document or code software. I found two opposite views in this regard. One view states that standards (in both, organisations and development) help modelling and maintainability of software. Additionally, as standards help to create a common language (business and technical language) they also facilitate communication especially within big groups. Therefore it is more likely that developers and users will be able to understand each other. The other view states that standards stifle creativity. Solutions worked out with strict standards would probably restrict users' freedom. Also, strict programming standards would not allow developers to design

improved algorithms that could perform better than the standardised ones. In the case of software development for complex organisations balance is needed between standards and creativity. Creativity to allow free interactions of users and standards to allow developers to model complex processes and code software for them. Extract 6.8 shows an example of standardisation of processes helping developers to design an information system. The original text is in Spanish (left) and I have translated it into English (right). Erich explains how a situation which was perceived as complex by developers was simplified by standardising (or formalising) the processes.

Extract 6.8

Online interview with Erich Pardo	**Date: 26th October 2004**	**Source: MSN Messenger**
Cecilia: como afrontarías un problema d desarrollo de software para una situación recontra-super-archi complicada Cecilia: imaginate q trabajas para varias empresas a la vez Cecilia: y q tienes q ver procesos de decisión, workflow y administración del conocimiento Cecilia: dime lo 1ero q c t ocurra Erich: algo así me paso Erich: le echaban la culpa de que el sistema no arrojaba info a tiempo Cecilia: aja Erich: así que lo primero que se me ocurrió es realizar una reunión con todas las personas cabezas involucradas en el proceso Erich: y de ahí salio que el proceso actual era una caca Erich: recién se enteraron de los problemas Erich: entonces se decidió cambiar el proceso y además los sistemas Erich: simplificando algunas etapas del proceso Erich: tratando de hacerlas lo mas automáticas Erich: además ya no se daban tantas excepciones Erich: como anteriormente se daban Erich: luego se le pidió documentar el proceso y con actas se comunico a todo mundo involucrado y se formalizo el asunto		Cecilia: how would you face a software development problem with a super complicated situation Cecilia: imagine that your user is many organisations Cecilia: and that you have to see processes like decision making, workflow and knowledge management Cecilia: tell me what comes first to you mind Erich: something like that happened to me Erich: they blamed the system because it did not provide the information in time Cecilia: aha Erich: so the first thing that occurred to me was to organise a meeting with all the heads involved in the process Erich: from that we concluded that the process was [colloquialism that expresses the intensity of disappointment] Erich: they just knew about the problems Erich: then it was decided to change the process and the systems (meaning software) Erich: by simplifying some of the stages of the process Erich: trying to make them more automatic Erich: so there were no many exceptions Erich: than before

Cecilia: entonces me dices, q en parte la solución fue simplificar los procesos de negocio para q el software sea mas sencillo, además de estandarizar procesos para q todos los involucrados hagan "lo mismo" Erich: no tanto simplificar el proceso sino formalizar el proceso es como un mix claro que también se mejoro	Erich: then we documented the process and with minutes we communicated everyone involved and the process was formalised. Cecilia: then you say that part of the solution was to simplify the business processes so the software becomes simpler, additionally you have to standardise the processes so everyone does the "same" Erich: not only simplify the process but formalise it, this combination improved it.

- **Incrementability**: incremental development is a strategy in which software is developed and delivered in pieces according to priorities set by the customers. Most agile methodologies advocate incremental strategies and aim at keeping the customer happy. With an incremental approach, developers are able to aim at small achievable targets and to show to the customers that some progress is being made. While the customers receive and use the first release of a software application, developers continue to work on the functionalities that were left. At the same time, developers are able to receive and process feedback from customers and to do pertinent modifications in the code when they release the next increments. Incremental approaches are strategies designed to cope with flexible and adaptable business settings. By using an incremental strategy developers would be able to respond quickly and accurately and modify software according to the new conditions in the organisation (see this idea reflected in extract 6.9). A problem for developers following incremental development is perhaps the inability of seeing the big picture. Being focused only on little achievable short-term tasks can hide the long term objectives.

Extract 6.9

Thread title: What design methodology ™ do you use?	Date: 07th March 2004	Source: JoS

"There are two types of actions, planned and situated actions. Planned actions involve conscious thought ahead of time, and these are easily captured in requirements meetings. Unfortunately, most of the work an "expert" does is situated: it is in response to some situation, where the exact conditions present will determine the response. These are the types of requirements that typically lead to the later "You didn't tell us about that!!!" "Any idiot knows that/Well you didn't ask!!!" exchanges. The iterative/ incremental helps to address these by allowing for a planned evolution (it wasn't a missing requirement, we just hadn't gotten there yet). As the user works with the (partially completed) system, they will get to experience the situations and feel that something is not quite right; this will lead them to bring their unconsious knowledge to the front of their mind, where they can communicate it as written requirements the programmer can add to the code (or expert system engine, or whatever)."
David Lathrop

- **Adaptability**: I defined this category as the extent to which the software development team can adapt to changes in its target organisation. Data in this category reflects the practices developers undertake to follow changes in users' requirements. Three important points are worth to mention here. The first one is that for most developers constant contact with clients or users during the development period will help them to evolve the process according to how target organisations change. This is only the opinion of developers, as in practice, they state that it is very difficult to do this, especially in projects for big organisations where many perspectives should be considered. This view contradicts other accounts in which developers express their discomfort at working with other people and talking non-technical language (these aspects are explored in categories C4-5: [Developers/Mentality] and [Developers/Working Habits]) The second point is that very often developers do not accept changes in requirements as they think these are a consequence of bad planning or users not knowing what they need. Some developers think that what is needed is a change in attitude by developers. If they thought of software as a service and instead of only a product, adaptation would be easier to understand. A service is continuous in time and is not inflexible in the sense that it is not as tangible as software and therefore can be changed (or adapted)

easier. Another aspect of this point is that usually people who have experienced change before are more receptive to new change. This applies to both developers and users (see extract 6.10). The last point is that incremental and iterative strategies as opposed to the waterfall cycle, traditional approaches, are more appropriate to deal with the adaptation of complex organisations. As mentioned in the incrementability category section, by delivering achievable pieces of software, developers can work at a reasonable pace and users get to know the software and are able to provide feedback to developers sooner.

Extract 6.10

Online Conversation with Jackeline Saavedra	**Date: 07th October 2004**	**Source: MSN Messenger**
Jackeline: me he dado cuenta del cambio actual de los usuarios Jackeline: por ejemplo Jackeline: antes, cuando iniciamos toda la revolución acá, intentando colocar cosas en web por ejemplo Cecilia: aja Jackeline: la gente era mas reacia al cambio, ahora nos están llevando la delantera, piden y piden y tienen visión Jackeline: ya ves a tías hablando de mp3 Jackeline: pero antes no era así Jackeline: no se si me dejo entender Jackeline: la tecnología ha ido entrando y las personas nos hemos adaptado pero sutilmente Jackeline: en varios lugares te dicen queno pueden hacer esto o lo otro porque "el sistema no lo soporta" Cecilia: aja, crees q después de un cambio grande la gente ya esta mas receptiva a cambios subsiguientes Jackeline: si, al menos eso me parece		Jackeline: I have realised the change in users now Jackeline: for instance Jackeline: before, when we initiated the transformation here, trying to put new things on the web for example Cecilia: aha Jackeline: people were more reluctant to change, now they are taking the lead, they ask for things and have vision Jackeline: you see old ladies talking about mp3 Jackeline: but it wasn't like this before Jackeline: Do you get me? Jackeline: technology has been introduced in our lives and people have been adapting subtlety Jackeline: in many places they tell you that they cannot do this or that because the "system does not support it" Cecilia: aha, do you think that after a big change people are more receptive to subsequent changes? Jackeline: yes, I think so

- **Predictability**: is the extent to which developers and their managers can predict time, budget and outcome of a software project. In general, it is very difficult for developers to predict these variables; the existing techniques do not seem to be accurate. As AllanL5 in extract 6.11 suggests, the nature of software

development is different from what these existing techniques assume. In software development every project is a different venture. Clients, users, developers, tools, platforms, code vary from application to application and the outcome of such combination is almost impossible to predict. Most software developers are not comfortable with a situation like this. According to them the reasons for this unpredictability are more social than technical. Developers like computers and software because for them these are more predictable than people. However, when dealing with people developers struggle to understand their particularities, ambiguities and indecision. A solution that some developers adopt is to not trust formal methods to predict projects but to use their common sense. Common sense is developed by their experience in previous projects. Developers will take shortcuts whenever an unexpected situation arises. Shortcuts are anything that they think is appropriate for any new circumstance they are trapped in. Although common sense is not 100% accurate, developers feel it is safer and more in control of their development activities.

Extract 6.11

Thread title: Why take shortcuts in Software development	Date: 28th July 2004	Source: JoS
"The problem is that currently creating software is more a craft than an engineering discipline. It is conceivable that software will NEVER have predictable schedule, it may just be the nature of the beast. Every project is different, the tools keep evolving, even ideas of what makes a good design (OOD? DFD? Lock down requirements? Let them float?) keep changing." AllanL5		

- **Levels of abstraction**: when designing and coding software, developers can organise information in hierarchies of knowledge. Higher levels in the hierarchies are the more abstract models of the information system, showing the functionality that the software will have. The lower level of abstraction is the actual code of the software. For some developers, modelling and designing at higher levels of abstraction may increase the long term productivity (see

extract 6.12) and reduce the complexity of software and will provide simpler models of the organisations. For example, some complexities could be hidden in black boxes. Once coding is being developed then these black boxes are exposed and their complexities are modelled one by one. Developers go down through the levels of abstraction as they add more detail to their models. This process ends with the programming specifications and finally the code itself. For other developers there is no value in abstract design as it just takes time out of their coding. Design would be more realistic when it is done at coding time. The developer will then face the real constraints and know what is achievable and what not. These kinds of developers state that coding is design and design is coding. There is no real difference between them both (see extract 6.5).

Extract 6.12

Thread title: Innovation, productivity and abstraction	Date: 23th June 2004	Source: JoS

"Abstraction provides greater long-term productivity at a cost of short-term productivity. It doesn't affect functionality/innovation, because functional requirements come first."

Nearly Nameless

- **Complexity of software development**: this category comprises data about the complexity of the process of software development itself and the complexity of the organisation or system which needs the software. By complexity I mean how social factors in both environments affect developers work. There are two aspects of complexity that I found reflected in the online data. The first one is how free social interactions between users affect the modelling and design of software. There are two opposite perspectives in this regard. One perspective sees free social interactions among users as making software design more complex. "Free social interactions" means that people will not follow standard procedures and will use their common sense. Having people acting differently in similar situations makes the modelling of their activities almost

impossible. The organisations' complexity will affect the complexity of software development. Developers have to come up with new ways of developing software that adapts to the idiosyncrasies of people. So far methodologies and technology need standard and formal procedures resembling mathematical formulas. For example, if for input A in a business process the outcome is always B ansd the calculation is known, then it is easy to reflect it in the software. However in a different process where the outcome (or calculation) for input A is not known or could vary a lot, methodologies and software are helpless. On the other hand, the other perspective sees free social interactions among users as making software development easier. Not all the social aspects in business processes need to be automated. In extract 6.13 a developer states that only the core functionalities of an application must be considered. Most users perform their complex tasks outside of software; therefore there is no need for developers to model these aspects and hence software becomes simpler. The second aspect of complexity is related to the first one (overt free social interactions). In situations where users are allowed to act freely and use their common sense, there is scope for hidden agendas and covert channels of information. Users would communicate what they are *supposed* to do but would not tell what they *actually* do at work. These practices would be unknown by developers and will not be considered in their designs. As a consequence the software will be incomplete and will not serve the "real" needs of users. In developers' eyes this is a complex system to model because of the difficulties at gathering information and understanding it, as well as putting more work at talking and watching users work.

Extract 6.13

Online conversation with Jackeline Saavedra	Date: 27th October 2004	Source: MSN Messenger
Jackeline: creo que te ayuda a acercarte a la realidad, me refiero al modelo, pero hay detalles que no se pueden modelar, solo se asumiría como caso excepcional o especial algo general Cecilia: ¿y cómo hace la gente q usa el sistema en esas situaciones? Jackeline: en el modelo se implementa un caso especial, esos casos especiales podrían entrar sin reglas o algo así, pero deja huella y sangre Cecilia: ? Jackeline: me refiero a que se podría contemplar casos excepcionales, pero sin entrar al detalle Cecilia: a la hora de modelar Jackeline: lo digo porque no hay forma de contemplar todas las excepciones Jackeline: el de se murió mi gato, o me quede botado Cecilia: aja Jackeline: a menos que el sistema sea de ese tipo y sea básico saber si la persona tiene gato y si esta vivo Cecilia: jejejeje.. ya seria control extremo	Jackeline: I think that it helps you get closer to reality, I mean the model, but there are things that cannot be modelled, we would assume it as an exceptional or especial thing, the model is something general Cecilia: and how do people that use the system in these situations do? Jackeline: we implement a special case in the model, these special cases could not have special rules, but would leave their track Cecilia: ? Jackeline: I mean that we could consider exceptional cases, but not in detail Cecilia: while modelling? Jackeline: I say this because there is no way we can consider all exceptions. Jackeline: something like my cat died or my car is not working Cecilia: aha Jackeline: unless the system is this type and it is imperative to know if the user has a cat and it is alive Cecilia: hehehehe.. it will be extreme control	

- **Productivity in software development:** is the efficiency with which software is produced. Factors most considered are the speed at design and programming, the number of bugs or errors in the software, and quality of software in general. Productivity is a very important matter for developers. Most of them are assessed by their productivity and their jobs depend on their performance (some are directly paid in those terms). Online members discuss about how to improve their productivity. From the online data it seems that the following 5 factors affect productivity positively:
  - Selecting the appropriate platform and tools, for the kind of application they are going to develop (see first text segment in extract 6.14).
  - Planning and designing what is going to be coded beforehand (not all developers would agree with this)

- Using abstract models to understand the system. This would yield productivity in the long term at the cost of short term productivity. (See extract 6.12.)
- Use of development standards
- Pair programming, a strategy emphasised by agile methodologies in which programmers work in pairs.

However I also found other aspects of software development that developers perceive as hindering their jobs:

- The unpredictable nature of software development. Developers do not know exactly how a project will behave and therefore do not have the chance to think on improvements in the long term.
- Having projects with no clear or explicit goals. As many individual goals exist within working groups it is more productive for developers to have their priorities well stated.
- Doing overtime work and having not rested developers.
- Planning and designing what is going to be coded beforehand as it takes time from their coding (not all developers would agree with this)

Finally, the human and social side of software development is also seen as influencing developers' jobs. The fact that it is performed by humans makes it less productive than managers would like. Another factor that derives from this one is that the machine metaphor that portrays developers as commodities is not uncommon in the development field. I saw that most developers although resenting this treatment from management also portray themselves as programming machines which are assessed by their productivity.

Extract 6.14

Thread title: Fundamental Laws of Software D. Part 1	Date: 10th December 2002	Source: JoS
"I know if I have to wait long time to get my code on a test platform; I sometimes lose the flow and have to spend time trying get back again." David Hickerson		
Thread title: Software developers as commodities	**Date: 31st December 2004**	**Source JoS**
"75% of the population now works in the service sector, where productivity is pathetic simply because it's mostly done by human beings. Thus, management looks for ways to either lower labor cost (outsourcing to the 3rd world) or improve productivity (computers, etc.), or both, but there's only so much that can be done." Fred		

6.2 C2: Software Engineering (SE)

Software engineering is relevant to the concern of complex organisations because it influences the way developers see organisations and users. I found that the great majority of developers advocate fully or partially SE's methods. The reason for this is probably that most of them come from a computer science background where software engineering and similar approaches are taught. Developers are also comfortable with computers and understand people and organisations via machine metaphors. Software development methodologies based on software engineering provide scientifically and mathematically proven and hence assumed valid practices. This means that for some developers engineering practices can guarantee the development of usable and reliable computer applications. However, this is not always true. As developers also acknowledge, these methods presuppose well defined and stable business settings. Human organisations are not like that. Trying to force hard methods into situations defined by human and social interactions (as in this study context: complex organisations involving people adapting and interacting freely) might not work. Software worked out like this would not solve the users' problems and would probably be rejected by them.

My data show that there is an opposing perspective to software engineering. This perspective (advocated by fewer developers) sees

software development as an art. As software artists, developers should follow their desires, preferences and tastes to design the best quality software. Contrary to software engineering, this "artistic" view puts emphasis on the human side of software development by acknowledging the unpredictable character of software users and puts people (developers) to understand people (users) as opposite to putting hard engineering methods to tackle soft issues with users. Extract 6.15 shows comments from developers who advocate each perspective.

Extract 6.15

Thread title: Objection to XP (extreme programming)	**Date: 18th July 2004**	**Source: JoS**
"I'm bothered by the statement that software is "creative". Not that I don't think creativity is necessary, but in the implicit assumption that it is more like the arts than like engineering. To me this indicates a misunderstanding of both art and engineering." Jeff Kotula		
Thread title: Monumental Incompetency- Part TWO	**Date: 8th June 2004**	**Source: JoS**
"Wishing (or demanding) that Software Engineering be a more disciplined process misses this key fact. It is always going to be a process of taking a customer desire, finding out what is implementable, implementing some of it, then educating the customer as to what is possible and desirable, so he can change his expression of what he wants." AllanL5		

6.3 C3: Software

This category was taken into account to look at my views of developers' perspectives on software itself and to compare and explore the similarities between complex organisations and the software their employees use. Data show that developers are able to grasp some human and social aspects from their target organisations which they try to cast in the software they develop. Two subcategories emerged from the data. The first subcategory is about the kinds of software. As most developers must have been involved with the development of different kinds of software it becomes important to emphasise the differences and to state and explain my focus on business software. In the second subcategory the characteristics of business software that are connected

to the characteristics of the model of complex organisations are mentioned and explained.

C3-1: Kinds of Software: there are different types of software depending on their purpose and technical platform they are developed for. Software developers that I met online develop different kinds of software. Depending on the platform, audience, and use of the software their approaches to development differ, especially in the technical side. Games and embedded software (e.g. software for mobile phones) require deep knowledge of the hardware in which the software is going to be run and aim at effective use of resources like memory. Off-the-shelf software like word processors are developed to provide generic features that would appeal to the general public. As the software is going to be bought by different kinds of people it cannot be customised for any particular taste. In fact, it will encompass various elements to suit various tastes and hence will be flexible. Market researches are carried out to find the characteristics of a standard customer which will represent the whole target market. Technically this kind of software is complicated as it is released for many platforms (i.e. kinds of computers and operating systems). The job of developers is to make the software behave consistently in each of the versions. By contrast internal software is customised for a particular kind of situation and hardware. Technically it is simpler to develop as it does not need to be proved in different platforms. However in the case of business applications, it is the social aspects of the organisations that are usually found difficult to understand and transform into suitable models to code. This topic was been also discussed in section 3.1 of chapter 3 though covering a wider spectrum of types of software.

Very few developers define the software that they code as Information System or an application for an Information System. More common terms are business applications or business software. Most developers think of users and management as their target customers. There is little acknowledgement of the term Information System as a sub structure

within organisations that organises the provision of information. In general Information Systems and Software are thought of as tools to automate repeatable processes. The next extract shows one definition of Information System found in a forum which describes IS as a synonymous with Software.

Extract 6.16

Thread title: Lack of Adaptability	**Date: 10th June 2004**	**Source: JoS**
"Informations systems increase the efficiency of the unchanging repeated tasks of an organization thus allowing more time for change and absorption of variety."		
name withheld out of cowardice		

C3-2: Characteristics of Software: Data from this category shows 9 characteristics of software. Characteristics of software were taken into account to explore how software exhibit or reflect the characteristics of the information systems and organisations they serve. In line with this thesis focus the following discussion concentrates only on business applications.

- **Reusability**. It is one of the cornerstones of modern programming. Reusability is the extent to which developers can use a piece of code more than one time. In the search for efficiency at programming, developers try to save time by writing code that can be used in other programmes. The main assumption here is that there are functionalities within an organisation which are needed by different areas or departments at different times to perform different business processes. Hence code representing that functionality could be cloned in different programmes. Reusability requires extra effort at the beginning but pays off in the long term when developers benefit from code that has already been written and tested. In extract 6.17 an online participant states that object oriented approaches emphasise reusability of code and that this also improves productivity.

Extract 6.17

Thread title: The API is King	Date: 28th August 2004	Source: JoS

"The good thing about object oriented programming is that you only have to do this once. That is to say, once you have obtained an elegant solution to a particular problem and you have the right abstractions to work with you will not have to do it again. The developers who use your code will have improved productivity as a result of the object orientation that you have added."

Emerson Clarke

- **Complexity** is understood in terms of how complex the structure of the target organisation and business process are and how complex (in technical terms) the software needs to be to fit them. It involves knowing what entities are involved and their relationships at the functional and technical level. Complexity of software does not necessarily correlate to the complexity of the organisation. For example, adding more reusability to software adds more complexity to it regardless of the kind of organisation it is intended for. Additionally, the kind of platform and programming language can also add more complexity. Most developers agree that the best solution is usually the simplest. More complexity is accepted only if it pays off, that is if there is an economic benefit from it. For example, in the case of off-the-shelf products a more complex solution can be acceptable if it is better than the simpler ones in terms of efficiency and appeal to the general market. In general, developers think that their job is abstracting complexity from users and developing simple solutions that can be understood by their users. In extract 6.18, two developers give their opinions about software complexity. The first one says that simplicity in software is directly related to quality. The second states that the nature of software is complex anyway and that platform and programming language are part of its complexity.

Extract 6.18

Thread title: Fundamental laws of Software Development part 2	Date: 6th December 2002	Source: JoS

"Adding complexity without adding to product quality causes a loss in quality. Complexity is difficult to produce and maintain. It is imperative that simplest solutions are found to a quality to product."

David Hickerson

Thread title: Objection to XP (extreme programming)	Date: 7th July 2004	Source: JoS

"But yes, there has never been a product made by man which really shares all the attributes of software. Infinitely modifiable, it supports an incredible amount of complexity. We're still deciding what language to use to create the beast -- and each language, language construct, and application domain changes dramatically what the final product looks like."

AllanL5

- **Controllability**: is the extent to which developers and users can control software. For developers it is important to be in control of all the (formal or informal) methodologies and tools they use to design software. Only in this way can they be sure that their software will work. Additionally, one of the obvious requirements from users is that the software can be controlled by them. That is that the software is easy to learn and to use, that is reliable and responds to all users needs. Developers' experiences have shown them that software that is uncontrollable is rejected by users. According to developers, the lack of standards in the industry and multiple perspectives about it causes confusion about how to create controllable software.

- **Reliability** is defined as the probability that the software will not cause the failure of a system for a specified time under specified conditions. Reliability has two aspects. Reliable software is one which possesses the appropriate functionality for a particular organisation or process. Reliable software is also one which is working fine in the technical sense (i.e. it works but does not necessarily have the desired outcome). Developers value more reliability of software than the creativity encapsulated in it. Software that works but is simple is worth more to them than software that possesses more inventive and creative features but which fails constantly.

- **Flexibility** is the extent to which software is capable of change. Flexibility is inbuilt in the code within the software. Flexibility depends on the vision the programmer has when he codes. A good programmer will think about future changes that will probably be done by a different person. For example, flexible code can be code that has every function clearly separated and explained. When a change needs to be done it should be straightforward, few functions should be involved and the programmer should find it easy to know where to do the changes. Another meaning of flexibility in software is its level of configurability (software's behaviour can be changed by changing the value of variables). As with reusability, flexibility needs to be planned from the beginning. Flexible code can be changed easily and this helps developers to follow when users change their minds or when business rules are modified (see extract 6.20). However there is a limit to flexibility as constant changes in code can make it inefficient (code becomes big and slow). Leftovers from previous versions of the software can make source code heavy to maintain. By definition flexible software is needed by Complex Organisations (as noted in 3.1 – A classification of software systems relevant to the model of complex organisations). The unpredictability of decision makers and the great variety of working styles within organisation(s) need software capable of being changed every time the needs of their users change.

Extract 6.20

Thread title: Are people complex?	**Date: 12th January 2005**	**Source: JoS**
"I do have to take into account individuals as far as usability goes but all I can do is try to design and code for flexibility as they laws and interpretations will always change and I will be going back into the code to inforce this new constraint on the users of the application." Douglas		

One example of flexible software is open source software. Open source software is any computer programme whose source code is

available for modification by other developers. Open source code is modified many times during its time span. As the source code is available, modifications could be done in series by the creator or any other developer. Despite the number of changes in its code, open source software is very well valued in the market because it allows easy customisation. Detractors of open source argue that it is less reliable than closed source as there is no formal entity who can guarantee its quality. By buying open source one cannot be sure if appropriate technical support will be provided by the creators of the code.

- **Adaptability** of software, as developers see it, relates to the extent to which software can be adjusted to fit different situations and to the extent users can adjust to the software. Adaptability, as opposed to flexibility, is not an inherent characteristic of the code but depends on external forces that make the system change. It does not necessarily involve a change in the code of the software but a change in the way people use it. Thus adaptability is a characteristic that emerges from software use. From the data it seems that developers believe that the level of adaptability depends on the level of dependence of the organisation on the software. When changes are needed it is more likely that software that is used in the core of the organisation's processes will be adapted. This means that users will adapt to it or that they will adapt the software to their needs. Contrary if an application does not belong to the core processes of the organisation it will probably be rejected by the users. This is part of the evolution of software. As users learn how to use software and how to assess it in terms of their needs software is adapted to fit the needs of the organisation. However, this is not an infinite loop, as according to developers software should not last forever. Developers and especially users should recognise that software has a relatively short lifecycle and that it will need maintenance when conditions in the organisations and technology change. The end of this process

is when software reaches a state of incompetence (it stops being useful). Also, technology changes and the increased competition force organisations to look for better solutions than to adapt their current software.

Extract 6.21 shows how users of an airline management system adapted to changes in the regulations. By performing some tasks manually they continued to provide the service to the public. As this application was part of the core of the organisation (selling tickets and checking in passengers) users had no other alternative but to subvert the system to make it work as they wanted. However, on the other hand, this extract also shows how problems of communication between developers and users can cause problems of coordination like this. This experience reflects Coakes view about the role of technology in organisations: "We see people making technology work the way they want it to work, deliberately or unconsciously. Right from the first computer system that was ever put into place, people changed it if they were unhappy with the way in which it worked. Either they didn't use it at all, or they used it for a different purpose." (Coakes et al., 2000, 9)

Extract 6.21

Thread title: Flexibilidad y Adaptabilidad (Flexibility and adaptability)	**Date: 10th November 2004**	**Source: Microsoft Discussion Groups (in Spanish)**
"Lo ideal es que el software evolucione paralelamente con la empresa. Pero es difícil que esto ocurra. Por una parte, y hay que reconocerlo, hay mucho software mal diseñado (casi me atrevería a decir que TODO el software acaba alcanzando su "nivel de incompetencia" tarde o temprano). Pero también hay mucha descoordinación interna. Durante varios años trabajé para una pequeña línea aérea en el desarrollo de su sistema de gestión. En ocasiones, la gente de mercadeo decretaba un cambio en el método de cálculo de las tarifas que no era compatible con el que teníamos implementado en los programas. Normalmente, yo me	"The ideal situation is that software evolved in parallel with the organisation. But this is difficult. On the one hand, we have to recognise, that a lot of software is bad designed (I would say that EVERY software ends reaching a "level of incompetence" sooner or later). But there are also problems with internal coordination. For many years I worked for a small airline on the development of their management system. Sometimes, people from marketing ordered a change in the method for fees calculation that was not compatible with the one we had implemented in the programmes. Normally, I would find out when I had	

enteraba cuando tenía que viajar: llegaba al aeropuerto y me encontraba a los operadores sacando las tarifas con calculadoras. Una vez, incorporaron un nuevo avión que tenía asientos de primera clase y asientos de clase turística, pero la línea estaba basada en el concepto de vender "clase ejecutiva a precio de económica". Ello implica una clase única. Pero el avión que hay viene como viene, y a ver como te las arreglas. El dichoso avión estuvo dos meses en un mantenimiento mayor, llegó al país, estuvo un mes más en trámites de inspección y matriculación. Y sólo cuando descubrí a la gente chequeando a mano en el aeropuerto fue cuando me enteré de que ahora teníamos varias clases: caso completo durante quince días ¿por culpa del software?" Leonardo mvp vb	to travel: I arrived at the airport and found out that the operators where calculating the fees with calculators. One time, they bought an airplane that had first class seats and tourist class seats, but the airline was working under the concept of "selling executive class at the price of the economy one". That implies only one class. But the airplane comes as it comes, and you see how you sort it out. The airplane was two months under heavy maintenance, came to the country, and spend one month more in inspection and registration. And only when I saw people checking in manually in the airport I realised that we had many classes. That was 15 days later. Was it the software's fault?" Leonardo Mvp vp

- **Usability** relates to the extent to which the software's graphic user interface (GUI) is easy to learn and easy to use (i.e., user friendly). It is seen as a mantra by some developers (a definition of mantra is given in the glossary). Software with poor GUI is useless for users and is therefore rejected. From forums' data (see extract 6.22) I see that usability is the only approach which developers use to address social issues in software. For most developers software development is a technical job, first in the list comes the technical requirements of hardware and software along with the requirements of functionality. Then usability comes in third place as another technical job. Developments in the usability of software have been made in the area of human-to-computer interaction. That is, the way one individual operates a computer and its software. However, interactions between individuals through computers have not been addressed. Even in those kinds of applications like groupware or social software, individual usability is considered over group usability. From the model of complex organisations we see that in addition to providing usable software to individual users it is also important to provide them with

interfaces through which they can interact with other people. Therefore software should also facilitate social (or business) interactions between users, that is, software should provide a social interface.

Extract 6.22

Thread title: Are people complex?	Date: 12th January 2005	Source: JoS

"The individual person's social impact on what I am asked to produce is relatively minimal. I have to be concerned with the individuals as far as usability is concerned but not on how they intend to use any particular screen. Part of this I believe is the heavy monitoring and regulation of the industry as a whole. Any time I am developing new functionality or creating a brand new system I have to be primarily concerned with the business rules defined for the process. So the managers and directors involved develop and document how and why any particular portion of the application will be used so that it will function according to their documented business plan and stay inside compliance issues."

Douglas

- **Maintainability** is a concept related to flexibility and adaptability. Maintenance involves the update, correction and bug fixing of software. It takes place throughout most of the life span of software. The more flexible the software is and the more adaptable their users are the easier maintenance becomes. However, as many developers argue (see extract 6.23), there should be a good business reason to invest in maintenance. Maintenance does not offer a good career progression path for developers. If the software is good and works they can leave it as it is. If their users have adapted a new (manual) system and are happy with it, the software can remain as it is. Maintenance is not the preferred activity for developers. It usually involves the revision of other people's code, the search for missing or outdated documentation and puts the developer under pressure due to time and budget constraints. (See Appendix 1 for an account of a maintenance job by a software developer.) The problem grows because of the lack of standards in the industry. For some developers, standards for documentation and coding would smooth the maintenance process

as it would be easier for them to understand new code and to find the source of errors.

Extract 6.23

Thread title: Bridging the gap b/t high and low level design	Date: 15th June 2004	Source: JoS
"Do you really need to rewrite it all? Can you refactor piece by piece instead of ripping up everything at once? I always find it painful to look back on code I wrote a few years back and think "Wow, I was such an idiot, I can't believe I did it *that* way"... but from a business perspective, if the thing works, I think you should really ask whether a rewrite is warranted. Will the benefits you gain in future extensibility, maintainability, etc. really make the effort worthwhile?" John C		

- **Social interface** emerged as a characteristic of software from the discussion of usability of software and is also connected to the emergence of social software. Data from forums showed that social aspects are considered in software development as far as the user as an individual is concerned. Social interface is a new topic in the field of software development. It concerns the type and structure of social interactions that the software enables. As much of software is being developed, for instance to enable people to communicate through the internet, new issues of group interaction through time and distance arise. A clear link to the model of complex organisations can be drawn as it concerns groups, multi-groups and multi-organisations using software. Although the topic of social interface is starting to be recognised as an important issue in software development, no actual development in the practices of usability (or any other related area) is being done. Extract 6.24 shows one of the few definitions of social interface design I found on the web. The writer concentrates on the difference between user as an individual and user as a group to define it.

Extract: 6.24

Thread Title: What is the "social interface"?	Date: 3rd September 2004	Source: JoS

"User interface design = designing software taking into account the users who will be using it.

Social interface design = designing groupware taking into account the social situations it creates."

Joel Spolski

6.4 C4: Developers

This category contains data that describes developers as professionals in their working environments. Here the data allows me to show a picture of how developers see themselves. This picture is built of 8 sub categories that collectively include their backgrounds, education and working habits.

C4-1: Training: comprises workshops, seminars and short courses a developer must attend to get up to date with technology. If developers work on their own they will look for training courses themselves. They will try to get new skills whenever they find the trend in the market is changing towards new technologies or if they find something which appeals more to them. If developers are being employed by a company they would expect management to provide those courses for them. However, this is not always the case. A feeling of resentment from developers appears when companies force them to learn something on their own. In most cases management force developers to the same time constrains, as if they were only producing software, without taking into account that they are delivering working software and learning at the same time. This strategy is usually not attractive for developers as extract 6.25 shows.

Extract: 6.25

Thread Title: What is "extereme programming exactly?	Date: 24th July 2004	Source: CodeGuru

"Many companies, will merely tell an employee to learn something on their own time, with no instruction or mentoring and expect the person to get it right immedialtely. This is (usually) doomed to failure. The result is that the company stagnates."

TheCPUWizard

C4-2: Education: this category contains information about formal education at universities that the developers receive before or while they are working. As software development is relatively a new field formal education in this area has not been available until recently. I found that a big proportion of developers have been self-taught especially in the area of programming. This group of developers differentiate themselves from the rest by claiming that they can use their common sense and have natural skills for developing software. They think that college education is usually outdated. As technologies advance rapidly they prefer to update themselves on their own. This group of developers are very technical and focus only on the programming and testing parts of development and usually neglect planning and design. The other group of developers are graduates from university or college. Most of them have a background in computer sciences, very few come from information systems or other management related backgrounds. However, as a result of having taken at least one management related module they acknowledge the value of good planning and design in software. Disputes arise between self-taught developers and developers with qualifications. The topics of discrepancies are the quality of their jobs, the formality of methodologies and the importance of planning in advance. Extract 6.26 shows an example of two opposite opinions regarding education for developers.

Extract: 6.26

Thread Title: Monumental Incompetency- Part TWO	Date: 08th June 2004	Source: JoS

"this whole thread demonstrates why college educated programmers are ALWAYS inferior to self-trained programmers.

yep. "

muppet is now from madebymonkeys.net

Thread Title: I am fragile	Date: 09th June 2004	Source: JoS

"One regret I have is I never got to graduate with a degree in Computer Science. If my parents were informed enough and could afford it, I would have gotten one. I hate to be treated like a VB programmer. I hate it when they emphasise the qualifier programmer with a double quote just to show you down. That is another reason I want to do work for Microsoft and get to do some low-level stuff or may be get to be a part of a team that's developing the next best development tool."

Sathyaish Chakravarthy

C4-3: Experience: some developers think that when employers need to hire new staff they look for experience rather than for a degree. For employers, experience in a particular area is the best proof of someone's ability. As RobDog88 in the first text segment in extract 6.27 says, employers want to make sure that developers can actually do what is needed. A related point is made in the second segment of the same extract. Because developers dedicate more time to improve their technical skills, (i.e. experience with programming languages) than the time to obtain experience and knowledge of businesses and systems domain, it is the non technical skills which are more difficult to develop. Formal methods for estimating price, size of project and time do not work accurately and reliably in practice. Developers have to learn these skills themselves through experience, previous errors and commonsense. As Christopher Hawkins in extract 6.27 says, some development skills are more an art than an exact science. Developers believe that the perfect combination is to have a good degree *and* working experience. Degrees provide the knowledge foundations which can then be developed by experience.

Extract: 6.27

Thread Title: Design your degree	Date: 28th September 2004	Source: VBForums
"Companies are looking for a programmer with proven experience because they want to be able to see if you understand and can put into motion good programming skills. Then if you have a degree on top of that, you will be in front of the other applicants." RobDog888		
Thread Title: Estimating Development Time...	**Date: 22nd July 2004**	**Source: JoS**
"Learning to give a good estimate takes time and experience. If you're being asked to estimate something that you've never done before - look out! You'll at least need to take the counsel of someone who has. It's pretty scary. It took me about 6 years to learn to produce an accurate estimate. There is definitely an art to it." Christopher Hawkins		

C4-4: Responsibilities: in this category I have gathered data about what developers think about the kinds of tasks they should be carrying out as developers. Here I focused on the tasks that were in the boundaries of their responsibilities. Because the coding of the software is in the centre of their attention some other tasks are perceived as not important or not part of development. One such unimportant task is the analysis and design of the software as separate formal activities. While some developers think that they should only focus on code and the rest of the things will come up in time, another group think that they should learn, understand and speak the user's language. According to this second point of view, developers should have a wider perspective of their target organisation and should not restrict themselves to only one perspective (i.e. the user's perspective) which will limit their understanding of the big picture of the organisation. It was mentioned that one way of acquiring knowledge about the target organisation is to become involved with users' tasks. This could be taken as a form of ethnography. However, ethnography is not widely known or not acknowledged as a valid practice by developers (or at least not by the same name). Another important point to note is the trend in the market towards developers who are able to perform the complete development process as opposed to developers who are specialised in only one part of it (e.g. programming). The skills looked for include systems analysis and design. Advocates of this trend in the market see software development as an indivisible task which has to

be carried out by the same team or person. The responsibility of a developer who participates in the whole development process is over all the outcomes of the process (e.g. documentation) as well as the final product, i.e. the software. Extract 6.28 shows the opinion of a programmer who disagrees with the latter point of view.

Extract: 6.28

Thread Title: Software developers as commodities	**Date: 31st December 2004**	**Source: JoS**

"I too have seen a real and troublesome trend of employers requiring the developers to "do it all" so to speak. In many instances, none of us can be experts at everything. Given the complexity of today's computer systems, is it reasonable to expect that any one person can be expert at business analysis, interviewing users, systems design, database design, interface design, general programming in his or her chosen language (or, more likely these days, programming languages), testing, implementation, writing documentation, and training users? Of course it isn't, and yet the realities of business system development (actually just about any type of development) often require that we have to take on, if not all, at least several of these roles."

One Programmers Opinion

C4-5: Mentality: this category describes the way developers approach their jobs. Most developers think they should be valued for their technical skills and levels of productivity rather than for their working habits and their social skills. In extract 6.29 a developer praises Microsoft's hiring strategy. According to him Microsoft look only for technical skills in new developers and do not care about appearances or working habits. This is the ideal for many developers. In fact they portray themselves as having poor personal and communication skills. One of the most commented upon weaknesses of developers, in the online data, is their inability to speak non-technical language and their reluctance to talk to non-technical people. In relation to this I perceived that a significant number of developers underestimate their users because they are not experts in technology. As a consequence they do not listen to them and start projects using the developers own assumptions. A good number of developers stated that they would prefer analysts to do the talking to users. These developers differentiate themselves from business or systems analysts and want only to focus on the programming phase of development. There is however other group of developers who actually

think it is necessary for them to get in contact with users to get first hand information that they can transform in code. Regarding technical skills, these are valued (by both groups of developers: the ones who do not want to talk to users and the ones how think it is necessary) according to the level of creativity that the developers are able to exercise. Being able to develop their creativity at inventing innovative algorithms is more valued by developers than the ability to perform repetitive tasks.

Extract: 6.29

Thread Title: I am fragile	Date: 08th June 2004	Source: JoS

"They _really_ are smart people. My co-worker was telling me, they don't bother about anything else except what's in your head. The interviewer eased him with, "You can roll on the floor, or pull your hair. No problems! Think it over, take your time but give me what you're thinking." when he was stuck with a question."

Sathyaish Chakravarthy

C4-6: Definition: data in this category shows that the term "developer" means different things to different people. As software development is still a nascent practice, there is no general agreement on what a software developer is. From discussions in forums, the following are the roles that a developer plays: architect, systems analyst, business analyst, programmer analyst, coder and tester (see extract 6.30). So far coder is the most played role by members in online communities. However, I found some members doing other jobs as well, for example, database administrator, network administrator and project manager. Despite the variety of roles that could be accounted developers agree in that coding of the software is the central task. My perception is that most developers perceive most of their activities as around programming. For example, requirements gathering and design are activities that tell the programmer what to code, and testing proves the correctness, completeness and quality of the code. Another interesting point that data shows in this category is that most developers perceive themselves as logical, individualistic and non-ambiguous humans. A good number of developers manifest their dislike of talking to users or working in teams. They also believe that every problem situation should be stated in a non ambiguous way. For this reason they usually find it difficult to understand complex

business situations and indecisive users who cannot express their ideas clearly.

Extract 6.30

Thread title: What is a developer?	Date: 23rd June 2004	Source: VB Forums

"Developer - Develops it
Analyst - Analyses it
Architect - Reviews the structure of it
DBA - Guards it.

Here how I have generally experienced it.

The architect determines the over all structure of the application.
The DBA dictates the design of the back end.
The Analyst organizes the business logic
and
The developer writes the code.

I have been in all of these positions, except architect.

I like being a developer though...I like dirty, smelly horse stables!"

Memnoch1207

C4-7: Working Habits: as mentioned in category [Developers/Mentality] developers portray themselves as having poor personal and communication skills. Their working habits confirm this. Most developers state that they would like to work alone in a quiet environment that allows concentration. However, the latest tendencies in development practices are going towards team work. Apparently this is because this increases developers' performance and efficiency at designing, assessing user requirements and at coding. New development environments are open ones where communication is fluent. New agile methodologies emphasise these kinds of environments where pair programming can be practiced. Developers who have not tried pair programming before usually reject it. The reasons are that working with other people would not allow them to concentrate and that other people's work styles might disturb them. The field data shows that in most cases when developers try pair programming for the first time, they like it and support it. Another popular working habit that is worth highlighting is developers' preferences for using electronic tools for communicating with peers. This is one of the reasons why I chose online ethnography as the method to investigate

software developers (section 4.1.2 of chapter 4). Extract 6.31 shows the opinions of two developers who support the use of instant messaging. The main reasons for using these electronic tools are first, that it is easier to send code electronically than by the phone and that these are faster and more effective than face-to-face meetings. Developers dislike physical meetings because they perceive them as a waste of time. E-mails are preferred for broadcasting news among other developers and instant messaging tools are preferred for faster conversations. E-mail and instant messaging tools require less time from developers than meeting in person and allow them to keep working while they are communicating.

Extract 6.31

Thread title: Chat and development	Date: 01st July 2004	Source: JoS
"I've used IM for development tasks in the past where I was working on a geographically distributed team. It wasn't uncommon for us to use IM and phone simultaneously, in fact. Much easier to IM some code to somebody than to try to spell it out verbally! And more interactive than e-mail." John C.		
"Several times I have had to wait week/weeks because I cant complete confirmation on how to move forward on a project, so what do you do, you hold a meeting, what has to take place for a meeting,you have to get everybody together at the same time, you schedule the meeting 'next-week', point being that a week/weeks sometimes has to be wasted, when people could jack into a system and converse on a issue instantly." Berlin Brown		

C4-8: Remote Work: is becoming more common nowadays especially as outsourcing is growing. Improvements in communication and technology allow for international partnerships and teamwork. Remote work is that kind of work in which an individual developer receives a job through an online web site regardless of his physical location. Contact between clients and developers is mostly done online and information and software are transmitted through the internet. There are two opposite views about remote work. Supporters of remote work say it is more convenient for them because it allows them to work on their own time and they do not need to travel or commute to work. Detractors say that remote work is cheap or school kids' labour and implies poor quality. For them no serious work could be done remotely. In extract 6.32 a developer

relates his experience at doing remote work. There is also a growing fear in western developers of remote work done in the third world. Developers' fees are usually cheaper and developers in the States and Europe cannot compete against them. A defensive mechanism that western developers adopt is to portray overseas work as having low quality.

Extract 6.32

Thread title: Chat and development	Date: 01st July 2004	Source: JoS
"I worked for him at 45$/hr for couple of years and finally he stopped giving work to me because he wanted someone right in his office. I never met him nor did I ever talked with him on phone! He used to send me mail with list of things to be done, I will do those changes, upload code on his server and he would pay me once the invoice goes over 1000$. He was wonderful client and I wish I can still work with him!" JD		

6.5 C5: Methodologies

This category contains data about practices and techniques performed by software developers. These practices are part of either formal software development methodologies or informal methodologies created by developers. Some developers state that they do not follow any kind of methodology. However, others say that as long as one is following a set of organised practices one is following a methodology. Advocates of formal methodologies have actually seen these methodologies work in successful projects. They state that methodologies are useful to ensure that one does not forget anything. They also help everyone to talk the same language. This is true also for informal methodologies as long as the team members agree on a set of practices. According to developers it is in big (bureaucratic) organisations where traditional methodologies are favoured as more control over the processes is needed and this is what those methods provide. Contrary, in small organisations, practices tend to be more informal and adapt to the styles of developers or customers.

Opponents of formal methodologies state that full adherence to them can hinder creativity and innovation. Additionally, as Matt H. in extract 6.33 states, no methodology has the right answer for every situation. Some developers would create their own set of practices and some others would not have any. Developers who dislike formal methodologies would like to be free to make their own decisions and not be trapped by a methodology. They say that dozens of methodologies are created just to respond to other methodologies' weaknesses. However, these new ones always bring with them more flaws. Being free to act and make decisions according to their common sense is the right strategy for them.

Extract 6.33

Thread title: Development Methodologies	**Date: 13th July 2004**	**Source: JoS**
"Sooner or later, you're going to have a problem where your methodology tells you to do the wrong thing. So you're forced to follow your methodolgy into foolishness, or say "Wait, no ... that's stupid. I'm going to do blah blah instead." Matt H.		

Data from this category shows seven types of methodologies or practices discussed in online forums. I will briefly discuss these in the next sections.

C5-1: Traditional: methodologies like the waterfall methodology are perceived as big ceremony methodologies. They demand a large amount of documentation and are comprised of rigid steps. Every deviation from the plan is assumed as a mistake and is corrected. Feedback loops are added to the waterfall methodology to account for these deviations. However, developers still think it has severe flaws as it is based on faulty assumptions. The waterfall method assumes that developers can predict every eventuality and that requirements from users do not change (see extract 6.34). In most cases this is not true. Another flaw in the waterfall methodology, as developers say, is the high risk of misunderstanding documentation. As the waterfall method emphasises the division of labour, documentation has to be pass through groups of developers in charge of different tasks. Developers say that it is unlikely that managers,

analysts, coders, testers, and users will have the same interpretation of documentation, especially because they do not work together. These misunderstandings can only be identified when the user sees the final product.

Extract 6.34

Thread title: Objection to XP (extreme programming)	Date: 08th July 2004	Source: JoS

"Waterfall design suffers from a major flaw - the assumption that you can know/predict every eventuality. This, of course, is false.

What *really* happens in waterfall design is that huge assumptions are made. Each assumption relies on previous assumptions. You end up with guesswork tacked onto estimates based on outright lies - a house of cards."

Philo

C5-2: OOD (Object Oriented Design): the most commented feature of object oriented practices is that they promote reusability. Reusability is very important for developers as it helps them to increase their productivity. See Reusability sub category in C3-2: [Software/Characteristics of Software/Reusability].

C5-3: Agile Methodologies: are the new trends in software development. Contrary to traditional methodologies, agile ones emphasise rapid response to changes in requirements. The goal is to keep the client happy by giving him pieces of software in many releases or increments each time with more functionality added. Extract 6.35 shows a summary of agile methodologies I posted in one of the forums.

Extract 6.35

Thread title: This time on Agile Methodologies	Date: 11th Novemeber 2004	Source: VBForums

"Hi,
I've been reading more about SW development methodologies. I found this article: The New Methodology
[http://martinfowler.com/articles/newMethodology.html] by Martin Fowler and I wanted to share it with you. I find it excellent, specially for people who are developing business applications, it has openned my eyes.

To summarise, Fowler says that (and I agree): ... as modern organizations evolve and adapt to their environments rapidly so should software development.

Traditional development methodologies (waterfall lifecycle) were created upon the assumption that requirements never change. Software were created based

on a snapshot of the organization's situation. Projects (price, time and design) were planned in advance and any deviation from them were considered a mistake which had to be corrected.

Agile methodologies are based on the assumption that organisations, users, and their requirements change. Agile methodologies are designed to adapt to changes: they are selfadaptive. To achieve that, users and developers work together and expect change all the time. Requirements, specs and code are reviewed constatly and changed when necessary. For Agile(rs) (is that a word?) things cannot be predicted. They aim for a good quality software that meets the needs of the clients and not for meeting the deadlines and not running over the budget.

It makes a lot of sense, doesn't it?"

Cecilia Loureiro

"Excellent read - thanks :)

We've been working a large contract for customer for almost 3 years now. It was supposed to take 18 months - that proved impossible. Even after we delivered the first 2 of 5 sub-systems, the first ones requirements changed dramatically. We got extra $$'s for re-working that portion."

szlamany

Agile methodologies require a change of attitude from stakeholders. Flexible developers and customers (users) willing to accept uncertainty are needed. (See category C1-6 [Software Development/Characteristics/Adaptability]). With agile methodologies the face of the final product is never visible until the end. However, with "Agile" there are more guarantees that the software will fit the users' needs because of the incrementalist approach and the constant feedback between increments. This is why agile methods are well fitted to ill-defined problems. One characteristic of agile methodologies is that they work better where access to the users is easy. Constant feedback from users using the latest release of the software is always needed before releasing the next increment. When developing of off-the-shelf software (see definition in section 3.1 of chapter 3) agile methodologies are less able to be responsive to change because feedback from users becomes more infrequent.

One of the agile practices most commented in forums is pair programming. Pair programming forces developers to work with peers. When one of the developers is programming, the other will look at the screen to check if any mistake is made. This, "agilers" say, increases

speed of programming, improves the quality of the software developed and allows communication between developers. Not everyone agrees with this. Pair programming is still a controversial technique as many developers prefer to work alone. (See section 3.2.1-C of chapter 3.) One of the pair of developers is forced to see what the other one is coding instead of coding himself. Detractors say that pair programming unrealistically assumes that developers are social beings and that they work well in teams. Nevertheless, evidence from the online ethnography data has shown that developers usually change their minds as soon as they actually do pair programming.

C5-4: Patterns: are standard solutions to common problems in software development. For developers, patterns save them time as the common functionalities of the system to be programmed are already known. Developers have to work only on the new, unique parts of the system. Patterns are popular among some developers. However, it seems that they only account for well defined and stable situations. Where adaptability and unpredictability is the rule, patterns would not help much the developer.

C5-5: Prototyping: for developers is the process of putting all requirements together in a software model in order to test its design with the users. There are two kinds of prototypes. There are paper prototypes which are drawn by developers and then shown to users. According to developers, paper prototypes are easy to produce and do give a good enough impression of how the software will look like. One of developers' biggest fears, as the developers in extract 6.36 mention, is having a user who thinks that the prototype is the final product. A paper prototype would send the right message, that it is just a possible scheme of how the software will be. The other kind of prototype is a working prototype. Working prototypes are developed using real programming tools and contain some of the functionality of the final product. However, as prototypes their objective is only to show the users how the software will behave. The advantage of a working prototype is that it could speed up

the development of the real product if it is well programmed and the users participate in its development. The other advantage is that a working prototype is a tangible proof that some work is being done. This works for users and for project managers who usually work under time and budget constrains. However, paper prototypes could also be misleading if the prototype simply shows that major changes are needed.

Extract 6.36

Thread title: Paper prototyping drawing tools	Date: 15th June 2004	Source: JoS
"AllanL, you forgot the other danger. If the user sees a mockup that looks real, then they think you're done." Steve Barbour		
""They'll think you're done..." -- very good point. It is necessary to balance this perception with the desire to remove ambiguity. With that point, doing a 'paper sketch' version as the original poster suggested has even more merit. It's less likely to be confused with the actual product." AllanL5		

C5-6: Free Will: means freedom for the use of developers' common sense to decide on what is best for a particular situation. It also means to adapt the development process to their needs. Most developers agree in that there is no perfect development methodology that matches all situations. Some developers claim that they do not use formal methodologies but use their "brains" and choose what have worked for them before. Developers assess their development environment, the type of software they will use, the duration of the project, etc. and select the best method(s) that serves their needs. Common sense is developed by developers through their working experience, education and training. A good place to start, some say, is to learn the formal, textbook methodologies and apply them. However, developers should not stop there. They should be creative and try new things when the other ones do not work. Also a change in attitude towards the nature of developers work is also needed. Businesses are not an exact science and do not possess laws as in physics. Businesses are unpredictable due to their human nature. Developers should be prepared to expect changes and adapt to them as well as not trusting any methodology blindly. In extract 6.37, both developers advocate the use of common sense in software

development. Their statements coincide with the common saying among developers "take what works for you and throw out what you don't need." It is not very wise for developers to use methodologies just because other people use them. Developers recommend learning the positive and the negative aspects of methodologies to decide whether a methodology or part of it is appropriate for a particular situation.

Extract 6.37

Thread title: Development Methodologies	Date: 13th July 2004	Source: JoS

"If you walk in with the attitude of "Oh, strict Extreme Programming solves all problems, lets implement it within our group!" and you don't allow for the fact that no methodology is one-size-fits-all, 99 times out of 100 you're going to crash and burn the project you're working on at the time. If you happen to be the 1 out of 100 where It Just Works, you're going to be an extremely vocal evangelist of whatever process du jour you picked and think It Always Just Works. But you'll be wrong."

Mr Fancypants

"My advice: learn a few methodologies, what they are good for, what they are not good for. Then, when you are in a situation, you can pick what is best for the situation at hand. Yeah, I know its a pain, and a few years from now, only the old farts will be doing UML, and the new farts will be doing some new buzzword compliant methodology. It is like asking "what is the best screwdriver?" Because the answer depends on what sort of screw you are trying to use."

Peter

C5-7: Modelling: is part of the design stage and is the process of generating a model of a problem situation and a model of the software that will solve the problem. As Michael Moser in extract 6.38 says, a model is a communication tool. Models are useful to sort out ambiguities between developers and users and between developers and to make sure everyone is speaking the same language. Although modelling is part of most development methodologies many developers are reluctant to use models. Developers who neglect modelling do so because it takes time out of their coding (see also category C1-5: [Software Development/Stages/Design]). These developers see programming and designing as one process which cannot be separated. Some developers who do modelling, like Devil's Advocate in extract 6.38, prefer to use the spoken language instead of a modelling language like UML. UML is found by some as not useful to describe the system which is being modelled.

Developers think UML diagrams are difficult to design and read and see them as an extra burden to their jobs.

Extract 6.38

Thread title: UML	Date: 27th August 2004	Source: JoS

"UML is a communication tool.
When you want explain something about your code, without going into low level details, then the best way to start is a class diagram/sequence diagram."

Michael Moser

"Many people forget the L in UML stands for language. I personally find English a better language to communicate design. Unfortunately some people around here are happier when they can't understand the design documents."

Devil's Advocate

6.6 C6: Business Issues

Into this category I gathered data about the soft side of software development, mainly being how developers and users see the target organisation and its relation to the software. Having developed the model of complex organisations (see figure 1.2 in chapter 1) to explain how I perceived social and business issues within organisations, I expected this category to be very important. However, data in the forums shows that in general developers overlook these issues to favour more technical ones. Nevertheless I gathered data from four categories:

C6-1: Domain Knowledge: is the knowledge about the target organisation(s), its business rules, users, problem situations, etc., that are needed to develop business applications. In big projects, developers' domain knowledge is usually incomplete. Individual developers only focus on the bits that concern the parts of the application they are developing. In general, it seems, it is the manager of the development team who needs to have the bigger picture. This is not the case of small or individual projects where developers have to work on all parts of the software.

C6-2: Business Rules: the main point in this category is that some developers expressed that following the organisation's official, written

documentation (business rules) would lead projects to failure. These developers say that they should pay attention to what their users actually do. The first text segment in extract 6.39 provides an example of how "rules" are not followed by staff. If a developer ignores what users do in real life and just follow written documentation the software would not be of much help. On the other hand, other developers believe that it is not convenient to develop directly for specific users. Instead they advocate development for business rules. The second text segment in extract 6.39 says that as users may rotate and change it is better for the business to have software that serves the broader perspective rather than focus on particular personal preferences.

Extract 6.39

Thread title: Are people complex?	**Date: 15th January 2005**	**Source: JoS**
"I'm going to go out on a limb and make the assertion that EVERY failed ERP implementation is one where the implementation team followed the written documentation and did not know to, or follow what the actual processes performed at that company actually did. Written docs: The TPS report will be generated from sales data in the computer. Actual process: The TPS report is generated from a spreadsheet that Suzy makes. And the interview goes something like: Me: why don't you use the numbers in the computer? Her: they are wrong. Me: why are they wrong? Her: well, Bob enters sales for customers to make his numbers look good, and they return them after the end of the month/quarter (channel stuffing). Fred enters bogus numbers. George doesn't enter anything at all, but since he is the "star" then no one can do anything to him. And the VP keeps 4 sets of books, and the computer sales system doesn't have the numbers for the real books, just the ones that the company thinks are real." Peter		

Thread title: Factores humanos que afectan nuestro trabajo (Human factors that affect our work)	**Date: 9th February 2005**	**Source: Microsoft discussion groups (in Spanish)**
"Pero creo que se esta dejando de lado un aspecto, que así como sistemas y tecnologías cambian a través del tiempo, los usuarios cambian en las instituciones (con esto me refiero a la rotación de personal), por lo tanto si centramos todo nuestro trabajo en	"But I think that we are leaving aside one aspect, as systems and technologies change through time, users also change in the institutions (by this I mean rotation of staff), therefore if we concentrate all our work in satisfying the needs and "caprices" of the user, without having a general	

satisfacer las necesidades y "caprichos" del usuario, sin tener una visión general del negocio (independiente del usuario), podemos crear un sistema cuya eficacia dependa del usuario, y si esta persona se va, su sucesor no va poder realizar el trabajo eficientemente, lo que conllevará a que el nuevo usuario demande cambios radicales al sistema original. Ahora no digo que dejemos al usuario de lado, ni que ignoremos sus opiniones o necesidades, si no que busquemos un balance entre las exigencias del usuario y las exigencias del negocio al realizar nuestros planteamientos." Johany	perspective of the organisation (independent from the user), we can create a system whose efficiency depends on the user, if this person leaves, his successor will not do the job as efficiently, this will result in the new user asking for radical changes in the original system. Now, I am not saying that we have to ignore the user and his opinions and needs, but that we need to look for a balance between the demands of the user and the demands of the business when we evaluate requirements and specifications." Johany

C6-3: Clients and Users: this category contains data about how developers deal with the different stakeholders involved in the development. Developers have to develop software for the users, the managers and the customer. In some organisations, these three roles could be played by the same people. However, when they are played by different people conflicts arise as they possess different perspectives and interests around the software. Users are the people who are going to use the software, and developers should focus on them to design user friendly applications. Management are the decision makers, the ones who know the rules of the organisations and how the software fits there. Finally the customers are the ones who pay the bills. According to developers these three roles are played by people with incomplete views of the problem and trying to make them agree on something is an extra burden on their job.

C6-4: Business side: of software is a category that contains data about the commercial side of software. For developers, software is a service that contains a piece of developers' intelligence and should be written to be sold. Developers' time is also valuable. Some developers state that they work under the dilemma of charging or not charging their users for every single piece of work they do. Although clients could see these

single pieces of work as small adjustments or add ons, developers think that they should discuss if there is a business reason for those. Developers are always under pressure from management to productively use their time and skills. Therefore every single task a client asks for should be fully justified as it costs money and time.

6.7 C7: Social Software

Social software is a term that has evolved from office automation, groupware, CSCW (computer supported cooperative work) and CMC (computer mediated communication) (Allen, 2004). (See a discussion about social software in section 3.3. of chapter 3.) Modern forms of social software include weblogs, Wikis, IM, online forums, etc. From the fieldwork data, I can see that developers define social software as software that allows, encourages or facilitates social interaction (see extract 6.40). However, this field is not very well developed yet, and most practices to develop social software are still trial and error activities. I have classified data about social software in three sub categories:

Extract 6.40

Thread title: Tracing the evolution of Social Software - comments	**Date: 27th October 2004**	**Source: Life with Alacrity Weblog**
"Social software recognizes that people are complex and interactions need to allow people to express their full richness as a person, or a human being. Groupware focuses on processes, roles and permissions." Chui		

C7-1: Usability: social software was considered as a category as it pays explicit attention to social interactions. Most attention in this area is paid to the social interface of software, probably because it is the practical side of this field. Concerns here grew around the fact that guidelines for usability (human-computer interaction) of software are not sufficient when groups of users are using the software to interact. One thing that developers interested in social software stated is the difficulty to figure out

the difference between the individual's and group's goals at using software and how to find out.

C7-2: Design: so far the best strategy for social software design has been to evolve software with "close eyes on users". Initially, a simple product is designed and released in the market. Then developers get feedback from their users, usually through the same online application. Incremental development is used to fix and amplify the most wanted features and reduce the least wanted. Communication with users is therefore vital to evolve software to a state in which it serves the needs of the group. Another point made here is that the changes in social software could affect the culture and behaviour of the group of users. The software will succeed or fail depending on the acceptance of that group. Finally an interesting question was asked by a developer who thinks that the design of social software could be influenced by developers' (geeks') preferences in socialisation (see extract 6.41). I think this is very likely because no formal methods for developing social software yet exist and developers therefore obtain their ideas from personal experience.

Extract 6.41

Thread title: Tracing the evolution of Social Software - comments	**Date: 27th October 2004**	**Source: JoS**
"Would you (the JoS audience) say that the "geekiness" of programmers has creeped into the social software they created, with negative consequences? Will it do so in the future? " Mack		

C7-3: Communication: this category explores social software as a facilitator of communication between users. Developers recognise that communication is a user's goal and must be considered as a first priority. According to social software enthusiasts, there are two characteristics of social software that could boost communication. First, software must provide different ways for interaction between users, however and whenever they like. Second, software must provide a shared space where people can gather and leave messages. These two characteristics

will help a community to grow around the software, and hence will make it successful.

6.8 Chapter Summary

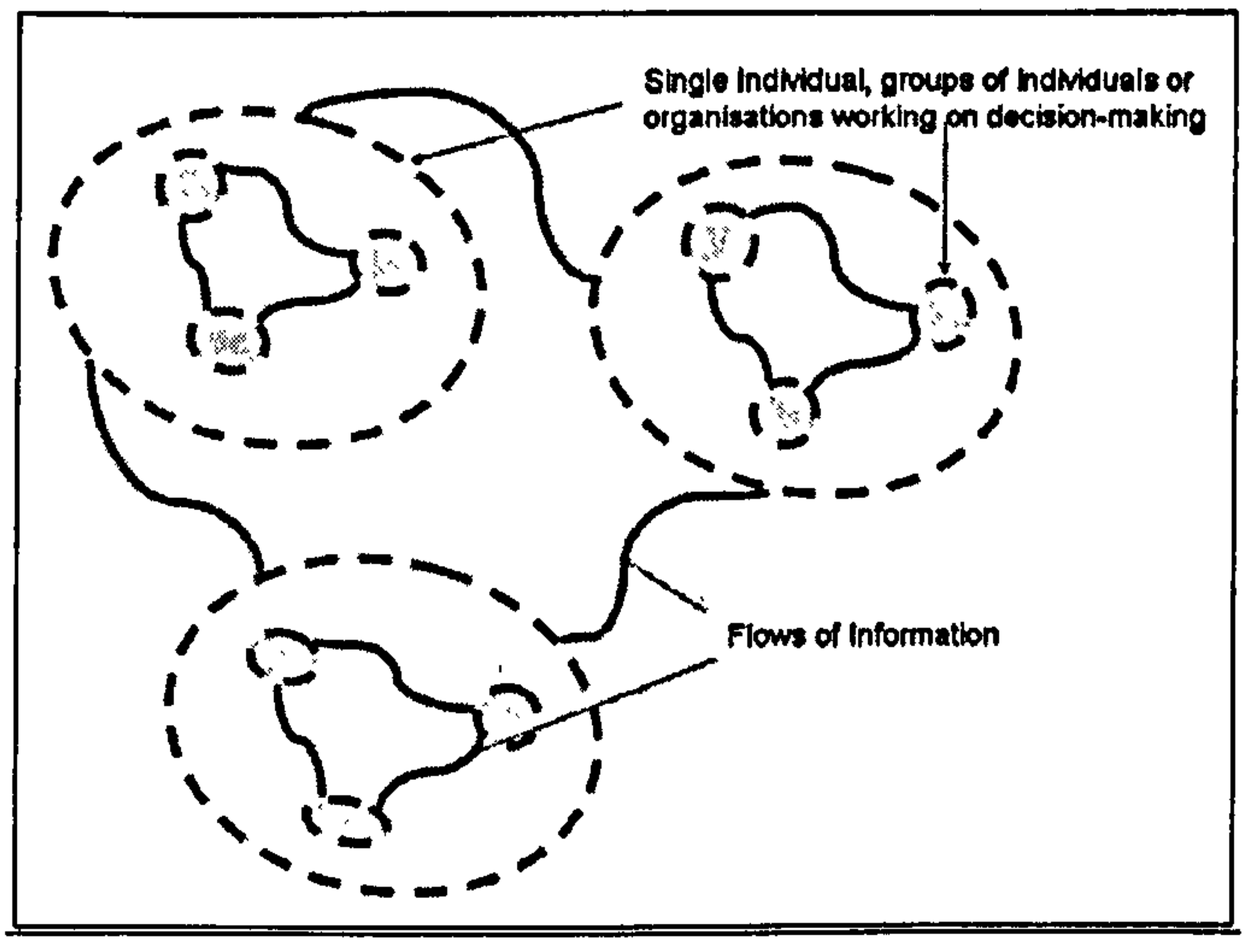
This chapter presented the data collected in my online ethnography as organised by categories. The data was presented in the context of the online conversations I collected. This exposition provided a descriptive picture of software development for complex organisations which was built from my interpretations of developers' understandings of their jobs. This picture is a useful contribution as it gives an idea about what developers think of the current state of software development for businesses. However, in its current shape, these findings do not build theory (Carroll and Swatman, 2000, 239). Wider implications of the findings to the research questions need to be considered to create theory. In the next chapter I provide a descriptive theory based model which explains how software developers make sense of complex situations and build mental models of organisations. The aim of the model is to introduce to developers relevant social issues that influence and affect their work and the software they produce. The model classifies the issues in three contexts: the development environment, the development practices and complex organisations, and explains how issues happening in the first two contexts shape developers perceptions of social aspects in the third context.

Chapter 7: Model of Social Issues in Software Development.

Having presented my interpretations of developers' views, based on the online ethnography, about the current state of software development for complex business environments in chapter 6, the aim of this chapter is to reshape those findings into a theory based model. Chapter 5 presented the data analysis strategy as a process of category generation and refinement followed by a stage of concept and proposition generation. Chapter 6 presented these concepts and propositions in the form of a broad picture of social issues surrounding software development. Following that, in this chapter I will build a descriptive model of social issues from these data by generalising and organising those findings. The model will describe the process that developers go through when they make sense of complex business situations and will address social issues in three different contexts. Additionally, the model of social issues in software development is presented as a complement to Orlikowski's model of "enactment of technologies-in-practice". Orlikowski's model explains how social structures emerge from people using technology and how use may change in time. My model of social issues addresses the impact this has on software developers' work. The first section of this chapter will re-introduce the social aspects of the model of complex organisations. The second section briefly discusses Orlikowski's model of "enactment of technologies-in-practice". Sections 3 to 6 introduce the model of social issues in software development, and discuss the social issues under each aspect of the model. Finally section 7 summarises the chapter. Additionally, section 8.4 of chapter 8 will take the ideas presented in this chapter and propose directions for future research which include further developments of the model of social issues in software development by concepts from Orlikowski's and Robey (1991) structuration model of technology and Orlikowski's (2000) enactment of technologies-in-practice.

7.1 Social issues in business organisations: the model of complex organisations

As presented in the introduction of this thesis (section 1.1. of chapter 1), the motivation of this research emerged from my working experience. Complex human and social issues arose during the design of a decision making system. During my participation in that project I found that these human and social issues were not addressed by any of the development practices we knew, and that our lack of knowledge of social aspects was hindering our work. In a process to address this lack of knowledge, in this thesis I have built a model of complex organisations that represents and generalises the issues I found at work. I developed this model from my pre-understanding of social issues and it does not gather other developers' views. The purpose of this preliminary model was to state my perspective of social issues in business organisations at the beginning of this study. I also used this preliminary model to extract a starting list of social issues which I used to inform online conversations with other developers when engaged in the online ethnography. That initial model of complex organisations is shown in figure 7.1 (repeated from figure 1.2 in chapter 1), and is a representation of working organisations which are interacting through decision making processes.



- Circles are actors
- Actors can be individuals, group of people or organisations
- Group and organisational boundaries are defined by people's perceptions (dotted lines)
- Continuous lines represent Flow of Information
- Processes can be started and ended by any actor
- Processes can take any shape
- Processes can split in two or more processes; and two or more processes can merge in one
- Extraction, creation and use of information is implicit in the model

Figure 7.1 Model of Complex Organisations

This model is discussed in detail in chapter 1. In summary, circles in the model represent individuals or groups of individuals interacting with each other. The lines represent flows of information and interactions between the actors. Complex human and social issues arise in these organisations. For example, interactions between actors are unpredictable and adaptive. Actors have the ability and power to decide and act as they wish depending on the situation. Actors also learn from previous experiences and improve their practices to new situations. Decisions can be taken individually or in groups. Collaborative group work is usually complicated as it involves many interactions which are subject to individual working styles and different agendas. Decision making processes, represented by flow of information between two or more actors, could start and end at any point and take any shape. Information is processed at all stages of the process. The needs of information vary from actor to actor and from process to process. In brief, this model represents highly unpredictable, adaptable multi-group work environments which do not correspond to the static, mechanical perspectives of business organisations on which the majority of literature on software development methodologies and practices are based.

However, problems arise when developers are forced to portray organisations as predictable and stable in time. These mechanical perspectives only hinder their ability to deal with social issues, such as unpredictability and adaptability of people, and hence do not help them to address these characteristics when they design software.

This thesis aims to address these human and social issues by providing a theory based model that highlights social issues that surround software development. This model will acknowledge the impact that adaptive and unpredictable use of software has on developers' work, their environment and their development practices. The data from the fieldwork shows that software developers perceive social issues happening in business organisations in different ways, and that these perceptions are partially dependant on preconceptions from the developers' environment and development practices. Awareness of the nature of social issues in organisations will help developers to comprehend users' behaviour. That will change developers' attitudes towards software and help them to decide which social aspects of organisations need to be addressed and moulded in software.

Also addressing social issues is important for developers' work because it will help them to improve the quality of their software and improve the rate of acceptance from their users. This is crucial for developers to keep their jobs. One of the main concerns stated by developers in online forums is the risk of failure or users' rejection of software and developers. Failure and rejection happen when software does not satisfy users' expectations and/or is not of any help at their work. In addition to technical issues, other social issues usually neglected by developers, also influence users acceptance. The data shows that according to developers their users can and do use their software as they wish regardless of the developers' intentions. Users will choose which properties of the software they are going to use and will also decide on the way they are going to use it. Developers are happy with this as long as their users are happy with the software. When users' needs change

(they inevitably evolve over time) and the software no longer helps much, developers are asked to modify it. Developers make those changes to software according to users' requirements and expect these will be accepted by users. Problems for developers arise when this maintenance work is not paid accordingly or when the software is rejected by users and developers lose their jobs. A model that addresses social issues will provide developers with a complementary view to the technical one (already held, as the data shows) that will help them understand the nature of their users and organisations and how to avoid rejection from users due to human and social issues.

The initial model of complex organisations is a model that emerged from practical experiences and captures some social issues as I perceived them from my working experience. However there is complementary theoretical model that focuses on how social issues emerge from human action. In the next section I will briefly discuss Orlikowski's model of "enactment of technologies-in-practice" which explains why users of technology are so unpredictable and changing. Orlikowski's model is relevant to the initial model of complex organisations because it concentrates on the relationship between social structures and the use of technology within organisations. However, this model of "enactment of technologies-in-practice" does not address the impacts that changing social environments have on developers of technology. Although Orlikowski and Robey (1991) suggest different avenues for investigating structuration processes during IS development (Jones et al., 2004, 318; Orlikowski and Robey, 1991, 159), Orlikowski has concentrated her research on organisational issues rather than in IS development. This is a gap that the model of social issues in software development aims to fill.

7.2 Model of enactment of technologies-in-practice

A perspective that addresses changing business environments is presented in the model of "enactment of technologies-in-practice" by

Orlikowski (2000). Orlikowski developed a model based on Giddens' model of the "duality of structure". The "duality of structure" states that social structures and peoples' actions recreate each other over time and space. Social structures are rules and resources which exist only in memory traces in humans' minds. Social structures are made manifest only in the instances when they are drawn on in action and interaction (Walsham and Han, 1991, 78; Rose and Hackney, 2003, 258). Similarly, human action reproduces existing structures, produces new ones and brings change (Walsham and Han, 1991, 78). Orlikowski (2000) uses these recursive concepts to explain how people enact social structures when they use technology and how people draw on those structures to use technology. Orlikowski proposes a practice lens to look at users of technology and at emerging structures from that use as opposed to looking at the properties of technologies and how people appropriate them.

The model of "enactment of technologies-in-practice" explains why complex organisations are adaptive and unpredictable at using technology. Orlikowski says that designers inscribe certain properties in the technology which users may decide to use or to not use. Therefore a particular technology can be used totally or only partially by users. The use of technology depends on users' needs, knowledge of the technology and situation, previous interactions with technology and "emotional and intellectual" attachments associated to the particular technology (Orlikowski, 2000, 410). Also, Orlikowski's model, shown in figure 7.2, presents "technologies-in-practice" as a kind of social structure which emerges from technology use. Orlikowski explains that users of technology are situated "within a number of nested and overlapping social systems" (Orlikowski, 2000, 411). Therefore the use of technology will recreate other social structures apart from "technology-in-practice". Similarly, users will draw on many varied social structures when they use technology. As technology use depends on too many social factors it cannot be predicted easily.

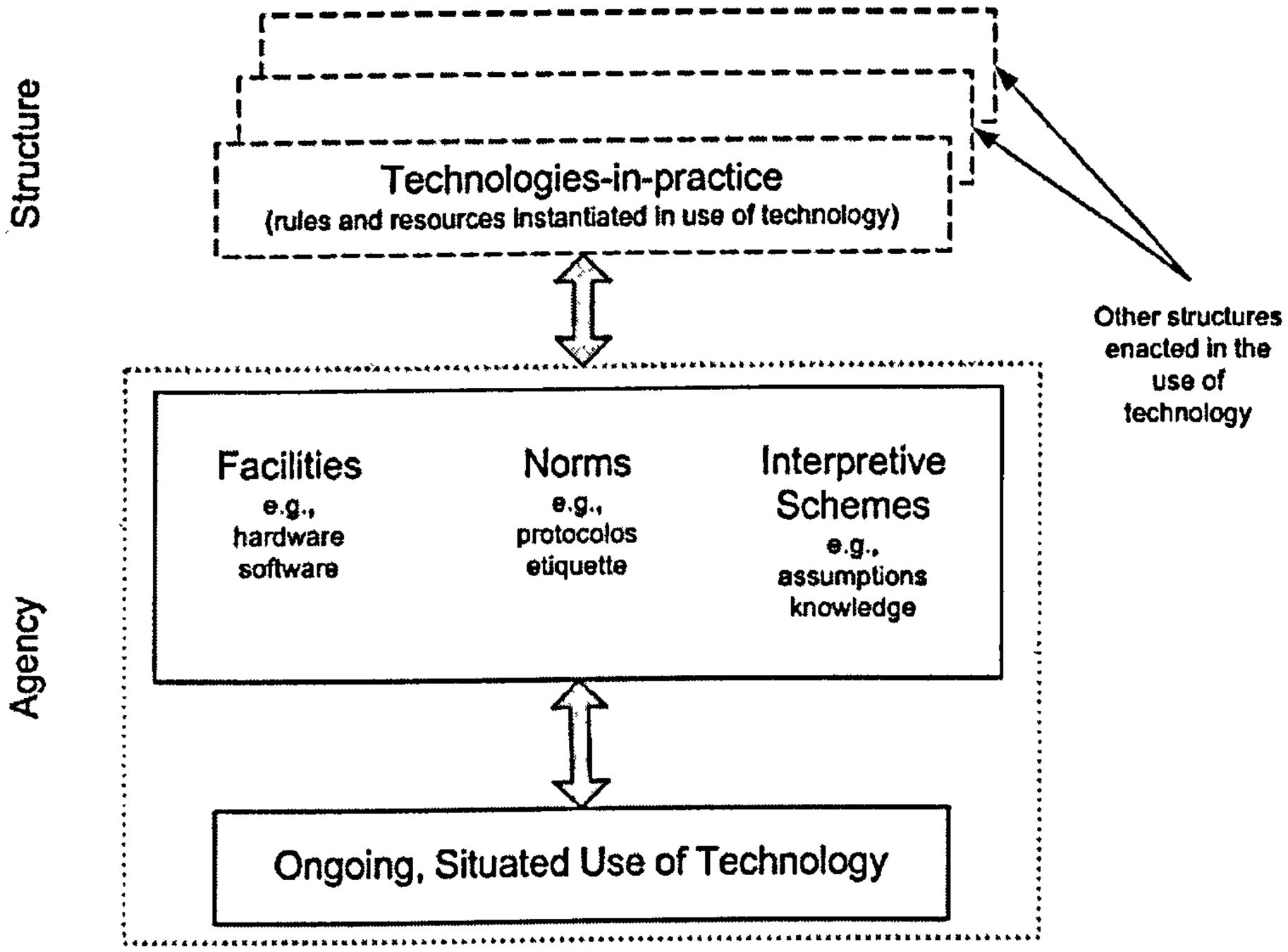


Figure 7.2 Enactment of Technologies-in-Practice
Source: Orlikowski (2000, 410)

Use of technology is also adaptive and evolves through time. Continuous use of a particular technology may be seen as becoming routinised. This could happen for a period of time. People would think of technology and social practices surrounding it as stable. Routines, habitual, taken for granted day-to-day activities (Rose and Hackney, 2003, 260), will develop and become institutionalised within the organisation. However, according to Orlikowski, this routinised and stable state is only provisional as "different elements continue to be developed, existing functions fail and are fixed, new materials are invented, new standards are set and users modify the artefacts and/or its content for new and different uses." (Orlikowski, 2000, 411). Users may choose to adjust their technologies-in-practice intentionally in response to changes in their environment; improvise to respond to new opportunities or challenges or they may also fall into errors at using the technology.

As shown, Orlikowski's model presents the use of technology as an adaptive and unpredictable process. Software developers have the job of creating software that can respond to these characteristics. Orlikowski's model presents a face of technology use that could allow social issues in business organisations to be explicitly recognised but fails at addressing the implications that this adaptive and unpredictable use has on developers work. Although Orlikowski's model has such key implications for developers it does not directly address them. The data collected however addresses these issues and is useful to complement this model. In figure 7.3 I present a framework that presents the model of complex organisations, Orlikowski's model of "enactment of technologies-in-practice" and situates my final model of social issues in software development as facing social issues (in complex organisations and the development environment) and completing the cycle of software development.

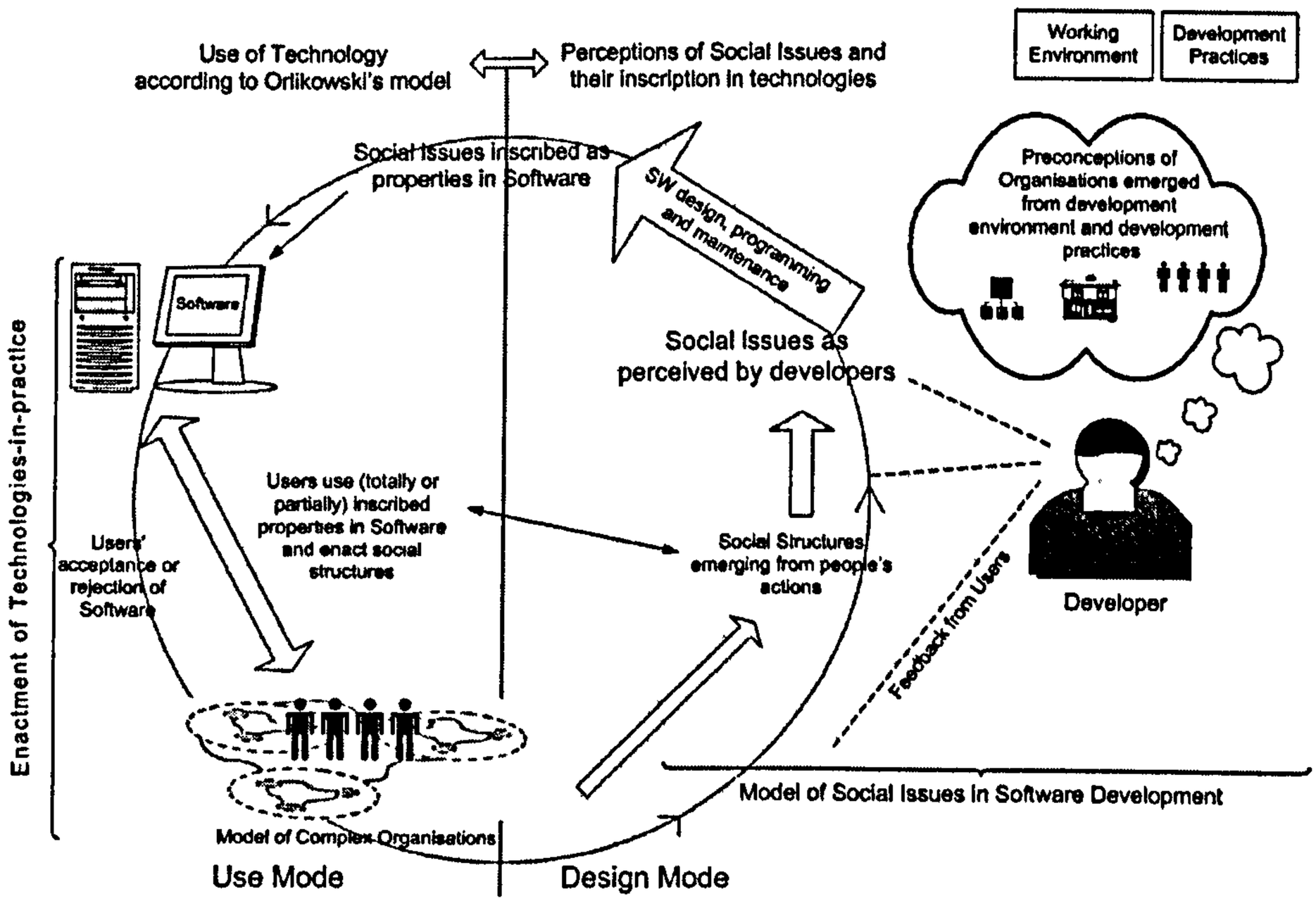


Figure 7.3 Framework of Software Development

Starting from the initial model of complex organisations (bottom left and reading clockwise) the framework presents social structures emerging

from complex organisations that use software. These social structures are perceived by developers and engraved into the software that they develop. Users of software may use totally or partially the software or may reject it. When the needs of users change, and the social structures emerging from the use of software change, developers are asked to modify the software. The cycle is repeated again, as developers have to examine the new social structures and requirements from users to adapt the software. The model of social issues in software development describes the process developers go through when they make sense of the adaptive and unpredictable multi-group organisations represented by the model of complex organisations and enactment of technologies-in-practice.

In an earlier work Orlikowski (1992) presented technology as a duality which comprises two modes of technology. The first one is the "design mode", and presents technology as created and changed by human action. The second mode is the "use mode" and sees technology as used by humans to accomplish some action (Brooks, 1997, 139). Orlikowski (1992) acknowledges that there is a misleading and artificial (Orlikowski, 1992, 409) disjuncture between the design and use stages. This means that a one-sided view of technology arises because one mode of technology is always invisible. Therefore designers of technology will adopt an open systems perspective whereas users will treat technology as a black box (Orlikowski, 1992, 407). For Orlikowski a structurational perspective on technology would posit "artefacts as potentially modifiable throughout their existence" and hence will portray technology as designed and used recursively (Orlikowski, 1992, 408). Although using the time-space discontinuity is useful for analytical purposes Orlikowski (1992) states that both modes are tightly coupled. The framework shown in figure 7.3 presents software as a duality with the design mode on the right hand side and the use mode on the left hand side. The model of social issues in software development presented in the next section describes how the design mode of software is influenced and affected by the social issues arising in the use mode. In this way I am recognising the

tight interrelationship between both modes of software and reconnecting them to remove the discontinuity.

7.3 The Model of Social Issues in Software Development

Before presenting the theory based model of social issues in software development I will remind the reader of the research aim already stated in chapters 1 and 4 as it is connected to the objective of the model:

> To identify and explain how the social issues within the software development environment and the social aspects of complex organisations shape software and the process of software development.

The model of social issues in software development created here answers to this aim and reveals "the system of interpretations and meaning, and the structuring and organizing processes" (Gioia and Pitre, 1990, 588) that happen during software development for complex business situations. This final model presents the social aspects surrounding software developers and organises these aspects in a sequential fashion which shows the order in which these aspects influence developers in their jobs. In other words, the objective of the theory is:

> To build a picture of Software Development for Complex Business Situations that describes and explains the process that software developers go through when they make sense of social aspects of the target organisation, the information system and the users that the software has to serve and when they assess and compare the technical aspects of software with the social issues within business settings.

Figure 7.4 shows a schema of the model of social issues at an earlier stage of development. Data from the fieldwork showed me that there were human and social issues emerging from the development environment and practices, and from within the target organisations that

influenced developers' perceptions. I found that the way developers perceived social issues in their target organisations depended on or were affected by the particular social issues surrounding their environments and practices. I noticed this by seeing the difference in perceptions between developers' who where working under different circumstances.

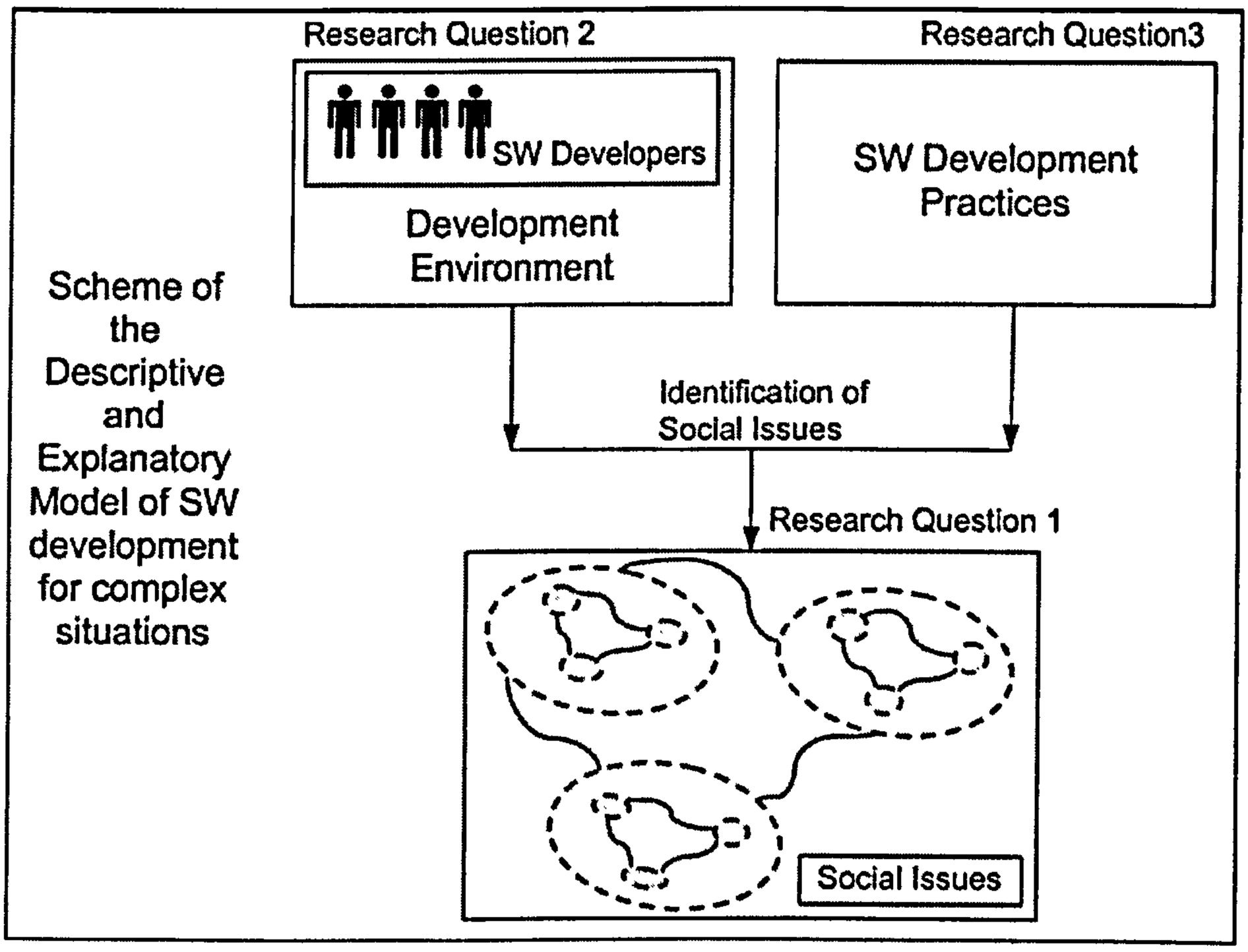


Figure 7.4 General Schema of the Model of Social Issues in Software Development

The schema of the model shows a flow (represented by the arrows) composed by the research questions (represented by the boxes). This flow shows how social issues are identified. Thus this model helps to understand the processes through which software developers make sense of their software and users. The model shows that the three research questions tackle social aspects within three different contexts. The first question addresses social issues within the context of complex organisations. The second question tackles social aspects in the software development context and the third one focuses on formal methodologies or informal practices (see table 7.1, repeated from table 1.1 in chapter 1).

Within each context the social issues affecting software development are listed and thus the research questions are addressed.

Q1: What are the human and social aspects of business environments that make software development complex and difficult?	Social Issues in Complex Business Settings
Q2: What are the social aspects in the development environment that allow or hinder the identification of social issues in business settings during the development of software?	Social Issues in Development Environment
Q3: What aspects of current software development methods address or neglect social issues in business settings? What is social software and does it have a role to play in improving these practices?	Social Issues considered in Development Practices

Table 7.1 Overall Contexts covered by the Model

Following the summary outline from figure 7.4, figure 7.5 shows the full model that describes and explains the process that software developers actually go through when they develop software for complex business settings. Each box represents the contexts that influence software development and the arrows represent the flow of the process. In each box I have included, in bullet points, the relevant social issues within each of the three contexts that influence software development. These issues were developed from the propositions that emerged from the process of data analysis. The process by which these propositions emerged is explained in chapter 5, section 5.2.3.

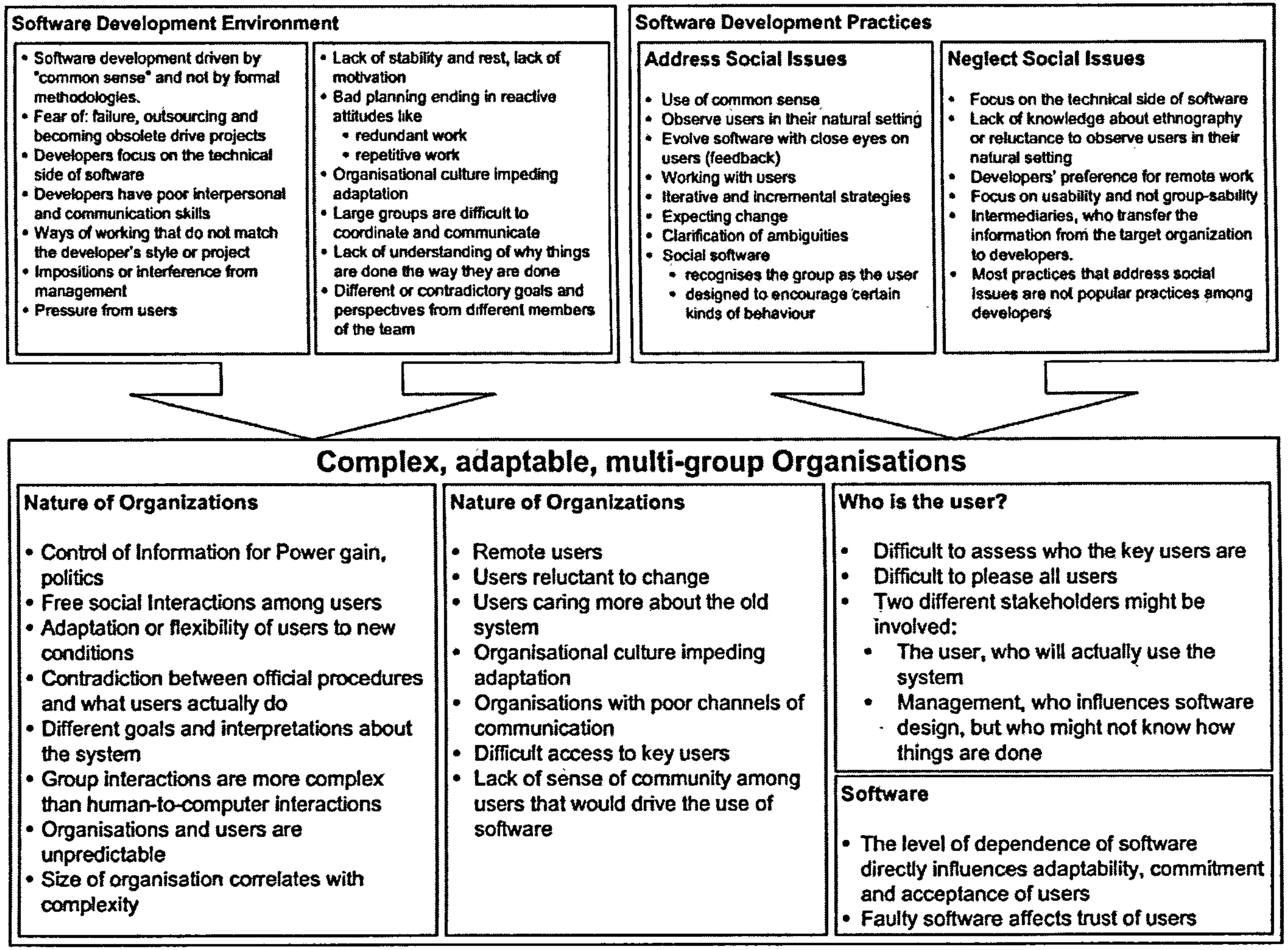


Figure 7.5 Model of Social Issues in Software Development

The model suggests that the software development process starts from previous experiences developers have had: their past or current job settings and formal or informal development methodologies that they have used or known. The flow is completed when developers are exposed to the complex organisation that is using the software and face

the social issues there (which then become the "previous experiences" in future development processes).

From the categorisation in chapter 6, data categories show that when software developers undertake a new project it is **not** the aspects of the target organisation which are first addressed or acknowledged. Preconceptions of the target organisations are built upon developers' background knowledge and experience. For example, developers working with object oriented languages will think of organisations as a set of interrelated objects, with features and methods, which can be encapsulated and coded. Also, developers using traditional waterfall methodologies will assume that organisations and users' requirements are stable and do not change through time. This background knowledge and past experiences influence developers' first thoughts and activities, and are presented as issues within the two contexts shown in the top of the model: software development environment and software development practices. Past or current arrangements at their work place dictate how they will start. The organisation of project teams and/or specialised groups that perform part or the totality of the development process influence the way they will start looking at the problem situation and finally how they will perceive it. For example, the perspective from a software developer who works at a software house where management dictates how teams are arranged and labour is divided would be different than the one who works remotely, as freelancer and has developed his own personal style.

In extract 7.1 I show a comment from an online participant, Jonathan A. He is a freelancer and does work remotely. He rarely, if ever, does have face-to-face contact with the final users of his software nor with their managers. Nevertheless, Jonathan confesses that this arrangement has worked for him and his contractors. In a case like this the developer builds a picture of a complex organisation from the picture given to him by other developers. For him the target organisation is an idea comprised by bits of organised information provided by other developers. Whenever he

needs clarification he asks through electronic media. Apparently the level of knowledge he gets about the target organisation is good enough to get his job done. Therefore, whenever Jonathan starts a new project he will use electronic communication because it has worked for him. He will also think of the new target organisation as bits of organised information provided by a third party because this is the way he is used to think of organisations. This, however, would not be the case in other kinds of projects, where the working practices assume physical presence is required to guarantee full understanding of all important issues.

Extract 7.1

Thread Title: Remote software work by bidding sites & online ads	Date: 30th June 2004	Source: JoS

"All 3 of these clients do web development for their own clients so I'm their programmer and they typically already have the design talent in-house."

...

"An additional thing is that I'm able to speak the business speak for the most part and I communicate with them intensely. Most of this communication is via email or IM. I am involved with some conference calls when we need to talk to one of their clients. I really like this arrangement because I have people out there drumming up work for me (their sales people) to keep me a fairly constant flow of work."

Jonathan A.

Additionally, software development methodologies preferred by developers or imposed by team leaders will guide developers on what, where and how to look for relevant information in the target organisation. Depending on the selection of methodology different software could be created for the same situation. For example, very technical approaches would make developers overlook social aspects that could be significant for the software to be successful. Also these kinds of approaches are generally not responsive to changing environments and will struggle when the context evolves and users change their requirements. Recent approaches however have been developed to allow developers to deal with changes. The most popular are the agile methodologies like XP: Extreme Programming formulated by Kent Beck (1999) and The Crystal Family formulated by Alistair Cockburn (2000a). These approaches

emphasise close work with users and rapid response to changes. Usually software is delivered in pieces to allow time for users to use the software and provide feedback, and for developers to continue their jobs without the pressure from users but with useful feedback from them. These approaches see software development not as a "design – implementation – use" sequence but as a continuous evolutionary process, closer to a service than a discrete artefact.

In extract 7.2 I present a different example of methodologies influencing a developer's perception. In this case a developer needs diagrams to understand organisations. The developer comments on how he struggles at getting the big picture of a situation when only verbal communication is used. He prefers to draw his ideas using diagrams because this allows him to decompose the situation into smaller understandable pieces. If he were not to use diagrams at all his perception of the problem situation would be incomplete. Whenever "meagain" faces a new assignment he perceives the target organisation and problem situation as a diagram rather than a set of words. In this particular case the use of a methodology that emphasises verbal communication, such as Scrum (an agile methodology which encourages daily face-to-face meetings to assess progress and upcoming work), would affect negatively the developer's perception of a target company and would affect his productivity.

Extract 7.2

Thread Title: Why Not Use Diagrams and Models	**Date: 16th June 20004**	**Source: JoS**
"Ummm this is interesting. I am quite fond of just taking a pencil and paper before I do things. When I worked at an ad agency I always whiteboarded stuff. I like to take something and organize it at a big level into simple, understandable parts ... they can be decomposed until the code level. But ... like the OP [Original Poster] ... that is not the way it's done where I'm at now. People prefer talking and all *artifcats* a programmer receives is through imho duck soup word docs. They ramble. Once you start to concentrate you are forced to ask questions. I had to ask, and ask again to get screen shots. I have a feeling that maybe people are just not comfortable enough with the domain to put anything on a whiteboard. If they put a lot of words in a template it then becomes something when really it's just duck soup but if only you recognize it for what it isn't, what do you do? "		

meagain

As seen, background knowledge and experience shape the eyes through which software developers will see complex organisations. Depending on these eyes different pictures of the same organisation could be drawn, independently of the particularities of the organisation. Social issues, if perceived, would be seen differently and would affect differently the final product. Additionally, the particular social issues happening in the target complex organisation, as perceived by developers, will shape further their perceptions at a later stage. On the one hand, positive feedback from users who are happy with the software and find it useful will reinforce developers' perceptions of social issues. On the other hand negative feedback from users who complain about the software will make them reconsider their views on the situation. This reconsideration could lead to changes in their current arrangements as project teams and methodologies or informal practices, changes which again will reshape their perceptions of the target organisation. When these perceptions are stabilised, social issues within complex organisations are identified, understood and considered for software development.

Extract 7.3 shows an example of how negative feedback makes one developer look for different alternatives whereas positive feedback reinforces another developer's views on organisations. In the extract Patrick needs help on how to deal with traders and Steve Jones (UK) shares his experience in similar environments with him. Patrick has a preconception of trading environments. He portrays traders as difficult people to work with. Put in the context of my theory, Patrick has received negative feedback from his users and needs to reconsider his view on the situation. Therefore he is looking for a different methodology that could give him better results. Steve Jones (UK) agrees in that traders are tough people but also says that there is a better way to see this and recommends a new strategy. Steve Jones (UK)'s exposure to traders' environment made him shape his perception of traders differently. He thinks traders' environments could be fun environments to work in. He

has also developed a strategy of under-promising and over-delivering which makes him and his users happier. He probably tried different alternatives before and this one worked. Thus in terms of my model, the positive feedback on his job has reinforced this view of organisations. Now he associates the under-promising and over-delivering strategy with trading environments. Similarly, as this strategy works and both developer and users are happy, his perception of traders became a more positive one.

Extract 7.3

Thread Title: Methodology for trading floor environment?	**Dates: 08th and 09th July 20004**	**Source: JoS**

"i was wondering if anyone had opinions of based methodology for developing software on a trading floor environment?

let's face traders can be jerks. they change there mind, they don't think. has anyone tried using XP? Has it really worked? or have you done just code-and-fix and solve whatever was thrown at you?"

Patrick

"In my experience of working with traders on the trading floor, it can be tough, but a lot of fun too.

As others have mentioned, traders time is for trading, anything else is eating into their p&l, which untimately eats into their bonus.

The best thing to do is to understand everything you can about their job and how they and/or their team works. In this way, you can just help them and they will be happy.

Don't be tempted to offer them the "best" solution. They don't give a crap about how elegant your shit is, they just want it to work.

My best advice is to under-promise and over-deliver, as with all projects, but with these guys especially. If you do a good job, they'll give you more work and you'll be set up."

Steve Jones (UK)

Having explained the flow of the process in the final model of software development, I will discuss the three contexts that make up the model and will relate them to the three research questions. As the discussion is long and for more clarity, I am dividing the discussion of each part of the model in different chapter sections. I will start from the software development environment (section 7.4) and software development

practices (section 7.5) contexts as they come first in the flow of the process of development. Then I will discuss the human and social issues in complex business settings in section 7.6.

7.4 Social Issues in the Software Development Environment

Having looked at the process developers go through when they make sense of social issues at designing software I will now explore in more detail the social issues within the context of the software development environment. The software development context comprises the physical setting where developers work, the way they are organised as teams and the overall atmosphere among developers. Contradictory perspectives emerge from development groups: from the managerial ones to the more technical ones. Development environments are usually a mixture of different (professional) backgrounds, ideologies and working styles that have to work together as a unit to be successful. Individual preferences shape the styles of the group at different levels and the group as a social collectivity (Checkland and Holwell, 1997) with its own identity also shapes the styles of its members. Working styles in a development environment are a reflection of the perspectives developers have of their jobs and their assignments. Additionally, their working styles also shape their perceptions of working environments and assignments.

In the following discussion I will explore how the combination of social aspects in the development environment found in the fieldwork influence developer's perspectives on complex organisations. These perspectives shape the way they perceive their users, how they understand business processes and address them at developing software. These aspects are shown in figure 7.6 as extracted as subsection from figure 7.5.

Software Development Environment

- Software development driven by "common sense" and not by formal methodologies.
- Fear of: failure, outsourcing and becoming obsolete drive projects
- Developers focus on the technical side of software
- Developers have poor interpersonal and communication skills
- Ways of working that do not match the developer's style or project
- Impositions or interference from management
- Pressure from users

- Lack of stability and rest, lack of motivation
- Bad planning ending in reactive attitudes like
  - redundant work
  - repetitive work
- Organisational culture impeding adaptation
- Large groups are difficult to coordinate and communicate
- Lack of understanding of why things are done the way they are done
- Different or contradictory goals and perspectives from different members of the team

Figure 7.6 Social Issues in the Software Development Environment

Social issues within the development environment emerge from four sources. The first source is the developer himself, his ideologies, feelings and personal preferences. The second source is the development project manager and the relationship between him and the development team. The third one is the user (or client) who can play a double role. Users are an important source of information for developers as they possess first hand knowledge about the target organisation. Users as well can become developers' supervisors and put pressure on them. The fourth source is the organisational culture in which the developers work. In the case of developers who work alone, organisational culture would equate to the personal style of the developer. Some of the aspects discussed below emerge from more than one source.

One of the most influential human issues found in the data is the use of "common sense" (as developers call it) for developing software. Common sense is a set of ideas emerged from experience which not necessarily have valid and reliable supporting evidence. Common sense is considered as part of the development environment context because it is a sign of the attitudes of developers towards their work. At work most developers trust more their common sense than formal procedures. A

good developer is a developer who can use his common sense and discern the best solution for a problem. In some environments developers who follow to the line methodologies without questions are seen as talentless with a lack of creativity.

Common sense was also explored in chapter 6 in category [Methodologies/Free Will]. The reason for using common sense, it seems, is the difference they find between the assumptions that standard methodologies have about organisations and what organisations are in reality. There is a popular saying among developers that says "there is no silver bullet", meaning there is no methodology that works in all situations or there is no magic solution in software. Developers experience this in practice when the methodologies that they use fail. The reason for this failure particularly in the business applications development field could be the wrong assumptions these methodologies come with. Coming from an engineering origin these methodologies emphasise a mechanical perspective of organisations. Such mechanical perspective is different from the initial model of complex organisations presented in the introduction of this thesis. This model suggests that organisations are unpredictable, adaptable and are comprised of groups of people working together in different ways. Attempts to mechanise these environments usually fail because of the human and social factors involved. Another factor that affects developers' perceptions of formal methodologies is that they are usually taught in a contextless fashion. It is assumed by many developers that the same methodology can be used to develop any kind of software. The problem arises when methodologies do not work in certain environments and when developers do not understand why this happens.

Contrary to hindering the identification of social aspects within complex organisations, common sense would in fact help the developer if used appropriately. Common sense shapes developers perspective of complex organisations as it gives them an open minded perspective with no predefined models. By using their common sense, developers address

the unpredictable and adaptive nature of complex organisations. In chapter 6, category [Software/ Characteristics of Software] I explored how unpredictability, adaptability and other characteristics of complex organisations are reflected in the software that developers produce. Common sense adds the human factor to software development practices that is needed to develop in a more agile way and to be able to sense these human factors happening in complex organisations. However, this common sense needs to be developed in a comprehensible way. Developers should be able to understand why things work in some situations and not in others. Data shows that the lack of understanding of development practices hinders developers' ability to understand their assignments. This is especially true for very technically oriented developers who are facing complex social environments. In these cases developers with a lack of understanding of the underpinnings of a methodology will follow them to the line and would not know what to do if a problem arises. As most development methodologies are technically oriented and do not account for social issues it is very probable that developers using them would face obscured contradictions. Understanding why things are done the way they are done is an important factor that may open the eyes of developers toward social issues in organisations. Additionally it will help developers to develop their common sense into proven and more reliable knowledge (closer to the reality of complex organisations).

Fear also shapes developers perspectives of target organisations. Fear is a factor that distracts the attention of developers from the social aspects to the technical aspects of development. The most commented on kind of fear is fear of failure which is caused by pressure from users over developers and management. Developers receive twice the burden of pressure as mangers themselves transmit their stress to developers. Fear would make developers focus on their personal productivity and performance to finish "good enough" software and would make them neglect important social factors in organisations. As productivity is assessed by the speed of their coding and the technical quality of their

software, other factors are obscured. When users exercise pressure over developers, developers' views of users also changes. It becomes difficult for developers to picture their users in their real working environments and to focus on the way users actually do things. On the contrary users become stressing agents for developers. Developers know that the picture users create of themselves is usually incomplete and comprised only of their personal goals. However, developers perceive it is not in their hands to judge this but only to do as their users and managers wish.

Other important human aspects are those personal characteristics that developers share and that shape the developers' stereotype as a group. As mentioned in chapter 6, in category [Developers] developers portray themselves as having poor social skills but possessing high technical proficiency. Coming from a computer science background most developers have very mechanised views of organisations. Additionally as developers' work is purely technical and is assessed by productivity it usually does not give space for extra social concerns. These factors combined with the developers' inability to relate with other human beings (as they themselves state) make it difficult to talk to and understand users. This inability is also reflected within the development working environments. Very few developers like to work with other people let alone work in open environments. This attraction for isolation limits developers from sharing points of view and considering perspectives that they have not considered. The consequence is that developers end up having incomplete views of the problem situation they are working with. Additionally these views of organisations could be different from the ones of other developers. Usually this problem of diverse views is solved by attending meetings and sharing experiences. Social factors happening in complex organisations come to the surface of developers' pictures when developers exchange ideas among them and with clients. However, as the account in chapter 6 describes, developers, in general, do not like meetings finding them a waste of time.

Finally, project managers can force views on developers. This could be positive when managers understand developers' work and possess a wider perspective of target organisations. However, bad management could become a factor that hinders the identification of social aspects in organisations relevant for software development. Data also shows that developers resent very much when managers do not know what developers are doing and lead them (developers) to the wrong direction. In essence, developers do not take well impositions that do not match their points of view and are perceived to not work. According to developers good managers should know the nuances of their work and the particularities of each individual working under their command. Management that imposes the wrong methodology to developers will limit them in their jobs. As mentioned before, as managers assess developers on productivity, in a way, managers take developers away from social concerns and lead them only to technical and financial ones. One example is when developers are forced to do repetitive or redundant work as a consequence of bad planning. Repetitive and redundant work is a result of managers not understanding the complexities of target organisations and of playing trial and error games.

7.5 Social Issues in current development practices

Current software development practices in the context of this thesis means three things: first, the formal, textbook methodologies, second, the informal but organised practices carried out by developers and third, the conflicts and differences between formal and informal practices. Data shows that where formal methodologies are adopted, these are usually imposed by management. Developers might or might not agree with these practices but at the end will adapt them to the developers' style. Where no formal methodology is overtly adopted data shows that developers still acknowledge the use of a set of practices which make sense to them. These informal practices are a result of combining parts of formal methodologies, inventing new techniques and using developers'

common sense. From these two kinds of development practices I have identified common characteristics that either allow or hinder the identification of social issues in complex organisations. These characteristics (extracted from the complete model shown in figure 7.5) are shown in figure 7.7.

Software Development Practices

Address Social Issues

- Use of common sense
- Observe users in their natural setting
- Evolve software with close eyes on users (feedback)
- Working with users
- Iterative and incremental strategies
- Expecting change
- Clarification of ambiguities
- Social software
  - recognises the group as the user
  - designed to encourage certain kinds of behaviour

Neglect Social Issues

- Focus on the technical side of software
- Lack of knowledge about ethnography or reluctance to observe users in their natural setting
- Developers' preference for remote work
- Focus on usability and not group-sability
- Intermediaries, who transfer the information from the target organization to developers.
- Most practices that address social issues are not popular practices among developers

Figure 7.7 Social issues in current development practices

From figure 7.5 we can see that the issue of common sense appears again as playing an important role at helping developers to identify social issues in business settings. Common sense helps developers to use their human and social sensors when they try to make sense of a complex organisation. However common sense in this context should be understood as the ability to discern how to deal with and understand a social situation and make the most of this knowledge to develop software. It also means to be able to see when a methodology (formal or informal) is not working and different measures should be taken. Common sense should be educated. The more exposure a developer has with adaptive and unpredictable business settings where heterogeneous groups of people work and interact the better moulded the developer's common sense will be.

Some current development practices help developers build their common sense. Most of these practices involve close contact with users and the users' working environments. Ethnographic related practices like observing users in their natural settings give developers first hand knowledge about how things are done in reality. Interviews with users let developers grasp users' views about their jobs and colleagues. Having users as part of the project team helps to strengthen the links between users and developers. It also helps to develop more trust and to open more channels of communication. This would help to tackle complexities in business organisations like discovering hidden agendas or covert practices among users. This was discussed in chapter 6, categories [Software development/ Characteristics/ Complexity] and [Software/ Characteristics/ Complexity]. Another aspect mentioned by some developers was that keeping a system of fluent feedback from users helped them to improve the quality of their software. User's feedback would feed developers with information about the actual use of software and would help developers to discover mismatches between developers' assumptions and reality.

Other aspects of development practices encourage developers to change their attitudes toward business organisations. For example iterative and incremental strategies are designed to allow developers to account for changes in requirements and correction of mistakes. These strategies were discussed in chapter 6 in category [Software Development/ Characteristics/ Incrementability]. My data shows that the source of mistakes is usually of human nature. For incrementalists (developers following incremental practices) users are ambiguous beings and the responsibility of such developers is to unravel the ambiguities in the users' requirements. Developers who work under the assumptions of traditional development methodologies usually find it difficult to accept that business organisations and people cannot be mechanised in predictable way and hence inflexible way (see chapter 6 [Methodologies/ Traditional]). This attitude leads developers to delays in their work and poor quality products.

Finally, Social Software is a new concept that has the potential to highlight social issues in the software development field. Social software was defined and explored in chapter 6, category [Social Software]. Social software is the latest step in the evolution of software used for communication. Social software is a new term and is not yet very well known among the development world. However, this new concept brings with it explicit concerns about human and social interactions and how they can be reflected in software. So far practices for developing social software are a improvised and a result of trial and error strategies. However from the data it was possible to extract two basic principles that may open the eyes of developers to social issues. The first principle is the recognition that the user of social software is a group and not only individuals. This acknowledgement of groups would allow developers to consider social interactions happening *between* people and how these interactions can be supported by software they develop. The second principle is that social software should be designed to encourage but not force certain kinds of behaviour. Software should suggest ways of doing things and let the user decide on the best way for him. This principle addresses the fact that decision makers in organisations act differently depending on the situation and that their behaviour is inherently unpredictable.

Aspects of current development practices that hinder the identification of social issues are mostly related to the technical orientation of the majority of methodologies in the field. A focus on only technical aspects is usually a consequence of basing software development methodologies on engineering ones. Other factors that encourage a technical focus are remote work and the use of intermediaries to transfer knowledge from target organisations to developers. Remote work is becoming more common nowadays as it saves the developers time and money and as it is more practically possible (see chapter 6, category [Developers/Remote Work]). Intermediaries are analysts, who are in contact with users and prepare functional specifications for programmers. Intermediaries exist

only where development tasks are divided and developers specialise in different areas. In other kinds of environments developers do analysis and programming. Not having direct contact with peers and users limits developers' mental models of their target organisations. In programmers' minds organisations become a set of technical requirements and diagrams that contain little if any indications of human or social issues. Where human or social issues are considered they are addressed only partially. The fieldwork data shows that human and social factors are considered only as far as individual usability is concerned. (See chapter 6, categories [Software/ Characteristics/ Usability] and [Social Software/ Usability].) Decision making work is done individually as well as in groups. (See the discussion about complex organisations in section 1.1 of chapter 1). Demands from modern organisations require that software be able to support social interactions among users. Software does not only serve individuals but a group who needs to communicate with each other to do business.

7.6 Social Issues in Complex Business Settings

The concept of complex organisations for this thesis was first explained in chapter 1 and further developed in chapter 2. To summarise, complex organisations are organisations with high degrees of human and social interactions, where decision makers act freely; individually or in groups. These characteristics make software development difficult. Working from this preunderstanding of complex organisations, I engaged in an online ethnography to see what were the perspectives of other developers. I found the aspects of complex organisations as shown in figure 7.8 (extracted from the complete model shown in figure 7.5) reflected in developers conversations. As mentioned before these perceptions are shaped by other social aspects within the development environment and the development methodologies the developers follow. Although developers do not acknowledge the aspects mentioned in figure 7.8 as social and important for software development they still are able to

perceive them. I have classified the aspects in issues emerging from the nature of complex organisations as perceived by developers, and issues that emerge from the relationship between the organisations (and users) and developers (and the software that they develop).

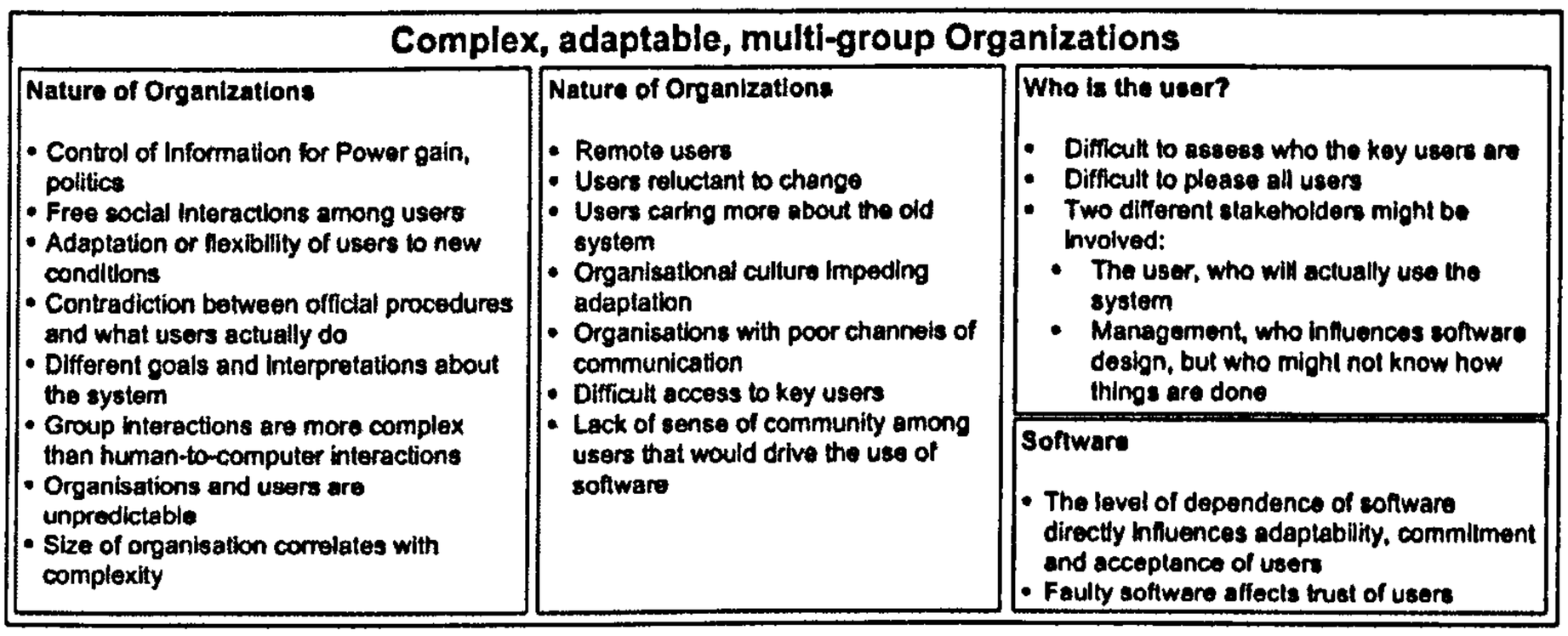


Figure 7.8 Social aspects of Complex Organisations

My data shows that developers experience social complexities in organisations through their interactions with users, their reading of documentation and the feedback they receive from use of software. Issues like unpredictability and adaptability in complex organisations, mentioned as part of organisations' nature are perceived and interpreted in different ways. Adaptability is the capability to change according to a new use or situation. This is perceived as a positive skill by some managers and users as they are required to respond quickly at changes in organisation and environment. This however is not always perceived as a positive skill by developers. The need for adaptability in software development (to follow the changes in business organisations) is also perceived as a consequence of users not being able to decide what to do and stick to one plan. In these kinds of situations users change their minds constantly at a pace that software and software development cannot follow. One cause of this perception could be hidden agendas within the organisation. These hidden agendas are difficult to discover and bring with them ambiguities and misunderstandings. Humans are able to adapt and work around these issues. However it is not that easy with software. Developers say that they need formal, overt procedures to

work from to develop and maintain good quality software. This contradiction is a consequence of trying to match two different "creatures": organisations with their unpredictable and adaptable nature and software which is predictable and does not adapt that easily. Some developers recognise that there is a contradiction between the needs of their users and the way software is designed. In extract 7.4 a developer suggests that users' needs should not be put away because of software limitations.

Extract 7.4

Thread Title: Are people complex?	**Date: 12th January 2005**	**Source: JoS**
"the world doesn't have to accomodate to the simplicity of your software development. Your software is not the center of the universe." Daniel Daranas		

Unpredictability is a consequence of organisations' adaptability. Some business processes cannot be standardised into predictable procedures because this would limit or restrict their adaptability. Free social interactions among users, especially decision makers cannot be standardised because of the nature of decision making. Users need freedom to act as they think, but need all the tools and information offered by software available first hand. Unpredictability as with adaptability, is most of the time perceived as a negative factor by developers; it is a consequence of not following standard procedures. Working with no standard procedures is thought by some to be an impossible task. Software can only be created for formal and predictable procedures. Another factor that increases the unpredictability of decision making processes is the need to work in groups. In these situations the number of interactions grows exponentially and processes become more unpredictable. Additionally, the inability of developers to see beyond human-to-computer interactions to group-to-computer interactions makes this factor more difficult for them to deal with.

Other social and human aspects within the complex organisation environment perceived by developers as problematic are the different goals and interpretations that management or users have about the system (business processes and software). The bigger the size of the organisation the bigger the number of goals and interpretations and the more problematic for developers to design software. Developers prefer clear requirements and specifications from users. Users are usually represented by one or more people who are in contact with developers. These representatives are in charge of transmitting the requirements from the whole group. However a risk this strategy brings is that the development could be limited to an incomplete view of the problem situation. On the other hand, considering all users' views about the problem situations becomes unmanageable sometimes because too many ambiguities arise. The clarification of ambiguities usually requires a personal or communication skill which developers themselves say they do not posses.

Although adaptability of organisations and users is perceived as making software development difficult by developers, the lack of adaptability in organisations is also perceived as problematic. Examples of lack of adaptability are: users reluctant to change and adapt to new software, users caring more about the old software and organisation's culture impeding this adaptation. Lack of adaptability makes the relationship between developers and their users difficult. Reluctance to change can make users hide their own needs, provide the wrong information or just not collaborate with developers. Developers could perceive reluctance to change as hostile and sometimes as a product of ignorance. In these cases communication is broken between reluctant users and developers and developers work on assumptions which will not necessarily fit the real needs of users.

Finally, issues arising from the relationship between the business organisation, developers and software can also affect the way developers see organisations. The most commented upon issue is the difficulty of

dealing with all stakeholders involved with the software. Developers express their concerns about not knowing how to please all stakeholders especially because they have different goals and interpretations of the system. It is also difficult for developers to establish who the relevant users are. Most of the time it is management who establishes the functionality of the software. However according to some developers most of the time, management, does not know how things are done in reality. For these developers the software should be developed for the people who actually do things. Another issue mentioned by developers is that the quality of the software that users already have influences the way the see new developments. Users of faulty software will usually reject new one or would not collaborate with the development of a new one. Additionally if users depend highly on a particular application it is more probable that they will be more collaborative with developers.

7.7 Chapter Summary

This chapter has presented a descriptive model of social issues in software development as seen by software developers. This is a theory based model that emerged from the data I collected in online forums. To create this model I used a theory building strategy that corresponds to the interpretive approach used through this study and the subjective nature of the phenomena studied. Sets of fieldwork data which accounted for particular instances of a phenomenon were categorised and generalised from their original context. The decontextualised data took the form of general concepts and propositions that represent more instances of the studied phenomena. These concepts, propositions and their relationships describe and explain my interpretations of the interpretations (Walsham, 1995, 78) of software developers about the nature of their jobs. I compared these concepts and grouped them according to the contexts shown in the model of social issues in software development. These contexts surround software development and contain social issues that affect it. The contexts are complex organisations; development

environments and development practices and each corresponds to one research question. In this way the three research questions identify social issues that influence software development.

The theory based model of social issues in software development intends to fill a gap in theory by addressing the implications that adaptive and unpredictable organisations have for software development. These implications are the human and social issues that are sometimes obscured to developers' eyes and the methodologies they use. The descriptive model suggests that software developers' perceptions of social issues in complex business organisations are influenced by the developers' background knowledge and experience. This background knowledge is built up from social issues within the development environment previously experienced and the development methodologies the developers use or know. This could be a recursive relationship as negative feedback from users of software will make developers reconsider their previous beliefs and look for new ones which yield better results. The social issues making up each context were extracted from the categories explored in chapter 6. From online conversations I perceived that these social issues affect developers and the software they produce. This model is not a deterministic one. It does not intend to predict the exact effects that social issues have on software (as in a cause-effect model). The model describes a set of social issues that *actually* affect developers work in different ways, as seen in online data. The intention of this descriptive model is to create awareness of social issues in the software development world as playing an important role in software quality and users' levels of acceptance.

In the next chapter, the conclusions, I will discuss the implications and limitations of this descriptive, theory based model. I will also propose further avenues for investigation and practice based on this model.

Chapter 8: Conclusions

Having presented the final outcome of this thesis in the form of a descriptive, theory based model, discussed in chapter 7, this chapter now concludes the thesis by providing a critical review of the whole research project. First in section 8.1 I summarise the research findings. Then in section 8.2 I discuss the contributions of this research. I distinguish between contributions to IS theory and contributions to methodology in IS research (as against methodology in Information Systems Development). Within section 8.2 I highlight the implications and limitations of this thesis' contributions. In the third section, 8.3, I provide an evaluation of this study by using the seven principles to asses interpretive research by Klein and Myers (1999), first introduced in section 4.1 of chapter 4. These principles were designed to guide interpretive researchers into understanding and following the philosophical and methodological underpinnings of the phenomenological and hermeneutical interpretive approaches and since those are the approach I have followed they are relevant for this thesis. In section 8.4, I suggest some potential opportunities for future research based on this thesis results. Finally, section 8.5 summaries this chapter.

8.1 Summary of Findings

This section presents a descriptive summary of the outcomes of this thesis. A critical analysis follows in the next two sections of this chapter (sections 8.2 and 8.3). The findings of this thesis are summarised in terms of the research aim and research questions. The research aim and questions were initially presented in section 1.2 of chapter 1 emerging from the research motivation presented in section 1.1 of the same chapter. The research aim and research questions were also presented in section 4.5 of chapter 4 as driving the research approach. Finally they were answered in chapter 7 with the descriptive, theory based model of social issues in software development.

The concern of this thesis was to study the human and social issues that surround software development. The initial interest emerged from my professional experience in information systems and software development (see section 1.1 of chapter 1). This professional experience, and afterwards a review of the relevant literature in IS, made me realise that there was a gap in software development research and practice. There seemed to be a lack of awareness and understanding of human and social issues within complex business organisations and software development environments by software developers. Although some research in IS and software development methodologies address human and social issues they do so by considering either business organisations or software development environments separately but do not connect both contexts together. As a consequence of the lack of consideration of human and social factors in either context, the quality of software is affected and the levels of software failure are increased (Cooper and Reimann, 2003, 5). The literature review (chapters 2 and 3) shows that there is a mismatch between the hard paradigms that the software developers use to work from, and the actual (human and social) nature of organisations (business organisations as discussed by Checkland and Holwell (1997) and software development environments as portrayed by Cockburn (2001)). Thus this thesis standpoint is that software development paradigms and practices should consider the human and social aspects of software development environments and business organisations as these both affect developers work. In order to consider human and social issues in software development, first, we have to identify these issues and find out the way in which they affect the work of developers. Thus the aim of this thesis is:

> To identify and explain how the social issues within the software development environment and the social aspects of complex organisations shape software and the process of software development.

Because the problem situation (explained in section 1.1 of chapter 1) exposes more than one area I designed three research questions each of which focuses on a different context. The first context is concerned with the human and social issues emerging from complex business organisations. The second and the third contexts focus on issues from the development environment. The issues are separated as aspects of individual developers and team groups, and the aspects of the development methods that they use. The reason for this separation is that apart from providing guidelines that determine the way developers work (in the development environment); development methods influence the way developers think of their target organisations and hence the human and social issues they can be aware of (which is the first context). The research questions are presented again in table 8.1 (repeated from table 1.1 in chapter 1):

Context	Question
Social Issues in Complex Business Settings	Q1: What are the human and social aspects of business environments that make software development complex and difficult?
Social Issues in Development Environment	Q2: What are the social aspects in the development environment that allow or hinder the identification of social issues in business settings during the development of software?
Social Issues considered in Development Practices	Q3: What aspects of current software development methods address or neglect social issues in business settings? What is social software and does it have a role to play in improving these practices?

Table 8.1 Research Questions

To answer the overarching aim and the specific research questions, data were collected during an online ethnography and classified into 72 categories and subcategories (the complete tree of categories is presented in appendix 2). These data categories provided a picture of software development being affected by human and social issues emerging from three different contexts: software development environment, software development practices and business organisations.

Issues under each category were discussed and examples from the online conversations were provided in chapter 6. In the next bullet points a brief explanation of the top 7 categories (which contain the other 65 subcategories) is provided:

- Software development: data under this category show that software development is still a practice in its infancy. Due to its human and social nature, software development varies from project to project. For developers, there are no two identical software development projects because software, users and development teams are always different. As a consequence, it is difficult for developers to agree on one particular approach for all situations.
- Software engineering: is a big influence on software developers' beliefs and software development practices. Software engineering is a "hard" paradigm which portrays organisations as structured and stable entities and software development as a predictable endeavour.
- Software: data under this category reveals the extent to which software developers are able to inscribe human and social issues within software. The ethnography shows that this is very limited to human-to-computer interaction issues which are considered only through usability guidelines.
- Developers: this category shows how software developers see themselves as professionals in their working environments. It reveals the most important human and social issues of the developers as individuals, as members of team projects and software development organisations. Broadly, developers usually emphasise their technical skills and neglect other aspects of their work. Software developers see themselves as having poor interpersonal and communication skills and in general, it seems, as unwilling to work in teams and interact with users.
- Methodologies: this category reveals software developers opinions of available software development methodologies. Data collected

focus on human and social issues within the different kinds of practices (and not on their technical aspects) and how these practices affect developers work. The most discussed software development methodologies (however not necessarily liked or used) were the agile methodologies. Agile methodologies focus on human and social aspects of the *development* environment and build on the skills and experience of developers.

- Business Issues: this shows how developers perceive their target organisation and the extent to which, they think, they should know it. Opinions about the extent to which software developers need to know their target organisations vary. Some developers see knowledge of their domain as being indispensable whereas others would neglect that aspect and focus only on programming (coding).
- Social software: is a kind of software which addresses their users as groups and focus on satisfying their needs for interacting and communicating. Although it is a relatively new concept, the data shows that software developers who know of it are more aware of the human and social issues of their users.

From the discussion of these categories a descriptive, theory based model was created. In section 7.2 of chapter 7 a theoretical framework is presented (see figure 7.3 of chapter 7) in which the model is situated as a complement to Orlikowski's (2000) model of enactment of technologies-in-practice. This framework presents software as a duality composed of a use and a design mode. The use mode is shown in terms of Orlikowski's (2000) structurational model of "enactment of technologies-in-practice". The use of software is represented as an ongoing process of reproduction between agency and structure. Human and social issues emerge from that relationship which have to be addressed in the design mode. The design mode of software is represented by the activity of software development and it is in this mode where my model of social issues in software development is situated. Therefore my model (which

focuses on the design mode) complements Orlikowski's model because this last one focuses only on the use mode of software.

The answer to the research aim takes the form of a descriptive, theory based model (presented in chapter 7) that shows the human and social issues that affect software developers and that explains how developers make sense of these issues. The model is repeated again in figure 8.1 (repeated from figure 7.5 in chapter 7). This descriptive, theory based model suggests that before addressing human and social aspects within their target organisations, developers create pictures of those organisations built from their preconceptions. These preconceptions are based on human and social issues from their past or current job experiences. For example, job settings, team arrangements and development practices that developers have used or known. Thus software developers' understandings of human and social issues within their target business organisations are demonstrably and directly influenced by their preconceptions of past experiences.

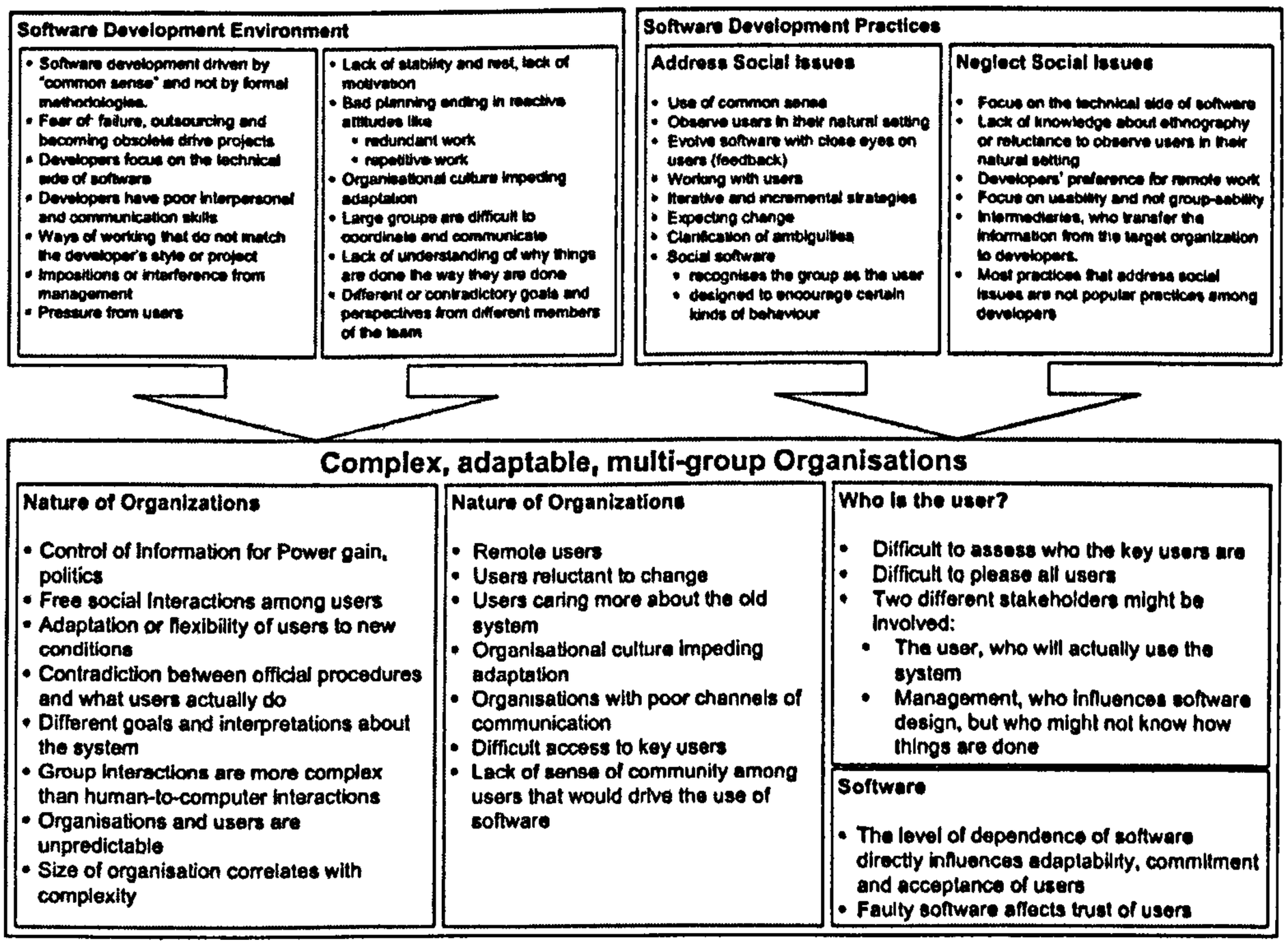


Figure 8.1 Model of Social Issues in Software Development

This model also enumerates human and social issues which were commented by software developers (during the ethnography). The human and social issues are grouped in three different contexts: software development environment, software development practices and complex organisations, presented in the following bullet points:

- **Human and Social issues in the software development environment:** the software development environment is usually a mixture of developers with different professional backgrounds, ideologies and working styles. From that mixture human and social issues emerge that shape the working style and arrangements of the group. Recursively, the styles and arrangements of the group shape the styles of the individual members. Developers might not share the same preconceptions of their target organisations. However, the way they are organised (in teams for example) forces them to agree on a way to do things. Usually it is the vision of the manager or the project leader which prevails over the others. This, in some cases, provokes resentment from some developers who think managers do not know their business and thus adversely affect the quality of developers' work. Also the concept of "common sense" emerges as a social construct within the development environment. Common sense is described by developers as the ability to discern what to do in different situations. In most cases developers will appreciate common sense as a valuable skill for software development (maybe more important than the knowledge of a software development methodology).
- **Human and social issues in current development practices:** the fieldwork data suggests that the knowledge of certain development practices provide developers with different perspectives of their own and target environments. For example, practices that compel developers to resolve

ambiguities in users' requirements propose the idea that their users and organisation may be uncertain of some issues regarding the software or might possess different (sometimes contradicting) points of view. Thus different kinds of practices make developers aware of the human nature of their users. Also, the concept of social software emerged as relevant. Although a new concept, social software definitions and design practices allow developers to think about human and social issues as emerging from the interactions of users. Social software also imposes the idea that these human and social issues need to be embedded in the software the developers produce. Among all the development methodologies and techniques discussed in this community of practice (virtual communities of software developers), "common sense" also arises as a development practice or as a way of deciding for oneself what practices to follow.

- **Human and social issues in complex business settings:** human and social issues in complex organisations are grouped in three areas: according to the nature of the organisations; issues regarding the identification of the user or users; and issues regarding the use of software in organisations. The perception of human and social issues in organisations by developers is influenced by the issues in the above two contexts. The nature of complex organisations is perceived by developers as ambiguous and influenced by politics. Ambiguity is a consequence of people having different perspectives and goals. Politics is perceived by developers as control of information for power gain. The more structured and traditional development paradigm the developer possesses the more ambiguous and politicised an organisation will be perceived. The relation between developers and their users is also a factor that affects developers' perception of their target organisations. Users are an important source of information. Depending on who is chosen by the organisation or the developers as user

representatives (managers or members of staff) and depending on how well they (developers and users) get along, developers will be able to understand their human and social aspects. This also depends on the development paradigm that the developers use and the team arrangement they have as the development paradigm encourages or not the inclusion of users in the development team. Finally, the level of dependence and the quality of the current software that the users use influences the way the users commit to the development project. The more dependent on the software the more participative the users will be. However, the lower the quality of current or previous instances of software the users have used the more sceptical and unwilling to collaborate these users will be.

The model of social issues in software development provides a descriptive account of issues that actually surround software development and explains in the form of a flow how these issues influence the developers' perception of complex organisations. However it is not (not yet) a structurational model as it does not uses structurational concepts (it is a descriptive model and does not theorise on the relationship between agency and structure). Further development in this area is proposed as a future research direction in section 8.4 of this chapter. In the next section the research strategy and contributions to knowledge made by this thesis, their implications and limitations are examined.

8.2 Research Contributions to Knowledge

Phillips and Pugh (2000) state that the research contributions section of a thesis should discuss the significance of the thesis' analysis and point out its limitations (Phillips and Pugh, 2000, 21). They also state that the thesis should provide an "original contribution to knowledge" (Phillips and

Pugh, 2000, 63) and offer a list of 15 ways of how a PhD can be original. In this section I will discuss the research contributions to theory and to research methodology of this thesis emphasising the way in which they are original (in the light of Phillips and Pugh (2000) list) and focusing on their implications and limitations to the field of IS.

Contributions to theory: This thesis has developed a theory based model whose objective is:

> To build a picture of Software Development for Complex Business Situations that describes and explains the process that software developers go through when they make sense of social aspects of the target organisation, the information system and the users that the software has to serve and when they assess and compare the technical aspects of software with the social issues within business settings.

From the list of Phillips and Pugh (2000) I can identify the following points as describing the originality of this thesis contribution to theory:

- Making a synthesis that hasn't been made before

The model puts together human and social aspects from software development and business environments and suggests a way in which these issues affect the thinking process of software developers. A review of the literature on information systems development and software development reveals that human and social issues are being covered in different ways. In the field of information systems development emphasis is paid to human and social aspects in business organisations and on developing ways to model and understand them (examples of this are SSM by Checkland and Holwell (1997) and Ethnographic approaches by Bentley et al. (1992)). Very little work is done on the software aspect of Information Systems. In the field of software development more work is put in its technical aspects than on its social aspects (for example the popular software engineering approach (Pfleeger (1991) and Pressman (2000))). Where the software development field pays attention to human

and social issues it does so by focusing on the characteristics of developers working alone or in teams but neglecting the aspects of the target organisations (for example: the agile approaches such as The Crystal Family by Cockburn (2000a) and Extreme Programming by Beck (1999)); or by concentrating on human-to-computer interaction issues as in software usability guidelines. Thus this thesis integrates (synthesises) two elements, social issues within business and software development environments, which have not been connected before, and creates a model of social issues in software development, which is an original contribution to theory.

- Looking at areas that people in the discipline haven't looked at before

This thesis' original contribution to theory is to consider that software development (and hence software as a product) is affected by social issues coming from the target organisation (complex organisation) and from the software development environment contexts. Another original contribution of this thesis is to model the thinking process that software developers go through when they try to make sense of the software they try to develop. The model states that developers' perceptions of their target organisations and the software they develop is affected by their preconceptions of human and social aspects of previous assignments and their knowledge about development practices. The descriptive, theory based model developed is an original picture of information systems development and software development as it focuses on the people *who do the software development job*. This has the potential to be expanded into more practical shapes which can be applied in actual software development work (see section 8.4 on future research for some ideas on how this could be achieved).

- Providing a single original observation in a competent piece of research

This thesis provides an original observation of the software development phenomenon as there is no other piece of research studying human and social aspects in software development through an online media (see next section for a discussion about contributions to research methodology).

Limitations of this model are, first, from the theoretical point of view, that in its current form it is not a structurational model and therefore does not complement totally the structuration model of Orlikowski's "enactment of technologies-in-practice". It hence needs more development to be compatible with the Giddens' original structurational principles. Second, the model considers only three contexts: business organisations, software development environment and software development practices as sources of the issues that affect developers work. There are other contexts which could be added to present a fuller picture of software development, such as the cultural differences among developers from different countries, software market, or new developments in technologies (this point is also taken as a future research possibility in section 8.4 of this chapter). This was picked up in part by data but not seemingly identified by developers as key themes. Third, the model does not entirely discriminate between types of environments that the software developers are working in. Data was used to build a holistic descriptive model, not intended (but could go on to be) a comparative tool. For example, developers could be working alone remotely or working in a software house as part of a larger team. I believe there may be some differences between the human and social issues to consider in either case. For example, developers working alone (i.e., one person project) would not have to deal with distractions or interruptions from their colleagues but will not get the opportunity of sharing ideas with them. This concern was also picked up in data but did not seem to be a "top" issue. Also, the model does not focus on structured processes like production processes

(more structured than decision-making and collaboration work) where people act less freely. Different human and social issues would need to be considered in the case of software development for operational applications such as the mentioned. Last, the relatively small number of people whose opinions I considered (though not an impediment for interpretive research, see section 4.6 of chapter 4 for a discussion of this approach) would limit the variety of issues I collected. More data collection focused on researching on new contexts and human and social issues for particular software development situations would enrich the present model (see section 8.4 on future research for some ideas on how this can be achieved).

Contributions to Research Methodology: this thesis has followed an interpretive, phenomenological and hermeneutical approach. Consistent with this an online ethnography of software developers was planned to collect relevant data and a theory building strategy was designed to analyse that data and create a descriptive, theory based model. From the list of ways to have an original contribution by Phillips and Pugh (2000) this thesis has contributed in the following ways:

- Carrying out a work that hasn't been done before: online ethnography on software developers (discussed in section 4.3 of chapter 4 and in chapter 5)
- Taking a particular technique and applying it in a new area: doing ethnography online, not to study the *communities* of developers themselves but to explore in the place where their work takes place, their beliefs about complex business organisations and the implications to their jobs

This thesis designed an original strategy to approach software developers and the topic of software development. Communities of practice of software developers are online and hence are places where actual software development work is done and relevant professional

conversations take place (a discussion about virtual communities of software developers is presented in section 4.2 of chapter 4). Thus an online ethnography is thought as an appropriate method to collect data. Also, the theory building strategy, although based on writings by Walsham (1993) and Tesch (1990), was carried out in an original (unique) way. Because in the interpretive paradigm there are no detailed guidelines on how to perform data collection and analysis I designed my own way which is explained in sections 4.4 and 4.6 of chapter 4.

Some limitations of the research strategy would include that the online ethnography does not offer the means to know the real identities of the participants and to verify the veracity of their statements. This is discussed in section 4.2.2, pragmatic characteristics of CMC, of chapter 4. Also the online ethnography does not allow me to explore background information on software developer's lives to be able to "'read' the social world behind the words of the actors" (Klein and Myers, 1999, 78), that is, behind the online media.

8.3 Reflections on the research: Evaluation using Klein and Myers' Seven principles for assessing interpretive research

The concern of this thesis was to study the human and social aspects of software development (see section 1.2 of chapter 1 for a discussion of the aim and research questions of this thesis, and chapters 2 and 3 for a literature review of related aspects). As the understanding of human and social issues involves and requires a high level of subjectivity from the researcher I considered an interpretive approach to be appropriate (see section 4.1 for an in-depth discussion of this thesis' approach). The interpretive approach allows the researcher to "understand the fundamental nature of the social world at the level of subjective experience" (Burrell and Morgan, 1979, 28). The interpretive approach is

aligned with this thesis' "idealist" ontological belief that organisations and information systems are "fundamentally, social rather than technical systems" (Hirschheim, 1985, 13) and that software development is a human and social activity (Weinberg, 1971). Following this paradigm I designed an online ethnography of software developers (see discussion in section 4.3 of chapter 4) through which I collected data about how software developers perceive and understand the human and social issues that surround their work. Being an immersive method the online ethnography allowed me to enquire in-depth into software developers' actions and beliefs. Following that ethnography I carried out a data analysis aimed at theory building. The outcome of this data analysis was a theory based model (presented in chapter 7) which describes the process that developers go through when they make sense of complex business situations and identifies the human and social issues that affect this process.

To assess the presented research process I chose to use a set of principles created by Klein and Myers (1999) (see section 4.1 of chapter 4 for a discussion of these principles). Appendix 8 of this thesis contains a summary of the reflections made using Klein and Myers (1999) principles. As seen in this evaluation, the process followed in this thesis fulfils the criteria designed for IS interpretive research. For instance, when reflecting on the conduct of the fieldwork, data analysis and formulation of the final model, it has been found that they satisfy these criteria. The accomplishment of these 7 principles and their relationships shows that this thesis is valid, reliable and credible effort consistent with interpretive research in IS. See also section 4.1 for a discussion of validity, reliability and credibility of this thesis.

8.4 Future Research

In this penultimate section I present a set of ideas for future research directions based on (the gaps left by and the new ideas generated by) the

results of this thesis. The first two proposals are concerned with further developing of the model of social issues in software development by using structuration theory. The third proposal is concerned with the transferability and confirmability of the results of this thesis (these concepts were introduced in section 4.1, Criteria for guiding and assessing the quality of interpretive research, of chapter 4) and the fourth one is concerned with the development of practical guidelines for software developers based on this thesis' model.

a. Further development of the model of social issues in software development

In section 7.2 of chapter 7 I introduced a theoretical framework for software development in which the descriptive, theory based model of social issues developed in this thesis is situated. The framework is shown again in figure 8.2. This framework presents software as a duality comprised of use and design mode. The use mode is represented by Orlikowski's model of "enactment of technologies-in-practice" (Orlikowski, 2000). The model of enactment of technologies-in-practice explains how users of technology (software) enact social structures when they use that technology and how they draw on those structures to use technology. The framework shows that the enactment of technologies-in-practice happens within complex organisations. Complex organisations are those organisations whose human and social aspects are difficult to understand and address by software developers. These organisations contain decision-making, workflow, collaborative work and similar processes whose behaviour is inherently unpredictable and adaptable, and in which work is performed by individuals and multi-groups with intricate social interactions. The design mode of the framework is comprised of issues happening within the software development environment and *influenced by* the issues represented by the model of enactment of technologies-in-practice (right hand side in figure 8.2). The descriptive model developed within this thesis is situated within the design mode of this framework and addresses human and social issues that affect software developers.

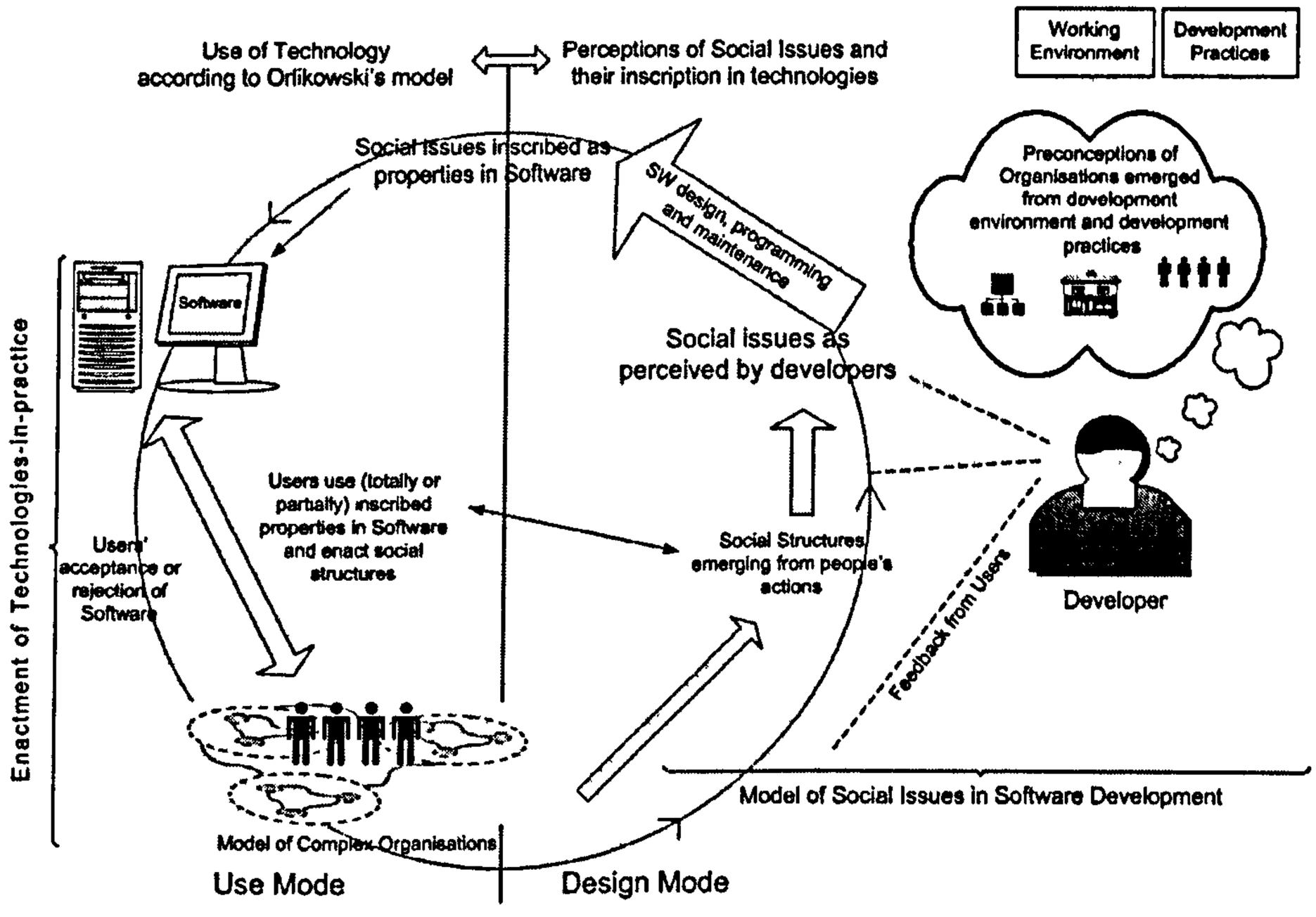


Figure 8.2 Framework of Software Development

The model of social issues in software development (shown in chapter 7) presents in a descriptive way the different human and social factors within three different contexts (software development environment, software development methodology and complex business organisations) that affect the work of software developers. It also explains how developers make sense of complex business situations (section 7.3 of chapter 7 contains a discussion of each aspect of this model). However, in its current shape (a descriptive model) it does not correspond to an extension of the structuration model of Orlikowski (2000) and does not truly fill the gap left by her model when it focuses only on structuration within organisations and users of technology. Thus a future research direction is to further develop the model by using structuration concepts particularly those corresponding to the enactment of technologies-in-practice. Figure 8.3 presents an idea of how this development could be carried out. In structuration terms, the model of social issues in software development suggests that the development of software (agency in structuration terms, presented as (A) in figure 8.3) depends on

preconceptions present in current structures within the development environment (presented as (B) in figure 8.3) and the actual process of structuration of technologies-in-practice within business organisations ((C) in figure 8.3).

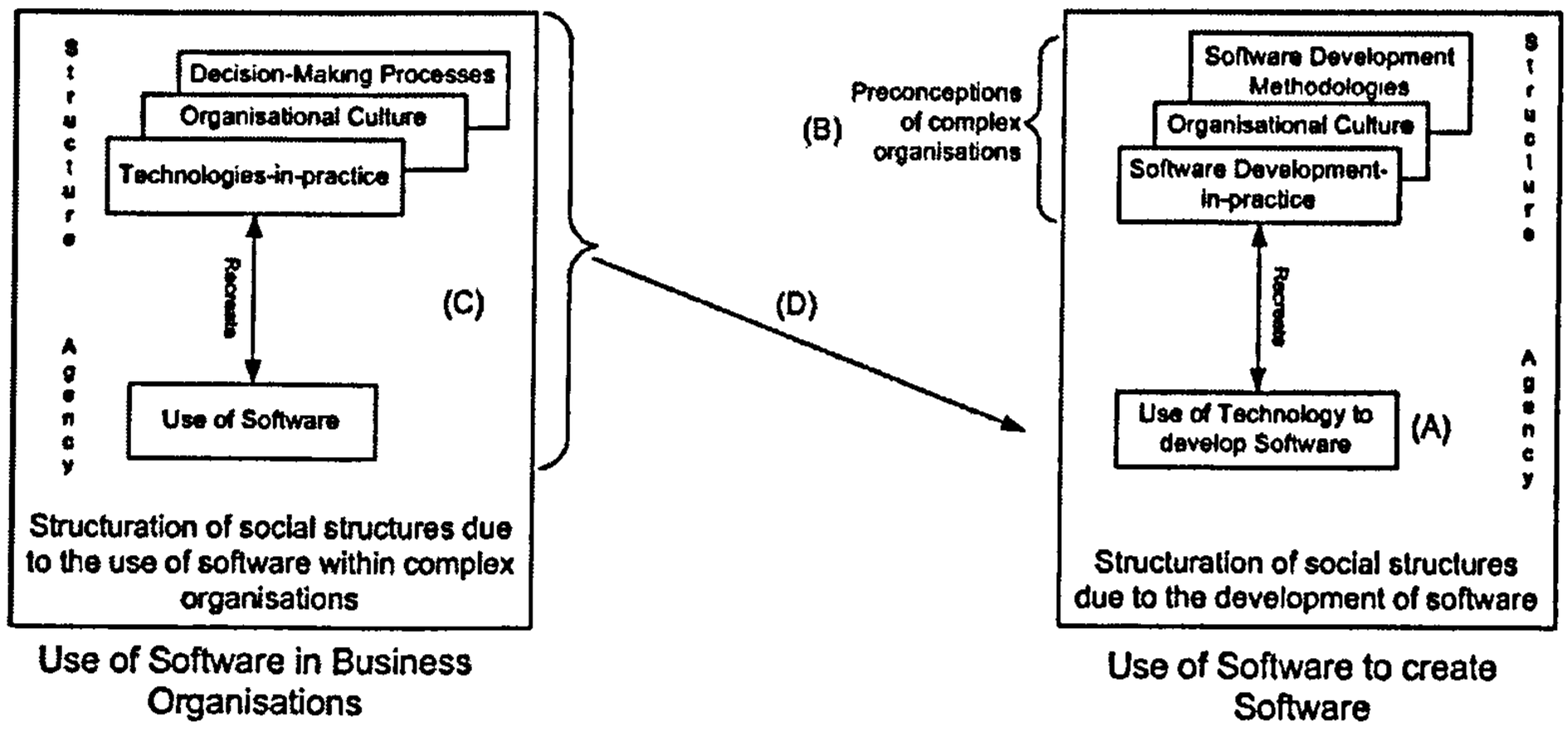


Figure 8.3 Development of the Model of Social Issues in Software Development using structuration concepts

The theory of structuration and in particular Orlikowski's model of enactment of technologies-in-practice could be replicated to explain the process of structuration within the development environment (box on the right in figure 8.3). However, more research is needed to explore the nature of the relationship between structuration in the development environment and structuration in business organisations (presented as (D) in figure 8.2). Thus the first proposal for future research is:

1. To use structuration theory and in particular the model of enactment of technologies-in-practice to explore the nature of the software development activity as it is influenced by structures within the development environment and structuration within business organisations

Additionally, it would be valuable to explore the nature and role of software as the product of the social process of software development. Orlikowski's (1992) Structurational Model of IT (discussed in section 2.2.4

of chapter 2) could be a starting point for this proposal. By applying the concepts of structuration of IT developed by Orlikowski (1992) to software, software could be thought of as constituting and being constituted by human agency. That is, software is "shaped by and shapes human action" (Jones et al., 2004, 318). Figure 8.4 presents an outline of this. As the descriptive, theory based model of social issues in software development suggests, software development affects the human and social aspects within the software development environment and the business organisations. Similarly it could be said that software is shaped by the human and social aspects within the same two environments. So the current descriptive model appears to be consistent with this proposal, but would require further exploration.

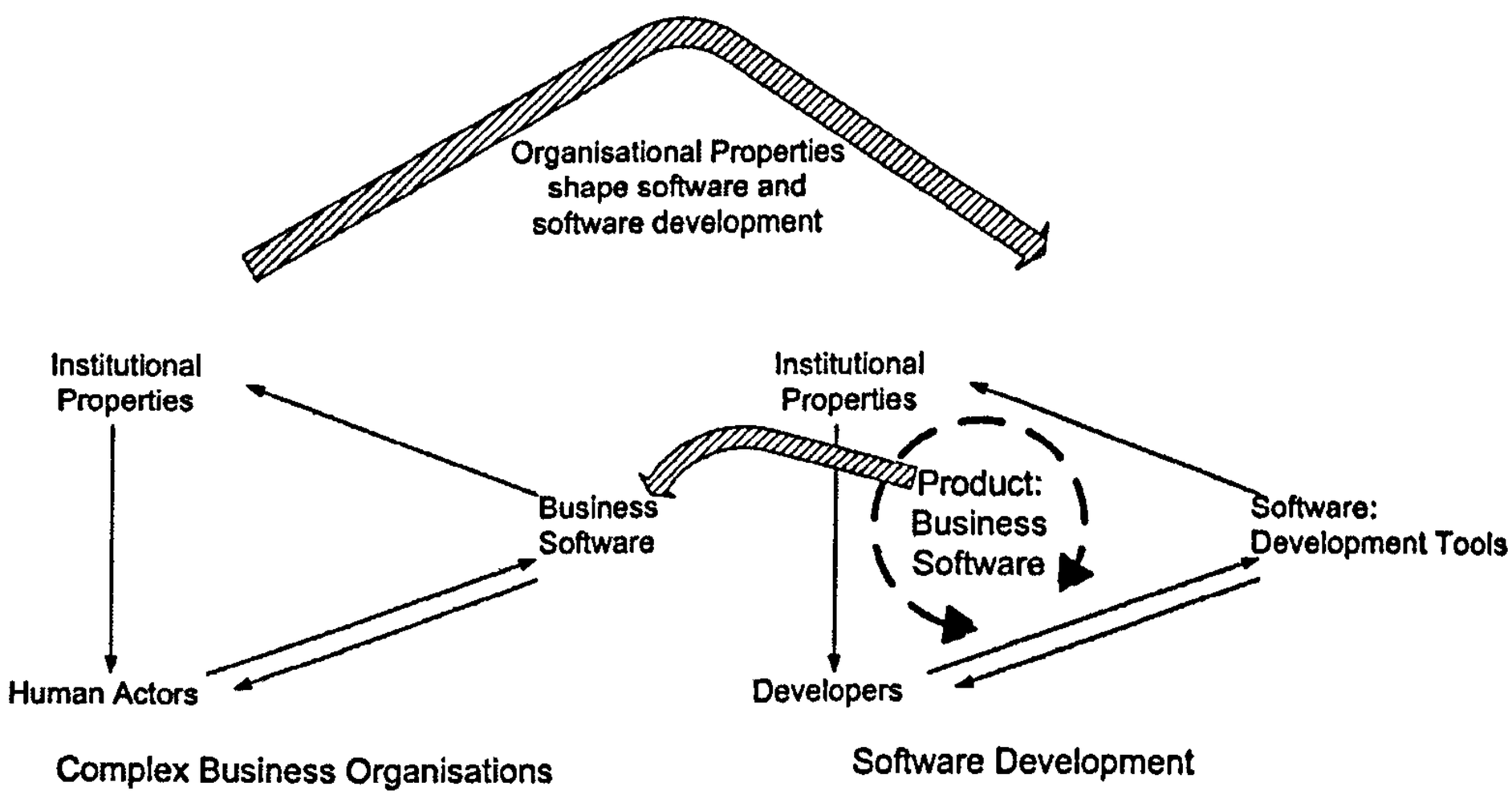


Figure 8.4 Structuration of Software

Figure 8.4 depicts the software development environment (right hand side) and the organisational context (left hand side), both of which use software, as containing structuration processes as originally conceived by Orlikowski (1992) (see figure 2.3 in chapter 2). Orlikowski's original model concentrates on the organisational environment (left hand side in figure 8.4) to explore the nature of technology. However this thesis proposes to look at both, software development environment and the business organisations, together to explore the nature of software. The descriptive

model has already done so but not by using structuration concepts. Thus the second proposal for future research is:

2. To use structuration theory to explore the nature of software as a result of processes of structuration within the development and business environments

b. Confirm and transfer the results of this thesis into other settings

In section 4.1 of chapter 4 I explored a set of criteria to assess the quality of interpretive research. Within Myers and Klein's criteria, I want to highlight the concepts of transferability and confirmability at this point. Confirmability tests the extent to which any research results can be confirmed by another similar study (Remenyi et al., 1998, 115). Transferability is the extent to which the researcher can generalise the findings to another context (Marshall and Rossman, 1989, 145). Confirming and transferring the results of this thesis will therefore enrich them and will provide more scope for their application. Thus the third proposal for future research is:

3. Carry out more research to confirm and transfer the results of this thesis

An example of this could be to use a different data collection method than online ethnography. For example, by structuring in-depth interviews of software developers in software houses and/or in-house internal developers the different and comparative aspects of the model of social issues in software development could be developed further. Also, by broadening the scope of software to not only decision-making, collaboration, workflow and similar applications more human and social issues could be brought to light.

c. Development of the practical side of the model of social issues in Software Development

A fourth proposal for future research would be to operationalise the outcomes of this thesis. That is, to translate these theoretical findings into a format that can be understood and put into practice by software developers. A theoretical model and a theory have limited use if they cannot easily be applied in the field. The practical side of this research's model could take any shape, from guidelines to methodologies. The fourth future research proposal is:

4. Use the model of social issues in software development or its further development with structuration concepts to create guidelines or methodologies for software development

The purpose of this practical development would be improve the awareness of human and social issues among software developers. By carrying out practices which are built upon the human and social aspects that affect their job, developers would improve the quality of the software they produce.

8.5 Chapter Summary

This last chapter of the thesis has exposed its contributions to knowledge. I have provided a summary of the findings, that is, a short explanation of the descriptive, theory based model developed within this thesis. Also I have discussed the implications and limitations of this model and of the research approach used. This thesis' contribution to theory is a descriptive, theory based model of social issues in software development. Also, this thesis has contributed to IS research methodology by designing an online ethnography consistent with the interpretive, phenomenological and hermeneutical approach. A short but critical reflection on the research strategy was also provided so as to discuss the validity,

reliability and credibility of the thesis. Finally, directions for future research where highlighted in the areas of further development of the descriptive model of social issues in software development by using structuration theory; by providing practical guidelines for software developers and by carrying out future research to confirm and transfer the results.

APPENDICES

Appendix 1: Summary of a typical day of a software developer.

Thread title: Oral Documentation - Not Worth the Paper It's Written On	Date: 10th June 2004	Source: Hacknot

"Scene 1 - a cubicle. Ed is slouched in an office chair staring forlornly at the screen in front of him. Except for the occasional insouciant jab at his keyboard, he gives the appearance of being comatose..

The day begins with my desire to extend the functionality of a legacy application, approximately 600K lines of code. I need to locate that portion of the code responsible for performing function X, so that I can insert function Y just after it. I go looking for function X amongst the code. I can't find it. In fact, I started looking for it sometime yesterday, and haven't found it yet. I check the folder marked "docs", to find it contains only a single `README.txt` file, the sole contents of which is the teaser "This directory will contain the docs" - apparently the dying message of a long extinct group of developers whose brains exploded before being able to make good on their promise. I find a piece of code that looks like its in the same ballpark as the code I'm looking for, and examine the revision history of the file it is in, to find that it has principally been developed by "Bob". I must find Bob. I need to find Bob. Bob will know where function X is. Here is my first problem. I cannot contact Bob directly, because I am but a lowly contractor. Bob is a valuable and in-demand member of my client's staff, and I can't just go up to him and steal his valuable time. There's a chain of command to be observed here! I must lodge a request with my manager to see Bob, who will forward that request to a liaison officer, who will forward that request to Bob's manager, who will then cue it up with Bob. If he's not too busy.

Scene 2 - a meeting room. Ed sits opposite a brown-skinned man wearing a turban.

The next day, I get to meet Bob. He can only spare 15 minutes to talk to me, because he's busy preparing for the next release of some whiz-bang new pile of crud. It's at this point that I discover that Bob's real name is "Sharmati Sanyuktananda", but everyone just calls him "Bob" for short. Bob is Indian. Bob's formal exposure to English was limited to the 15 minutes he spent reading "Miffy Learns English" while waiting in line at Immigration for his visa to be processed.
I try and talk with Bob, but it is like talking with Dr Seuss. At the end of 15 minutes, I have learnt almost nothing from him, and he keeps repeating something about public transport, which seems to have no relevance. His final word is "Sue", who I know is another member of the client's staff. So I contact my manager to organize some time with Sue.

Scene 3 - a meeting room. Ed sits opposite a nerdish looking woman wearing glasses with a very strong prescription.

Next day, I discover, to my significant relief, that Sue speaks English quite well. Unfortunately, her memory is a little hazy on the bit of code I'm asking her about. She remembers dealing with it about a year ago, but there's been a lot of water under the bridge since then. AT this point, I am beginning to consider tying weights around my feet and jumping off that bridge. She can't tell me where functionality X is, but she's pretty sure it isn't where I'm looking. "Have you tried asking John?", she queries. So I contact my manager and request a

meeting with another client staff member, John.

Scene 4 - a meeting room. Ed sits opposite a cool dude with sideburns and shoulder length hair.

Next day, John is disarmingly candid about the code I'm dealing with. "Oh yeah, I remember this crap", he begins. "We wrote that it in about a week, sometime last year, when we were up against the wall. It's absolute rubbish." "No kidding", I think. John is my guardian angel - he knows that function X got ripped out at the last moment, so they could meet their deadline. But then they put it back in a bit later, when things slowed down, and it's kept in a different module in the version control system. Which one? "You'll have to ask Declan", says John in a matter of fact way. I ask my manager to queue up some time with Declan.

Scene 5 - a cubicle. Ed is slouched in an office chair, browsing the advertisements on an employment web site.

My manager replies a few hours later, saying that Declan left the company a few months ago - maybe someone else knows. Have I tried asking Bob? "

Mr. Ed

Appendix 2: Tree of Categories, complete version

The research question(s) to which the category is related is shown in brackets.

1. SW-Dev (RQ1, RQ2, RQ3)
    - 1.1. Problems (RQ2)
    - 1.2. Goals (RQ2, RQ3)
    - 1.3. Process
        - 1.3.1.Coordination (RQ2)
        - 1.3.2. Improvement (RQ2)
        - 1.3.3. Division (RQ2)
        - 1.3.4. Communication (RQ2)
    - 1.4. Management (RQ2)
    - 1.5. Stages
        - 1.5.1. Planning (RQ2)
        - 1.5.2. Analysis (RQ2, RQ3)
        - 1.5.3. Design (RQ2, RQ3)
        - 1.5.4. Coding (RQ2)
        - 1.5.5. Testing (RQ2, RQ3)
        - 1.5.6. Documentation (RQ1, RQ2, RQ3)
    - 1.6. Characteristics
        - 1.6.1. Controllability (RQ1, RQ2, RQ3)
        - 1.6.2. Standardability (RQ1, RQ2, RQ3)
        - 1.6.3. Incrementability (RQ2, RQ3)
        - 1.6.4. Adaptability (RQ1, RQ2, RQ3)
        - 1.6.5. Predictability (RQ2)
        - 1.6.6. Levels of Abstraction (RQ2)
        - 1.6.7. Complexity (RQ1, RQ2)
        - 1.6.8. Productivity (RQ2)

2. SW-Eng (RQ2)

3. SW (RQ1, RQ2)
    - 2.1. Kinds of
        - 2.1.1. IS (RQ1)
    - 2.2. Characteristics
        - 2.2.1. Reusability (RQ2)
        - 2.2.2. Complexity (RQ1)
        - 2.2.3. Controllability (RQ2)
        - 2.2.4. Reliability (RQ1, RQ3)
        - 2.2.5. Flexibility (RQ1, RQ3)
        - 2.2.6. Adaptability (RQ1, RQ3)
        - 2.2.7. Usability (RQ1, RQ3)
        - 2.2.8. Maintainability (RQ3)
        - 2.2.9. Social Interface (RQ1, RQ3)

4. Developers
    - 4.1. Training (RQ2)
    - 4.2. Education (RQ2)
    - 4.3. Experience (RQ2)
    - 4.4. Responsibilities (RQ2)

- 4.5. Mentality (RQ2)
- 4.6. Definition (RQ2)
  - 4.6.1. Coder (RQ2)
  - 4.6.2. Analysts (RQ2)
- 4.7. Working habits (RQ2)
- 4.8. Remote work (RQ2)

5. Methodologies (RQ2, RQ3)
- 5.1. Traditional (RQ3)
- 5.2. OOD (RQ3)
- 5.3. Agile (RQ3)
- 5.4. Patterns (RQ3)
- 5.5. Prototyping (RQ3)
- 5.6. Free will (RQ2)
- 5.7. Modelling (RQ3)

6. Business Issues
- 6.1. Knowledge about business
  - 6.1.1. Domain knowledge (RQ1)
  - 6.1.2. Business rules (RQ1)
  - 6.1.3. Clients & Users (RQ1)
- 6.2. Business side (RQ1)

7. Social SW (RQ3)
- 7.1. Usability (RQ3)
- 7.2. Design (RQ1, RQ2, RQ3)
- 7.3. Communication (RQ1, RQ3)

8. Virtual Communities
- 8.1 Posting Habits
- 8.2 Motivations
- 8.3 Culture

Note: Data category [Virtual Communities] is not part of the analysis results. This category was created to collect data that describes the nature and behaviour of virtual communities of software developers for this research. See a discussion about Virtual Communities in section 4.2 of chapter 4.

Appendix 3: Analysis and Theory Building

This appendix shows part of the analysis and theory building process. Comments organised by categories were revised and connected to research questions. Propositions from comments were then refined and then organised by research question. The first extract shows an example of how comments from one category where linked to a research question. In this case the whole set of comments labelled as research question 2. The second extract shows the complete list of propositions grouped by research question.

1. Comments from [Methodologies-Free Will] connected to Research question

Comments on Category: [Methodologies-Free Will]
Definition: To choose what's best for a particular situation. To adapt the process to the needs. ***(Research Q2)***
1. There is no one SDLC to rule them all. Different organisations or projects need different methodologies. No methodologist can describe what to do in all circumstances. 2. Developers have to analyse their environment, type of software and duration of project and select a method that best serves their needs. "Take what works for you and throw out what you don't need". 3. Free Will= Pragmatic programming, methodologies are a good place to start but no place to stop. Developers should be open minded. Methodologies are good for starters but senior programmers have already developed common sense and familiarity with common practices, have the experience and thoughtfulness to decide what to use in any circumstance. 4. Free Will does not work well in teams if developers do not communicate. Analysts should share their ideas with the others to make things work smoothly. 5. Some developers claim that they do not use a methodology but use their brains and choose what have worked for them before. 6. Free Will practices "recognise that businesses are not a science and that there are not elegant laws like in physics whereby observation can be distilled into scientific fact."

2. List of Refined Propositions organised by Research Question. Left column shows the propositions and right column shows their category and the number of the comment of origin. In many cases propositions emerged from more than one category.

Question 1

What are the human and social aspects of business environments that make software development complex and difficult?

Propositions	**Category and comment of Origin**
Excessive or unnecessary control of information	[SW-dev-controllability 5] [SW-dev complexity 8] [SW-adaptability 4]
Free (social) interactions among users	
Informal, covert practices	[SW-dev-complexity 6]
Users who do not follow standard procedures	[SW-dev-standard 9]

Propositions	Category and comment of Origin
Users cannot be controlled as computers	[SW-dev-controllability 6] [SW flexibility 4]
Different people doing the same jobs but in different ways	[SW-dev-complexity 2,3]
Difficulty of knowing key users, people who control the information, people who possess the domain knowledge, and their relationships.	[SW-dev-controllability 7]
Help users to adapt to any software and make a good use of it even if it's not 100% fit to their needs.	[SW-dev-complex 1,4,5] [SW-complexity 2] [SW-flexibility 7] [SW-adaptability 6] [SW-social interface 1] [SocialSW-design 6]
Group interactions are more complex than human-to-computer interactions.	[SocialSW-design 7] [SocialSW-communic 5]
The unpredictable nature of users, organisations and the markets. Developers do not know how they will change in the future. Their work becomes reactive rather than proactive.	[SW-dev 3] [SW-adaptability 7]
Different (personal or official) goals from users or groups from the same organisations.	[SW-dev-complexity 7] [SocialSW-communic 2]
The uniqueness of every business situation.	[SW-dev 3] [SW 2]
Different interpretations about the same system	[Clients 3]
Users and managers having different ideas about their jobs.	[Clients 3]
Users who do not know what they need	[Clients 3]
Users who argue among themselves about the requirements.	[Clients 3]
Descriptions of jobs and requirements containing ambiguities	[Clients 3]
The impossibility to satisfy all users of software. There is always someone who complains and wants changes.	[Clients 9]
Adaptation of users, groups or organisations to new conditions causes changes in business settings (Flexibility?)	[SW-dev standard 8] [SW-dev adaptabil 5]
Size of organisation. The bigger the size	
The more control needed	[SW-dev-control 8] [SW-dev-standard 10]
The more complex structures maybe… the more covert channels of communication.	[SW-complexity 1]
Users reluctant to technology change, users who do not	[Clients 6]

Propositions	Category and comment of Origin
understand technology.	
Organisational culture impeding adaptation of technology. Organisations with heavy processes.	[SW-dev-adapta 9]
Pressure from users over time and budget and because they want to control the project.	[Business side 8] [Clients 12]
The level of dependence of users, groups and organisation on the software may decide on their commitment, adaptability and acceptance of it.	[SW-adaptab 1]
Users caring more about the old system.	[SW-adaptab 11]
Faulty, not suitable software being used by users and affecting their work.	[SW-reliability 1] [SW-adaptab 2] [SW-usability 2] [SocialSW design-6]
Contradiction between business rules (formal procedures) and the way the users actually do their jobs.	
Users' and Developers' domain knowledge isn't business rules	[DomainKN 1,4] [Clients 7]
Users' feelings about their jobs and software need to be considered.	[Clients 5]
Managers who act as key users and maybe are part of the project team but who do not know what their subordinates do (the people who are going to use the software).	[Domain KN 6]
Organisations where channels of communication are poor (software wouldn't work there)	[SocialSW communic 1]
Remote users who possess key domain knowledge (multi-organisation?) and who are difficult to access -> poor communication, poor feedback.	[Clients 1,11] [SocialSW design 4]
Lack of sense of community in a group or organisation that would drive the use of software. A community that grows around the use of software.	[SocialSW communic 2]
What is the relationship between complexity of business setting and complexity of software?	[SW-dev complexity 9]

Question 2

What are the social aspects in the development environment that allow or hinder the identification of social issues in business settings during the development of software?

Propositions	Category and comment of Origin
Different perspectives about the software development field from different developers who may participate in the same team.	[SW-Dev 1,2,3,4,5] [SW-Dev-Process-Communication 2] [SW-Dev-Management 3] [SW-Dev-Stages-Doc 7] [SW-Eng 1] [SW 1] [SW-Controllability 1] [SW-Flexibility 3] [Dev-Responsibilities 2,5] [Dev-Working Habits 8,9] [SW-Dev-Standard 1] [SW-Dev-Increment 4] [SW-Dev-Adaptab 3,8] [SW-Dev-Levels 1,2,3] [SW-Reusability 1,2,3,4]
Close minded attitudes toward software development hinder the ability to find the best strategy for a given situation.	
Software development being driven by common sense and not methodologies or standards.	[SW-Dev-Proc-Imprv 10]
Software development methodologies could be a constraint of creativity rather than a help.	[SW-Dev-Standard 4] [SW-Dev-Predict 3] [Methodologies 1]
Developers prefer learning and developing judgement and common sense which will help them more than following methodologies.	[SW-Dev-Standard 3] [Dev-Experience 2,3] [Methodologies-Free Will 2,5]
Developers can forget to do activities or processes if these are not controlled by a methodology.	[SW-Dev-Control 1] [Methodologies 6]
Different goals from different members of the team.	[SW-Dev-Goals 1,3]
Ways of working that do not match the developers' working style.	[Mentality 8]
Developers need to adapt to the standard procedures at the software house.	[SW-Dev- Proc-Improvements 9]
Ways of working that do not match the problem situation, users and target organisations.	[SW-Dev- Proc-Improvements 2] [SW-Dev-Increment 4] [Methodologies-Free Will 1] [SocialSW-Design 3]
There are no silver bullets.	
Having a reactive attitude rather than a proactive one.	[SW-Dev-Manag 1] [SW-Dev-Control 2] [SW-Dev-Standard 2]
Repetitive work as opposed to creative work, which is considered a waste of time.	[SW-Dev- Probl 1,2,6] [Dev-Mentality 9]

Propositions	Category and comment of Origin
Developers underestimating their jobs and getting bad results.	[SW-Dev-Problems 2]
Impositions or interference from management or users.	[SW-Dev-Problems 4,8] [SW-Dev-Manag 2]
Time and budget pressures	
Sudden change of teams	
Politics	[SW-Dev-Proc-Improv 3]
Assessment based on productivity and not quality.	[SW-Dev-Product 1]
Assessment based on working habits and not skills.	
Being treated as commodities.	[SW-Dev-Product 5]
Contradiction between time and budget as goals of projects and the unpredictable nature of Software development.	[SW-Dev-Predict 2]
Lack of stability causes probably fear of outsourcing.	[SW-Dev-Problems 5]
Lack of rest. Extra hours are considered a consequence of bad managed projects.	[SW-Dev-Problems 5] [SW-Dev-Productivity 4]
To do other people's work. It depends on how work was assigned and on the software procedures in the software house.	[SW-Dev-Problems 8]
Fear of failure drives projects. Sharing culpability with clients in case something fails.	[SW-Dev-Goals 4] [SW-Dev-Planning 2]
Fear of outsourcing	[Dev-Coder 4]
Fear of becoming obsolete	[Dev-Mentality 5]
Large groups are difficult to coordinate and communicate.	[SW-Dev-Proc-Coordination 2]
Organisational culture or developers impeding adaptation of new procedures.	[SW-Dev-Proc-Improv 4] [Dev-Working Habits 7]
Developers or PM's who do not know why things are done the way they are done, and who probably have distorted and incomplete views of the project.	[SW-Dev-Proc-Improv 5] [SW-Dev-Manag 5]
Frustration, lack of motivation and initiative of powerless developers, probably caused by	
Not having the power to introduce changes, or to make design decisions.	[SW-Dev-Proc-Improv-7,8,9]
Lack of support and reciprocity of PM's.	[Dev-Mentality 6]
Not having the appropriate technology to develop and test software.	[SW-Dev-Testing 12] [SW-Dev-Productivity 2]

Propositions	Category and comment of Origin
No training	[Dev-Training 1,2]
The nature of developers	
They have poor person and communication skills that do not help them to talk to colleagues and users and gather correct requirements.	[Methodologies 2] [Methodologies-Free Will 4] [SW-Dev-Standard 5,10] [Dev-Mentality 7] [Dev-Analysts 1] [Dev-Remote Work 1,2] [SW-Dev-Proc-Improv 6] [SW-Dev-Proc-Div 4] [SW-Dev-Proc-Comm 3,4,7] [SW-Dev-Analysis 4] [SW-Dev-Design 8] [Dev-Responsibilities 3]
They do not understand users' language.	
They need and work with non-ambiguous information.	[Dev-Definition 1]
They are individualistic, not particularly fitted for working in teams.	[Dev-Definition 1]
Prefer quick answers and do not like long meetings as these are considered a waste of time.	[SW-Dev-Problems 3] [SW-Dev-Proc-Comm 1] [Dev-Working Habits 1,2,3,6]
Some prefer electronic means of communication rather than face to face meetings.	[SW-Dev-Analysis 1]
Developers and PM's not giving importance to having appropriate documentation	
Reliance on pure oral communication only: maybe useful for first stages of development.	[SW-Dev-Proc-Comm 5]
Not having written documentation: useful for maintenance.	[SW-Dev-Problems 3] [SW-Dev-Proc-Div 1] [SW-dev-design 9] [SW-dev-document 5]
Having outdated documentation which is believed to be the updated one.	[SW-Dev-design 7]
Developers who focus only on the technical side of software and forget about or can't see the social side of organisations.	[SW-Dev-Analysis 4]
Developers who think of users as computers or software.	[SW-Dev-Control 6] [SW-Dev-Predict 1]
Not having users involved in the development process.	[SW-Dev-Analysis 3] [SW-Dev-Adaptab 2]
Improvised developers in the team. Improvisation in the field.	[SW-Dev-Problems 7] [Dev-Education 5]
Antagonism between, developers, analysts, PM's and consultants.	[SW-Dev-Manag 4,6] [Dev-Coder 5]
Influence of developers' preferences in socialisation on	[Dev-Mentality 2]

Propositions	Category and comment of Origin
the software they build.	[SocialSW-Design 2]

Question 3
What aspects of current software development methods address or neglect social issues in business settings? What is social software and does it have a role to play in improving these practices?

Propositions	Category and comment of Origin
To observe users of SW in their natural setting (office?) and learn about the social issues happening there.	[SocialSW-Design 5]
Usability - to focus on human to computer interaction	
Not enough	[Usability 1]
Prototyping	[Methodologies-Prototyping 2]
Evolve software with close eyes on users	
To consider feedback from users to develop the next increments.	[SW-Maintain 4] Methodologies-Agile 1]
Maintenance of software, incremental and iterative development and coding for flexibility	
to account for changes in organisation, due to social issues	
(maintenance) documentation is needed	[SW-Dev-Document 4]
coding standards are needed to ease maintenance	[SW-Dev-Standard 3]
Customers driving the requirements?	[SW-Dev-Analysis 3]
Developers acquiring domain knowledge, uncovering hidden practices and resolving ambiguities	[SW-Dev-Increment 3]
Getting involved with users	[SW-Analysis 4,5] [SW-Dev-Design 8] [SW-Dev-Control 7] [SW-Adaptab 2] [Methodologies 2] [SocialSW-Design 4]
Establishing rapport with users	
Learning user's language	
Being in users' shoes	
Clarification of ambiguities in requirements and specs.	[SW-Standard 1]
Prototyping	[SW-Dev-Analysis 12] [SW-Dev-Design 5] [Methodologies-Prototyping 2]
Standards for communication with clients	[SW-Dev-Document 1] [SW-Dev-Standard 5,10]
Developers and PM's good attitude of expecting change	[SW-Dev 3]

Propositions	Category and comment of Origin
at any time during the development process.	[SW-Dev-Increment 1,4,5] [SW-Dev-Adaptab 3,5,10]
Use of common sense and judgement to adapt methods to problem situation and users.	[SW-Dev-Adaptab 5]
Practices that force developers to foresee changes of organisation and software in the future.	[SW-adaptability 7]
Goal of software development project -> satisfaction of users.	[Methodologies-Agile 4]
The consideration that social software should not force but encourage certain kind of behaviour. Users should be free to use the software as they wish.	[SocialSW 1] [SW-dev-Standard 7,8] [SW-Social Interface 1]
The acknowledgement of a group as a user and of communication as the users' goal.	[SocialSW-Usability 1,2] [SocialSW-Communic 2] [SocialSW-Design 7]
Remote work could be a counterproductive strategy to learning about social interactions in a business setting.	[SW-Dev-Analysis 1]
Focus on the technical side of software only, neglecting the social side of organisations.	[SW-Dev-Testing 4] [SW-Dev-Document 3]

Appendix 4: Dictionary of Online Codes

Term	Definition
.NET	Microsoft® .NET is a set of Microsoft software technologies for connecting information, people, systems, and devices. It enables a high level of software integration through the use of Web services—small, discrete, building-block applications that connect to each other as well as to other, larger applications over the Internet.
3G	Third generation protocols. In mobile telephony, 3G support much higher data rates.
ADO	ActiveX Data Objects. A Microsoft technology that provides data access to any kind of data store.
AFAIK	As far as I know
API	Application Program Interface
Blog	online (personal) journal
BTW	By the Way
Chandler Project	Chandler is the combination personal information manager and platform that has enormous potential to help re-energize the stagnant PC desktop software marketplace
CMS	Content Management System
COM, COM+	Component Object Model, the MS-paradigm to connect components. MS has implemented the base technology for COM on the NT platform. Software AG has ported these on MVS and UNIX. A COM-object defines its interfaces. Components from different machines can be combined using DCOM . COM+ isan extension to Microsoft's Component Object Model (COM). COM+ builds on COM's integrated services and features, making it easier for developers to create and use software components in any language, using any tool. Delivered on the Microsoft Windows platform, COM+ is designed to preserve and extend current investments in COM. Applications currently using COM technology will work in the COM+ environment.
COTS	Commercial off-the-shelf, commercially available products that can be purchased and integrated with little or no customization, thus facilitating customer infrastructure expansion and reducing costs.
CSS	Cascading Style Sheets, style sheets when attached to documents describe how the document is displayed or printed, e.g. a CSS sheet is attached to an HTML document, to influence its layout when accessed via a browser. CSS supports cascading, i.e. a single document may use two or more style sheets that are than applied according to specified priorities (=cascade).
CVS	The Concurrent Version System, a source-code control system that

Term	Definition
	Project Builder can use to manage changes in source code over time and across multiple developers.
CYA	Cover your ass
DBC	Design by Contract
DEBIAN	A 100% Open Source Linux distribution http://www.debian.org
DUH	Colloquial comment on one's (or someone else's) lack of knowledge or brain power.
FAO	For the attention of
Flaming	insulting someone on a message board
Fork	ranching: the act of branching out or dividing into branches
FWIW	Acronym for "For What It's Worth"
GENTOO	Gentoo Linux is a high-performance ports-based Linux metadistribution for x86, PowerPC, UltraSparc and Alpha Processor systems.
GNOME	GNU Network Object Model Environment
GNU	GNU stands for GNU's not UNIX and is thus a recursive acronym
HCI	Human Computer Interface
IDE	Integrated Development Environment
IDEF	IDEF (Integrated Definition) is a group of modeling methods that can be used to describe operations in an enterprise. Originally developed for the manufacturing environment, IDEF methods have been adapted for wider use and for software development in general. Sixteen methods, from IDEF0 to IDEF14 (and including IDEF1X), are each designed to capture a particular type of information through modeling processes. IDEF methods are used to create graphical representations of various systems, analyze the model, create a model of a desired version of the system, and to aid in the transition from one to the other. IDEF is used along with gap analysis.
IIRC	If I remember correctly
IIS	Internet Information Server.
IMHO	In My Humble Opinion
IPO	(Initial Public Offering) is the first time a company is selling stock to the public.
ISV	Independent Software Vendor.
JDE	Java Development Environment
JFDI	Just F***ing Do It

Term	Definition
LOL	Laughing Out Loud
Longhorn	Next version of Windows, codenamed "Longhorn"
Mantra	Sacred syllables or words. Sacred utterance.
MMORPG	Massively Multiplayer Online Role Playing Game
MONO	The Mono project is an open source effort sponsored by Novell to create a free implementation of the .Net Development Framework
MSDE	The Microsoft Data Engine (MSDE) makes SQL Server technology available in development contexts that were previously restricted to the Jet database engine.
MSDN	Microsoft Developer Network
MSMQ	Microsoft Message Queuing (MSMQ) technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Applications send messages to queues and read messages from queues.
NNTP	Network News Transfer Protocol, the protocol used to send, distribute, and retrieve USENET messages.
OP	Original Poster
OTHO	On The Other Hand
PHP	Self-referentially short for PHP: Hypertext Preprocessor, an open source, server-side, HTML embedded scripting language used to create dynamic Web pages.
PHPGroupware	phpGroupWare is a multi-user web-based suite written in PHP. It contains many applications, including a Calendar, email package, contact database and project manager. It also includes an extensive API for writing new applications.
Plone	An open source Content Management System (CMS).
Postmortem	A phase at the end of a software project during which project team members evaluate the project and learn lessons that can be applied to the next project. "Postmortem" also refers to the report created during the postmortem phase.
Python	Python is an interpreted, interactive, object-oriented programming language—comparable to Tcl, Perl, or Java.
QA	Quality Assurance
RDF	Resource Description Framework - a metadata standard
RSS	RDF Site Summary (RSS) is a lightweight multipurpose extensible metadata description and syndication format. RSS is an XML application, conforms to the W3C's RDF specification and is extensible via XML-namespace and/or RDF based modularization.

Term	Definition
Rutime	The period of time during which a program is executing. The environment in which a user runs a Panther application. The runtime environment is different from the development environment in that the editors and repositories cannot be accessed.
SAMBA	Samba is an open-source implementation of Microsoft's SMB/CIFS protocol for file and printer sharing. Samba lets Linux computer masquerade as Windows NT or 2000 servers, offer better performance and stability at a cheaper price.
SDK	Software Development Kit
SharePoint	Microsoft Collaboration Tool.
SOL	SH** Out Of Luck
Spamming	sending e-mail(s) to innapropriate newsgroups
Spoof	To counterfeit a software program in order to get access or information on a system illegally.
SPS	SharePoint Portal Server
SSH	Secure Shell (SSH), sometimes known as Secure Socket Shell, is a UNIX-based command interface and protocol for securely getting access to a remote computer.
STFU	Shorthand for Shut The F*** Up
STFW	Search The Fucking Web
TCP	Transmission Control Protocol. Together with Internet Protocol (IP), TCP is one of the core protocols underlying the Internet. The two protocols are usually referred to as a group, by the term "TCP/IP." TCP enables two computers to establish a connection and exchange information. TCP guarantees delivery of data and also guarantees that information packets will be delivered in the same order in which they were sent.
TDD	Test Driven Development
TrackBack	A system implemented by many blogging tools that allows a blogger to see who has seen the original post and has written another entry concerning it.
Troll	A purposely stupid, inflammatory, or downright wrong article (closely related to flamebait). Its purpose is to get people mad and make them look stupid and gullible.
TTFN	Ta Ta For Now
UAT	User Acceptance Testing
UDP	User Datagram Protocol, is one of the protocols for data transfer that is part of the TCP/IP suite of protocols. UDP is a stateless protocol in that UDP makes no provision for acknowledgment of packets received, i.e. it will not tell you if data was received at the destination.

Term	Definition
VBS	Visual Basic Script
VI	Vi is a very simple, yet powerful, editor. Unlike emacs, which is a "modeless" editor, vi uses "modes". That is, at any point in an editing session, you may be in insert mode, command mode, etc.
VoIP	Voice over Internet Protocol; the practice of packet based networks instead of the standard public switched telephone network to send voice data.
WIKI	A Wiki is a collaboratively-edited website which many people also view as an anarchistic publishing tool. The distinguishing feature of wikis is that they typically allow all users to edit any page, with full freedom to edit, change and delete the work of previous authors.
WSDL	Web Services Description Language. The standard format for describing a web service. Expressed in XML, a WSDL definition describes how to access a web service and what operations it will perform. Usually pronounced "whizz-dul" (to rhyme with 'whistle'), WSDL is seen (with SOAP and UDDI) as one of the three foundation standards of web services.
WSS	Windos SharePoint Server
YAGNI	You Aren't Going to Need It.
YASNS	Yet Another Social Network Service
YMMV	Your Mileage May Vary (or your experience could be different)
ZOPE	Zope is an open source web application server written in Python. Plone uses Zope

Appendix 5: Permission Request

Thread Title: Being Researched	Date: 16th July 2004	Source: JoS

"Being Researched
Yes, the PhD student again...

I've been reading this forum for 7 weeks now and I am very much engaged with it. To summarise, I am doing a PhD in Software Development from the point of view of management/business. I started some conversations last month with very interesting and helpful replies.
http://discuss.fogcreek.com/redirect.asp?http://discuss.fogcreek.com/joelonsoftware/default.asp?cmd=show&ixPost=144731&ixReplies=12
http://discuss.fogcreek.com/redirect.asp?http://discuss.fogcreek.com/joelonsoftware/default.asp?cmd=show&ixPost=147094&ixReplies=9
http://discuss.fogcreek.com/redirect.asp?http://discuss.fogcreek.com/joelonsoftware/default.asp?cmd=show&ixPost=150306&ixReplies=21

I do thank all of you who shared your opinions with me.

I have been lurking this and other forums to update myself with the current and fashionable issues in the development world. I believe that all the material posted provides a good opportunity for research.

As my PhD is within the social sciences I am focusing on people's opinions on issues- I.e. software development. From the discussions here, what I am trying to do is gathering some information that I would like to reflect on in my thesis. It is not only the replies to my posts that I would like to use but maybe information from other threads in which I don't participate.

I would like to ask you all and Joel if there is any impediment to do so, including your own opinions. I do not intend to make direct reference to any of you. If so, I will ask you first. I do this as I think it is ethical to do so.
What do you think?

Cecilia Loureiro-Koechlin
c.loureiro-koechlin@mgt.hull.ac.uk"

Cecilia Loureiro
Tuesday, July 06, 2004

"You want to refer to posts written on this message board but you don't want to credit the authors with the posts? Is this actually an academic project? "

Kalani
Tuesday, July 06, 2004

"Kalani,

Yes, it is an academic project. A PhD project to be precise.

Maybe my post wasn't that clear.

My thesis will contain the analysis and conclusions of discussions. It will classify and summarise topics. However, I would probably need to quote some of you.

I am happy to credit the authors of the posts provided the authors do not have any objections and give me permission to do so. "

Cecilia Loureiro

Tuesday, July 06, 2004

"Seems to me you should go ahead... if you want to quote someone, and they've left an email address to follow, ask them. If they left no email address, treat it as an anonymous quote. Done and done. "

Greg Hurlman
Tuesday, July 06, 2004

"I agree with Greg. As far as I know, there's no legal/ethical reason to have to ask either. I've always treated information available on the public Internet just like a work you'd find at the library (the MLA even has a 'formal' way to reference Internet resources).

Please let us know about your work when you've finished it too. It'd be interesting to read."

Kalani
Tuesday, July 06, 2004

"thank you for your comments and for your support. :)"

Cecilia

Cecilia Loureiro
Tuesday, July 06, 2004

"Cecilia, I'm kinda curious, as you seem to have it backwards - whether or not to *quote* someone's post on the 'net without permission may be debatable, but I'm curious about an academic institution where a PhD student thinks *citing* a quote used is optional?

Are there really a lot of theses that say "then this other guy said 'Managers should follow before they try to lead' and someone else replied 'not necessarily'..." ?

Philo"

Philo
Tuesday, July 06, 2004

"Even if the thesis doesn't quote people, if you make some kind of conclusion based on data culled from this site then you ought to cite the site (so that others can validate your conclusions). But I'm sure you're aware of that kind of thing. It's obvious even for trivial undergrad papers. "

Tuesday, July 06, 2004

"Ceclia, have you contacted your University ethics officer on this one?

I ask merely because in Australia you would need ethics clearance before going "into the wild" and there would be guidelines in place on these sorts of things.

(OTOH, you might have already done this, and if so, I apologise) "

Les C
Tuesday, July 06, 2004

"What was the hypothesis of the work? Did you get any observational/experimental results and which conclusions have you drawn? "

Just me (Sir to you)
Wednesday, July 07, 2004

"Thanks very much for your comments.

Mr. Philo, yes, citing is a common practice, especially in the social sciences. You may cite someone to support or justify a conclusion.

Mr. Blank, yes I am going to acknowledge JoS... as the source of the quotes and the articles posted (which I think are excellent quality)

Mr. Les C., in social sciences ethical issues are sometimes as important as the outcomes of research. If the research is not conducted ethically it may be subject to criticism by supervisors and examiners and may be invalidated. For this reasons PhD students take some modules on philosophy and ethics.

In my case, I think I am taking a risk as online research is a new practice... methodologically as well as ethically, so I have to do it carefully... however this will also give my work more originality and in my opinion more reliability... as I believe online forums and the internet in general are valid places to approach software developers. (Instead of, for example, going to a software house and interviewing IT people, don't you agree?)... I think the internet gives me the opportunity to talk to more people from different backgrounds and levels of experience.

Mr. Just Me (Sir to you) ... I don't have a hypothesis. That works more for quantitative research (where for example you can test assumptions in a lab). I have chosen to do qualitative research based on online observations... that is, online discussions with software developers... what I have defined is a problem situation (based on work experience) and a theoretical framework which at the moment is a bit wide but I will narrow it down with my participation in your forum. I am not trying to validate my framework but refine it with your discussions.

I am really glad you welcome my participation in this forum
thanks again

Cecilia "

Cecilia Loureiro
Wednesday, July 07, 2004

"What makes you think there are any real software developers here? :-) "
Just me (Sir to you)
Wednesday, July 07, 2004

"maybe it is the name of the forum: Joel on SOFTWARE that might appeal to developers...
or the content, there is very interesting stuff discussed here"
Cecilia Loureiro
Wednesday, July 07, 2004

"What aspect of software development are you most interested in for this project? Maybe we could point you to the most applicable web-log, if this one isn't it."
Kalani
Wednesday, July 07, 2004

"Generally you do not need to ask permission to quote published work (under guidelines of fair use, of course). In fact, there is the concept that you *should not* ask permission, because doing so weakens the concept of fair use.

And a board like this might well be considered published work, in a public forum."
mb
Wednesday, July 07, 2004

"There is a lot of debate on this issue. Some people like you would say that information on the web might be considered as published material whereas others would consider the same material as private (especially information in discussion forums... especially those where you have to sign in...). I am just trying be on the safe side by asking for permission."
Cecilia Loureiro
Thursday, July 08, 2004

"Those who can't do, teach. Those who can't teach, write theories."

--

Thursday, July 08, 2004

"Kalani,

I am studying software development for difficult, complex and ill-defined business environments. For example... situations where it is difficult to specify requirements..., situations where you have a big number of potential users who need to be consulted and who will use the system, but who have very different opinions about it... or situations that change rapidly (nowadays this is getting more common)... As organisations and their processes change and adapt to their environments so must the software (so how do we deal better with changes? during development and after). At the moment my hypothetical scenario is that of multi-organisations containing processes that need decision making, workflow, knowledge management, etc. I am also assuming that the software must adapt (or evolve) to changes in those processes; this doesn't mean changes in requirements but conditions for flexibility in applications so users are not trapped in them. I would probably have to change these assumptions (If they are not realistic (who would buy a system like that??!!)) but that will depend on the feedback from online discussions.

Reading this forum I have found some topics which would be helpful... like: development methodologies: waterfall, agile methodologies (XP, Scrum, etc), components, UML, distributed/centralised systems, abstraction layers, abstration/productivity/innovation, remote software work, but also discussions about how's your everyday life as developers, what you believe will be the future of development (btw I enjoyed a lot the API war related threads)...and stuff like that. :))) "

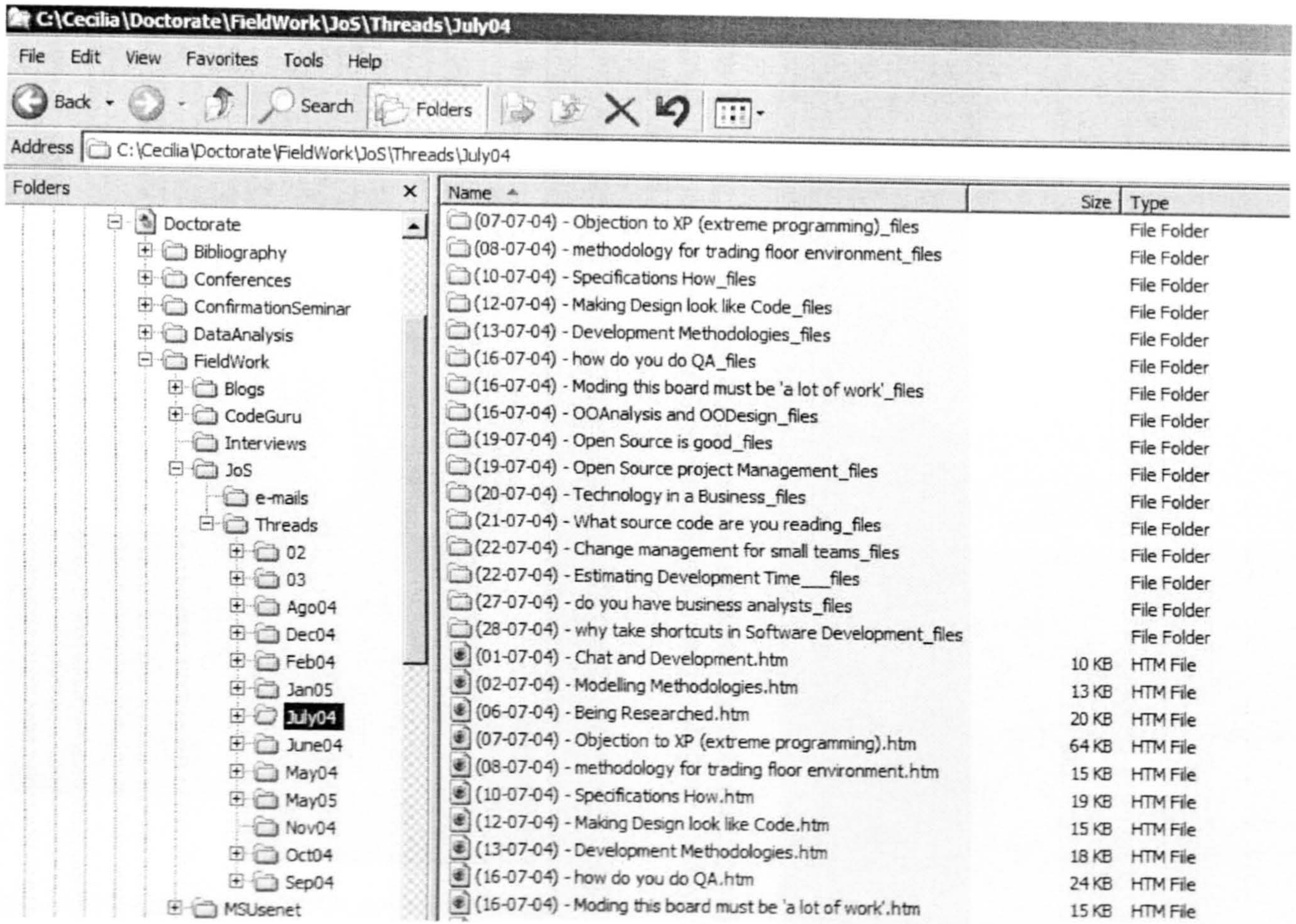
Cecilia Loureiro
Friday, July 09, 2004

Appendix 6: Online Sources used in this thesis

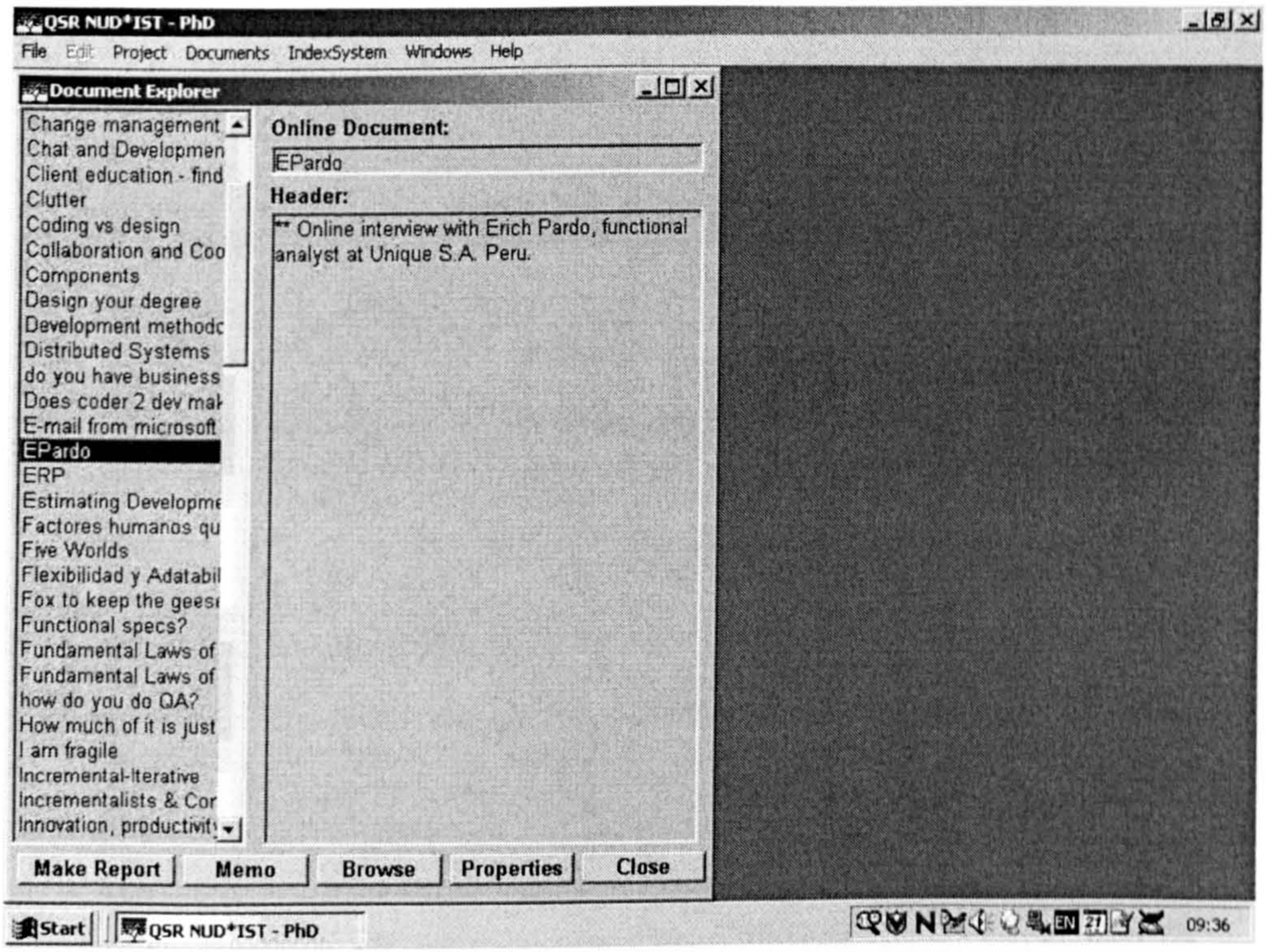
Name	URL
BookBlog	http://alevin.com/weblog/
Code Guru Forums	www.codeguru.com/forum
Eric Sink Web Log	http://software.ericsink.com/index.html
Hacknot	www.hacknot.info
JoS Discussion Groups	Old version: http://discuss.fogcreek.com/ threads up to September 2004 New version: http://discuss.joelonsoftware.com/ threads from September 2004
Life With Alacrity	www.lifewithalacrity.com/
Online interviews	MSN Messenger
Microsoft NewsGroups	http://communities2.microsoft.com/communities/newsgroups/en-us/default.aspx Select Spanish-Desarrollo-Desarrollo
Rands in Repose	www.randsinrepose.com/
Visual Basic Forums	www.vbforums.com
Wikipedia	www.wikipedia.org/ used to look for information on technical topics
WikiWikiWeb	http://c2.com/cgi/wiki?WikiWikiWeb used to look for information on technical topics

Appendix 7: Screenshots of the Organising System of the Data

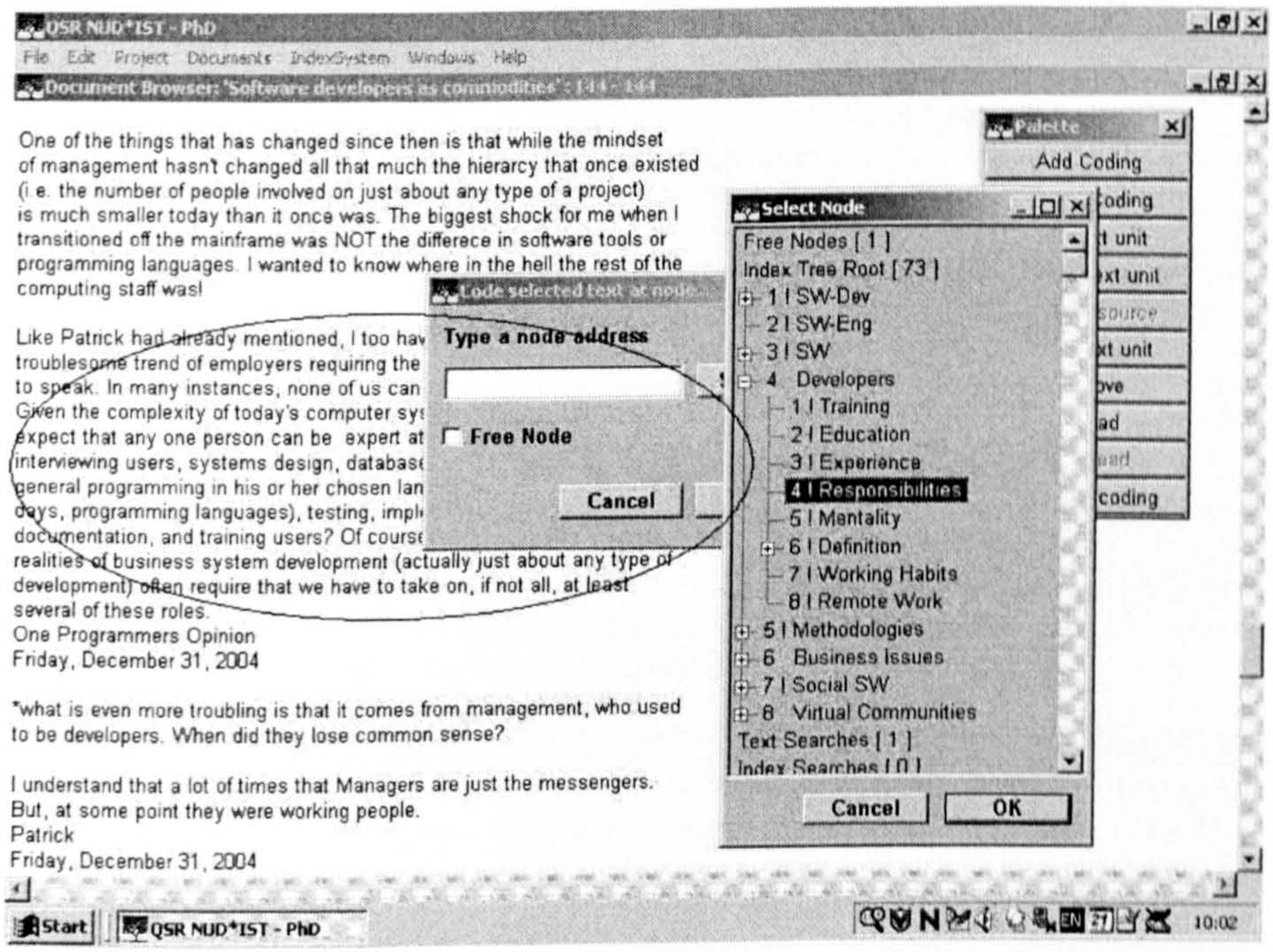
a) The following screenshot shows the fieldwork data files names and the folders they were organised in. These files are copies of the online conversation I stored in my computer. Each file was then entered in a NUD*IST project as shown in the second screenshot.



b) The next screenshot shows the NUD*IST project created for my thesis and the documents that it contained:



c) The following screenshot shows how a segment of text (encircled) is assigned to the [Developers/Responsibilities] data category coded as (4,4) in NUD*IST:



Appendix 8: Thesis Evaluation using Klein and Myers (1999) seven principles

1. The Fundamental Principle of the Hermeneutic Circle
This principle suggests that all human understanding is achieved by iterating between considering the interdependent meaning of parts and the whole that they form. This principle of human understanding is fundamental to all the other principles.
This thesis has followed an interpretive ethnographic approach which has given it a unique shape; by using a different approach the results would have been different. Therefore this thesis provides (only) one of the unlimited possible perspectives (a part) of software development (the whole). In section 8.4 of this chapter I propose a set of directions for future research. One of the proposals suggests a confirmation of this thesis results by carrying out more research by using different approaches which could complement the model of social issues in software development. Thus further research would provide with more different perspectives (parts) concerned with the process of software development (the whole). In section 4.3.3 of chapter 4 I discussed the process of interpretation of data used in this thesis. This process follows a phenomenological and hermeneutical perspective. The hermeneutical circle is presented as the "dialectic process" (Boland, 1985, 195) "between the understanding of the text as a whole and the interpretation of its parts" (Myers, 2003). I follow this approach during the data collection and analysis because it allows me to explore in-depth the beliefs of software developers about their jobs and complex business organisations. In chapter 5 I explained the practicalities of the ethnographic and data analysis process. This process is comprised of several iterations through which I gained insight into software developers' interpretations. I analysed the individual turns in online conversations as parts and the context of the complete conversations as the whole. Once coded each text segment was analysed as parts and the context of the data category as the whole. Each data category was analysed as parts and transformed into the final theory based model as the whole. During this process the relationship between each aspect and the whole model is drawn in. In chapter 6 I discussed the set of categories being the subcategories and the issues covered by each category the parts. I then put all the categories together in chapter 7 and develop a model which contains all the issues highlighted by the data categories (the whole). Additionally, my perception of this thesis' results is that the model of social issues in software development is a part of (an interpretation of) the process of software development as a whole. The mentioned descriptive, theory based model focuses only on human and social issues in software development and is not concerned with its technical aspects (unless those technical aspects affect or involve human and social issues).
2. The Principle of Contextualization
Requires critical reflection of the social and historical background of the research setting, so that the intended audience can see how the current situation under investigation emerged.
This thesis started from a personal experience and interest. The social and

historical background of this thesis is initially based on a personal venture. In fact this thesis is about me discovering my role as an IS and software developer and now as a PhD researcher. One of my principal aims was to understand my job (software development), the people I worked for (the users), and my role as a software developer (and any software developer). The literature review in chapters 2 and 3 document the social and historical background from the academic point of view (therefore is not only me having a background), which helped me to learn what other authors have said about related topics in IS and software development, and to develop a competent research proposal. Also, being an ethnography its data emerges from an openly declared social and historical context, that of actual software developers and the developed descriptive, theory based model draws upon this.

Chapter 1 of this thesis presented the motivation of this thesis in the form of a professional experience in information systems and software development. This chapter set the context of this research as being devoted to software development as experienced phenomena within the field of information systems and focused on complex business processes in which human and social issues complicate the work of software developers.

Chapters 2 and 3 provided a general understanding of human and social issues in organisations, information systems, software and software development. I focused on concepts that appeared to be relevant in the context of this research rather than on generic information systems or software development research. I did so because I found very little research on social issues in software development particularly ones covering both social issues in organisations and the development environment. In chapter 2 I was more interested in setting a theoretical background for software development that focuses on "soft" definitions of organisations and information systems. The concepts reviewed included those which address human and social issues within those contexts. In chapter 3 my aim was to provide a broad view of software and software development by exposing their human and social aspects which correspond to the view of organisations and information systems presented in chapter 2.

3. The Principle of Interaction Between the Researchers and the Subjects

Requires critical reflection on how the research materials (or "data") were socially constructed through the interaction between the researchers and participants.

As this is an interpretive study the findings are recognised to be the result of my interpretations of the interpretations of software developers about the human and social nature of their jobs. This is shown in chapter 4 where I present this thesis' research paradigm and in chapters 6 and 7 where I present the research findings and final outcome.

The online ethnography was a learning process for me. From the beginning of the fieldwork, when I looked for online forums of software developers, I learned how these communities where structured and the kinds of topics that where discussed. I also learned that in general, discussions about software development were mainly technical. However from the texts discussing technical issues, software developers revealed aspects of their working environment and their users' that expose the human and social issues that make their work complicated. This happened especially when discussions grew long enough to allow online participants to develop meaningful interactions and relationships. By witnessing and participating (and encouraging discussions to explicitly expose human and social aspects) in these interactions I was able to explore the human and social aspects of

their work.

Section 4.2 of chapter 4 describes virtual communities of software developers which is the field where I carried out the online ethnography. Virtual communities of software developers were chosen because they are "places" in which actual software development work is done and were relevant discussions take place. I defined virtual communities and created a classification of types of members and their online behaviour. I also discussed the linguistic and pragmatic characteristics of the online conversations and the implications for the data collection and analysis in this thesis. Additionally online communities were discussed in terms of the concept of communities of practice. "Communities of Practice" helps to explain how social practices, including language (and communication), are developed by online members when they participate in online activities.

In section 4.3 of chapter 4 I established my role in the online ethnography as "participant as observer" thus placing the participants and the researcher in context.

4. The Principle of Abstraction and Generalization

Requires relating the idiographic details revealed by the data interpretation through the application of principles one and two to theoretical, general concepts that describe the nature of human understanding and social action.

The data interpretation involved the discovery of themes which were initially defined in the context of the online conversations from which the data was extracted. Further analysis disconnected these themes from their original contexts converting them to the generic concepts and propositions. In section 4.4 of chapter 4 I presented the data analysis strategy carried out in this study which focuses on coding segments of data, connecting segments to common themes and then generalising and abstracting the themes from their original contexts.

Abstraction and generalization from the interpretive point of view does not rely on the representativeness of the sample with regard to a particular population, but on "the plausibility and cogency of the logical reasoning used in describing the results from the cases and drawing the conclusions from them" (Walsham, 1993, 15). In section 4.6 of chapter 4 I discussed this research's theory building strategy that follows the principles of abstraction and generalization.

In chapter 5 I presented the actual fieldwork I did which follows the above strategies. I displayed the process of abstraction and generalisation of data by providing examples of data categorisation and concept generation. By openly displaying this process the reader understands and verifies how raw data were transformed into a theory based model thus providing reliability to this thesis.

5. The Principle of Dialogical Reasoning

Requires sensitivity to possible contradictions between the theoretical preconceptions guiding the research design and actual findings ("the story which the data tell") with subsequent cycles of revision.

Doing this study has made me learn more about and reflect on the human and social nature of organisations, information systems and software development. The data I collected contain rich accounts of software developers' experiences, some of which are familiar to me. Having done a meticulous analysis of the data I was able to contrast the theoretical structured and predictable preconceptions present in some IS literature (in

which I believed) and the "facts" in the software development world as seen by software developers (shown throughout chapter 6).

In chapter 1 I presented a professional experience of software development in which I exposed my preconceptions of organisations, information systems and software development. At that time organisations and information systems for me presented similar characteristics as software. That is, in my mind they were structured, standardised and unambiguous.

The literature review, presented in chapters 2 and 3 and this thesis' findings presented in chapters 6 and 7 show a different picture of organisations, information systems and software development. This picture emphasises aspects such as ambiguity, adaptability and unpredictability which are consequences of their human and social aspects.

Also in section 4.1.2 of chapter 4 I exposed my personal values to inform the reader of some possible biases during the study.

6. The Principle of Multiple Interpretations

Requires sensitivity to possible differences in interpretations among the participants as are typically expressed in multiple narratives or stories of the same sequence of events under study. Similar to multiple witness accounts even if all tell it as they saw it.

In chapter 6 I presented the findings of this thesis organised by categories. The findings provided a picture of software development as understood by software developers. In this picture I exposed different interpretations of human and social issues by software developers, some of which were completely opposed. This was not a conflict between the subjects of study's interpretations and my interpretations but a confrontation (as I interpreted it) of conflicting viewpoints between the online participants. This does not mean that I did not play a role in the process, quite the opposite. I played a role by initiating and leading some online conversations.
Also I was the one who interpreted the online posts and decided which ones where relevant to the subject of study (as explained in chapter 5 of this thesis).

7. The Principle of Suspicion

Requires sensitivity to possible "biases" and systematic "distortions" in the narratives collected from the participants.

Although that I did not follow this principle exhaustively - Klein and Myers (1999) leave open the possibility that interpretive researchers may not follow this principle as there is "considerable disagreement about the extent to which social research can (or should) be critical (Deetz, 1996)" - in chapter 6 (research findings) I contrasted different and conflicting points of view by software developers. I did so not with the aim of discovering "false preconceptions" (Klein and Myers, 1999, 77) but to expose the legitimate different perspectives I found in the fieldwork.

The online ethnography I performed, though being an immersive method, allowed me only to explore in-depth software developers' views on their work. However it did not provide me with enough tools to judge the validity or veracity of their statements nor did it give me background information on software developer's lives to be able to "'read' the social world behind the words of the actors" (Klein and Myers, 1999, 78). (this is discussed in section 4.3.2 of chapter 4)

Appendix 9: Additional organisational definitions

a. Knowledge Management

This section explores some ideas within the field of Knowledge Management (KM). Although KM does not cover all the aspects of complex organisations it is useful to connect the concepts of information, knowledge and learning within organisations to the model of complex organisations, Handy's intelligent organisations and Drucker's knowledge workers.

Workers in complex organisations are involved in processes through which information flows. In these processes information is created, used and recorded. People use the knowledge generated from previous experiences to perform new tasks. This utilisation of knowledge creates even more knowledge which is used and recorded. In relation to this Coakes et al. (2004) define "process knowledge" as the "knowledge of how organisational processes are performed within their own task environment and also within the overall organisational internal and external environment." (Coakes et al., 2004, 119). Coakes et al. (2004) also note that process knowledge is comprised of fluid and sticky knowledge. Fluid knowledge is that knowledge that flows around the organisation by word of mouth or other ways of interpersonal communication (as in for example decision-making processes in complex organisations). Sticky knowledge is built from (knowledge workers) personal past experiences, training, skills and competences developed through interaction with the processes.

The concept of knowledge management can be related to the concepts of adaptation and evolution in organisations. Information and knowledge are the means through which organisations adapt to their environments and innovate their practices. Nonaka (1998) defines these kinds of organisations as "knowledge-creating". These organisations "consistently create new knowledge, disseminate it widely throughout the organisation,

and quickly embody it in new technologies and products" (Nonaka, 1998, 22). A complementary definition is given by Garvin (1998). Gavin uses the term "Learning Organisations" which are "skilled at creating, acquiring, and transferring knowledge, and at modifying its behaviour to reflect new knowledge and insights" (Garvin, 1998, 51). Both definitions coincide in affirming that organisations create and transmit knowledge continuously. The model of complex organisations could be described partially as a "learning organisation" as the extraction, creation and use of information is implicit in the model (see figure 1.2 of chapter 1). However the utilisation of knowledge is only one aspect reflected in the model of complex organisations. The activities performed by knowledge workers resulting in intricate working and social interactions which make organisations look in an unpredictable manner are not considered by the knowledge management field.

b. Complexity Theory

In section 1.1 of chapter 1 I mentioned that my colleagues and I found our target organisation seemingly chaotic. Contrary to what we were expecting, standardised procedures and predictable behaviour, the organisation kept on changing and was unpredictable. To understand more about (apparent) chaotic situations, in this section I am exploring complexity theory. Complexity theory is based on the theory of chaos which deals with complex natural phenomena. Chaos theory studies the patterns of behaviour with irregular characters and tries to classify them in infinite categories. Chaos is defined by this field as "an intricate mixture of order and disorder, regularity and irregularity" (Parker and Stacey, 1994, 11). Chaos provides a better explanation of how the world works than a more orderly perspective which used to prevail (Parker and Stacey, 1994, 7). Using ideas from chaos theory, complexity theorists define organisations as living systems interacting with each other in a nonlinear fashion and characterised by irregular patterns of behaviour that cannot be reduced in a simple way (Stacey et al., 2000, 17). In other words, complex systems are:

- Systems comprising a large numbers of individual agents.
- These agents interact with each other according to rules that organise the interaction between them at a local level.
- Agents endlessly repeat their interaction referring back to their rules, that is, interaction is nonlinear and this nonlinearity is expressed in the variety of rules across the large numbers of agents.
- Ongoing variety in the rules is generated by random mutation and cross-over replication.

Source: Stacey (2001, 71)

According to Parker and Stacey (1994) The characteristics of complex systems are:

- Non-linearity. Linear systems are the ones in which one action causes only one (predictable) effect. Non-linear systems are characterised by the uncertainty of the outcomes given a specific cause. Here one action can result in multiple effects.
- Negative and positive feedback. Feedback is a way of learning in organisations. In planned systems, negative feedback changes the output of a system so as to diminish the discrepancies between the intended and the actual outcomes. However, in nonlinear, complex systems, positive feedback instead of nullifying the deviations reinforces them in order to amplify the discrepancies and destabilising the system.
- Bounded instability: is an intermediate state caused by continuous change between stability and instability.
- Self-organisation. It is the process in which organisations evolve into chaos. During this process they produce unpredictable and complex forms of behaviour.

Source: (Parker and Stacey, 1994, 36).

Complexity theorists state that due to the mentioned characteristics, it is very difficult to predict and control the behaviour of chaotic organisations in the long future (Parker and Stacey, 1994, 41). However, in the short term complex systems are more predictable as "it takes time for small changes to escalate into major consequences" (Parker and Stacey, 1994, 15). Complex organisations need to live in a state of chaos or bounded instability to survive to the continuous change in their environment. Therefore they are far from being designed, controlled or managed in old fashioned manners (which seek to standardise predictable processes).

The relevance of complexity theory to this thesis is that it describes several aspects of the model of complex organisations. Similarly to complex systems, the model of complex organisations presented in chapter 1, contains processes appearing and disappearing, people performing a variety of tasks in different styles and information which is flowing through the entire organisation with no apparent patterns. Nevertheless, within this seeming disorder the organisation is working and surviving. The model of complex organisations is therefore a non-linear system that self-organises itself in a continuous change within bounded instability. Information systems and software designed for such organisations should be embedded in their continuous change and their bounded instability. In this way both the complex system and its information system will adapt and self-organise together. Therefore, in this context an information system is not a static piece of software but a living part of the organisation. The mentioned complexity theory concepts are useful devices to build and enrich the model of complex organisations. Unfortunately, they cannot be easily translated into practice and does not seem mature enough to be applied in the software development field.

c. Actor-network theory

Actor-network theory (ANT) was created by Michel Callon and Bruno Latour. It "draws on the sociology of science and technology" and is

"applied to the analysis of socio-technical innovation" (Harrison and Laberge, 2002, 497). ANT has proved to be a helpful tool to research situations which involve the use of information systems in a social context (Tatnall, 2003, 274) because it focuses on the interaction between humans and non-humans in socio-technical situations (Tatnall, 2003, 279). ANT sees the world as a network of actors. It reduces the complexities of the world by limiting analysis to a series of well defined, discrete entities (Callon, 1986, 28). An actor is "an entity able to associate texts, human, non-humans and money" (Selman and Wragg, 1999, 330) that builds a world with other entities like it. An actor-network is an interrelated set of entities that form part of the actor's world. A network is composed of "social and natural contexts" (Callon, 1986, 20) as human and non-human entities are enrolled as part of the network. Actors are themselves simplifications of other networks. Every point in an actor-network is the "intersection of two networks: one that it simplifies and another which simplifies it" (Callon, 1986, 32). Actors form networks by enrolling other entities in a process called translation. This process does not involve any definitive rules (Callon, 1986, 33). Translation involves:

- The definition of roles, their distribution, and the delineation of a scenario,
- The strategies in which an actor-world renders itself indispensable to others by creating a geography of obligatory passage points and
- The displacement imposed upon others as they are forced to follow the itinerary that has been imposed.

Source: Callon (1986, XVI)

Thus processes like decision-making may be formed by the enrolment of people by other people. Each process might be represented by a network of actors who are involved by other actors, to the convenience of the formers and because of the expertise and knowledge of the latter. As the needs of decision makers develop, they may ask others for more

information, may go through a workflow which can cross organisational boundaries, etc. Roles are not fixed or pre-established and their distribution may not be necessarily successful (Callon et al., 1986, XVI). Therefore organisations' behaviour is unpredictable and may change through time. ANT's perspective of the social world and in particular organisations shows them as unpredictable because the processes of enrolment and translation through which networks are built are not pre-established. Networks of actors could take any shape and size. Also, ANT emphasises the formation of multi-networks as different actors could enrol different people and form different networks which can then be enrolled by other networks. This resembles the multi-group aspect of the model of complex organisations. Business processes like decision-making and workflow can cross organisational boundaries, and at doing so different actor-networks are created. Additionally, ANT is a very useful framework to analyse organisations, especially to study interactions between actors such as social interactions between people and human-to-computer interactions. Social issues emerging from these interactions affect the way processes are carried out and consequently the information systems that are serving them.

In the model of complex organisations (see chapter 1) I defined actors as individuals or group of individuals involved in processes of decision-making. Actors participate in decision-making processes as their skills are needed and as other actors enrol them. Thus the concept of enrolment applies is useful to explain how the decision-making processes are formed. Each decision-making process in the model of complex organisations might be represented by a network of actors who are involved (or enrolled) by other actors, to the convenience of the latter and because of the expertise and knowledge of the former. However, the actor-network theory does not help to identify and explore the specific aspects that influence interactions. Besides, it focuses on the connections between humans and non-humans, and little is said about aspects that influence these connections. It could also offer a static perspective of organisations which is taken for granted. Interactions are represented as

constant in time, rejecting adaptability and the possibility of unpredictable behaviour is not considered.

Appendix 10: Information Systems Development Approaches

a. ISD Organisational Approaches

This group of approaches address "organisations as a whole: their inter-connection, strategic use, re-engineering and planning" (Avison and Fitzgerald, 2003, 43). The information systems arise from processes of discovering the organisations needs.

Systems approaches they consider an information system as a whole which has human, technical and organisational elements into it. Systems approaches are very popular in the teaching of information systems but unfortunately they are not reflected in the practices of software development. Software developers use a different set of methodologies based on different paradigms (fieldwork data, see category C5 in chapter 6). The following are the most popular systems approaches:

- **The general systems theory** is based in the principle that "systems can only be understood when viewed as a whole" (Ford and Woodroffe, 1994, 23). For the systemic a system can be described by identifying its elements, the relationships between the elements and the system's boundaries (Pfleeger, 1991, 11). People within organisations act purposefully, that is that all actions have an objective and the job of an IS developer is to undercover these purposes and capture them in the information systems. Additionally, organisational systems are less predictable than computer programs as they involve humans. People would follow instructions, make mistakes or interpret the system in different ways. There are two branches in the systems approach. The hard branch views systems "as though they exist and they can be engineered" and the soft branch sees systems as concepts not existing but representing

"a way of viewing, and therefore understanding, complex real-world activities" (Avison and Fitzgerald, 2003, 47).

- **Soft Systems Methodology** (SSM) "aims to bring about improvements in areas of social concern by activating in the people involved in the situation a learning cycle which is ideally never-ending" (Checkland and Holwell, 1997, 14). In the field of Information Systems SSM is concerned with understanding and dealing with problem situations in which there is a high human and social component. SSM is used at the beginning of IS projects to draw a rich picture of the problem domain. Root definitions are developed out of that picture and then a conceptual model is generated. By a comparison between the conceptual model and the real world a systems analyst is then able to identify the required changes to improve the situation.

- **Multiview** is a framework which sees ISD as a hybrid process involving the technologists (information systems developers) and the users for whom the system will be built (Avison et al., 1998, 127). Multiview comprises 5 stages each of which provides a different view of the system to be built.
  - analysis of human activity
  - analysis of information
  - analysis and design of socio-technical aspects
  - design of the human-computer interface
  - design of technical aspects

Effective Technical and Human Implementation of Computer-Based Systems (ETHICS) is an ISD approach that pays equal attention to the technological, social and organisational factors of an information system. Apart from designing computer applications ETHICS is also concerned with redesigning the working environment so users can improve their quality of live with the

work they do. Another aspect of ETHICS is its participative orientation. Those affected by the system such as users, managers, suppliers, customers, etc. may take part in its design and operation (Avison and Fitzgerald, 2003, 450). ETHICS sees ISD as a process of change and focuses on solving conflicts that arise between participants during that process. ETHICS however does not consider IS developers as an affected party as it only focuses on the organisation in which the information system is located. Also ETHICS does not pay especial attention to the development of software or the software environment at all.

IS Planning approach involves the strategic levels of the organisation. This approach sees information systems as a way of achieving the objectives of an organisation. In this approach Information Systems are planned alongside other areas of the organisation. Plans are built considering that information systems should be aligned with the overall strategic direction of the organisation. The concern of planning approaches is the definition of "optimal informatics architecture" (Beynon-Davies, 2002, 400) which is thought as reflecting the structure of the organisation (Beynon-Davies, 2002, 267). The process of planning comprises the analysis of information of the current activities of the organisation, the analysis of the current and future trends in the environment and the analysis of the trends in information technology. As with the previous mentioned approaches (and the following ones in this group) the planning approach lacks focus on the practicalities of the development of software.

IS Strategic approach is closely linked with the planning approach and involves top management levels too. According to Avison and Fitzgerald the strategic approach sees information systems as tools for "obtaining competitive advantage" (Avison and Fitzgerald, 2003, 49). The objectives are to "identify better ways of doing things" (ibid) which will lead the organisation to

"increased revenues, greater functionality, better products and service, improved presentation or image" and "an improvement to the organisation's competitive positioning" (ibid).

Business process reengineering is concerned with the re-definition of old and outdated rules and assumptions (Avison and Fitzgerald, 2003, 56). The redefinition of business processes involves also the creation or maintenance of the information systems that serve those processes. The aim of this redefinition is the improvement of organisational performance measures such as reduction of costs, increase productivity and quality. Differently to the three previous approaches, business process re-engineering does not hold holistic perspectives of the organisations. BPR'S focus is at a more micro level of the business processes.

Project Management focuses on particular endeavours of Information Systems Development. Its aim is to deliver information systems (and software) on time, within budget and with acceptable quality to satisfy users' needs. Project Management does not necessarily addresses the organisation as a whole, although there could be projects which involve information systems across the entire organisation. In general big projects are divided in smaller achievable projects which can be run and controlled easily.

b. ISD Modelling Approaches

Modelling approaches are centred on the task of modelling the system to be served and/or the system to be designed. For example, the system to be served could be a business process and the system to be designed could be a computer application to automate some steps of the process. Modelling is "the ability to describe the situation confronting the analyst" (Wilson, 1990, 10). The description of the problem situation is called a model and is used to help the developer to identify problematic areas within organisations which need to be addressed. An advantage of the

modelling approaches as seen by this thesis is that the resulting models could be a step towards the design and development of software. Different methodologies propose different kinds of models and focus on different aspects of organisations. Wilson (1990) defines four kinds of models. The first three were previously defined by Ackoff (1962) and are quantitative representation of the situation and the fourth was added by Wilson and is more qualitative. The kinds of models are:

- Iconic: a replica of the situation with similar properties and expected behaviour.
- Analogic: a model that reproduces a similar behaviour that that of the problem situation, though it is physically different.
- Analytic: mathematical or logical representation of the physical laws that govern the behaviour of the object of study.
- Conceptual: pictorial/symbolic models which cover the qualitative aspects of the situation and usually precedes the others.

There are three broad subgroups of approaches under the modelling approach:

> **Data oriented approaches** which are "derived from database methodologies" (Olle et al., 1991, 12) and are concentrated on classifying data and identifying their relationships in an organisation (Avison and Fitzgerald, 1995, 38). Data models are not necessarily built as part of a database project. A data model could be an end on itself (Avison and Fitzgerald, 2003, 76) that helps the developer to understand an organisation.

> **Process oriented approaches** are based on specifying the "functions of an information system" (Olle et al., 1991, 12). "Process modelling describes the logical (real-world) analysis of the processes" (Avison and Fitzgerald, 2003, 75). Generally, structured ISD methodologies follow process oriented approaches.

Their aim is to break down processes into smaller manageable pieces which can be transformed into computer programs.

Object oriented approaches unifies the two previous perspectives by modelling data and processes, as well as people and software (Avison and Fitzgerald, 1995, 38). Objects could represent anything in the real world. Objects have information about them, in the form of data fields, and can also perform activities in the form of methods. Relationships can be established between objects to represent how these objects interact in the real world. For example, the object "client" holds information about clients such as name and address and can "buy" (method) products. "Client" is related to "bill". The object "bill" contains information about bills issued to the organisation's clients. A "bill" can be "issued" (method) for any purchase that is made. Every organisation process or activity can be represented in this way. The advantage of this approach is that objects can be easily transformed into computer code which speeds up the process of development.

c. ISD People or Participative Approaches

These approaches emphasise "the role of people in developing and using an information system" (Avison and Fitzgerald, 1995, 38). Thus they refer to human and social issues within the organisations which is the concern of this thesis. Also, they go as far as accounting for the practicalities of the development and use of software. The approaches mentioned in the previous sections may work as they have proven their viability. However, in some situations, human or social issues may arise which affect the (use of the) information system developed. For example, users can reject software because they could fear that will make their job less secure. People approaches are designed so that people, especially end-users, can participate in the development process and understand

how an information system and/or software may benefit them. Participative approaches comprise:

Joint Application Development (JAD) is an approach for determining systems requirements. It is a participant process in which the stakeholders and a facilitator contribute with their views of the problem. The requirements agreed are then used for the "initial prototyping" (Turban et al., 2002, 493) of the information system.

End-user computing (EUC) emphasises users' capability of serving their own needs. With end-user computing users are able to develop and maintain their own software applications. EUC is ideal for user satisfaction and takes some pressure out of information systems departments. Usually EUC is used for standalone, simple applications leaving more complicated and bigger application for IS professionals.

Co-operative development is concerned with aiding analysts in the process of developing information systems. This approach uses groupware technologies and sees the process of development as a workgroup. "User involvement in development is especially important for groupware" (Grudin, 1991, 314) as it facilitates the process of understanding of group dynamics.

Ethnography is a well known practice in sociological and anthropological undertakings (Hammersley and Atkinson, 1995, 2; Geertz, 1993, 5). It comprises a variety of techniques which the ethnographer uses to immerse himself into others people problems so as to "hear, see and begin to experience reality as the participants do" (Marshall and Rossman, 1989). "Ethnographic studies are helpful in informing the systems design process" (Bentley et al., 1992, 1). It provides "information systems researchers with rich insights into the human, social and

organizational aspects of information systems development and application" (Harvey and Myers, 1995, 22). A weakness of this approach is that it does not account for the technological aspect of software development. So it would have to be used in conjunction with other approaches or methodologies which guide the development of software (for example a modelling approach).

d. ISD Engineering and Construction Approaches

As the name says these approaches see the development of information systems as engineering endeavours. Information systems are seen as tangible products which can be planned and developed accurately. Criticisms of these approaches are concerned with the lack of consideration of human and social factors. A well known example of an engineering approach is prototyping. **Prototyping** is the process of building software prototypes. A software prototype is a preliminary version or a model of all or part of a system before full commitment is made to develop it (Smith, 1990, 42). Prototyping is a very popular practice (fieldwork data, see category C5-5 in chapter 6) and is largely used for tackling ill-defined problem situations (like complex organisations) as they "serve as the formal statement of needs" (Smith, 1990, 45). There are two more motivations for using prototyping:

- Speeding up the process of development by presenting workable versions of the software in the early stages of the process. This perspective is also known as Rapid Application Development (RAD) which chooses effective practices that are oriented toward achieving your schedule objectives (McConnell, 1996, 3).

- Involving users by letting them participate in the process of development so as to build a better picture of their needs and the problem situation. This is also called co-operative prototyping in which users and designers are involved in an "ongoing learning process" (Bodker and Gronbaek, 1991, 331).

e. External Development

External development refers to the acquisition of off-the-shelf or shrinkwrap software packages or to the outsourcing of the development of IS. Off-the-shelf software is designed to fit the needs of a wide variety of customers. Although that to develop this kind of software one need to focus on generalities and commonalities among certain kinds of organisations and business processes, certainly, some of the already mentioned ISD approaches can be used in their design. However, in most cases customisation is needed to fit the particular needs of an organisation. External development is also subject to human and social issues in their target organisations in both, initial and customisation development work. Therefore it is relevant to this thesis. External development can also be done by outsourcing. Outsourcing is the commission of ISD to a third party. This could include the analysis, the gathering of requirements and the development of software as well. In this case the software product will be developed specifically to the (human and social) needs of the client organisation. Any ISD approach mentioned before could apply here.

f. ISD Software Approaches

These set of approaches assume that the solution for a problematic situation involves the development of software. As they focus on software development (they are technology oriented) they usually neglect the human and social aspects of target organisations.

Software engineering (SE) is one of the pioneer formal approaches to ISD. It is a "strategy of producing quality software" (Pfleeger, 1991, 5). It is mostly based on reductionist approaches in which the analysis of problem situation is done by "breaking that problem into pieces" and then building the solution by synthesising the solution components found for each *piece* of the problem

situation. SE is concerned with the construction of software. However, this is one of its main criticisms as it does not consider the human and social aspects of information systems such as people's interactions. Nevertheless, it was welcome at its time as it set the principles for a more "disciplined approach" (Avison and Fitzgerald, 2003, 140) to software development. Software engineering is usually associated with Information Systems Life Cycle. The life cycle represents the process of information systems development from the beginning to the very end. This process is divided into steps each of which is concentrated on different tasks that have to be accomplished before undertaking the next one. Many variations of the information systems life cycle have been developed most of them permit the revision of previous stages.

Automated or CASE tools provide leverage at any point of the development cycle (Fisher, 1994, 4). As the process of developing software is a human activity system, it also needs support from information systems (Beynon-Davies, 2002, 325). CASE tools are designed to support a particular methodology or a group of them. CASE tools are focused on the design and implementation of the software in different stages of the development life cycle. Front-end CASE tools are oriented to the analysis and design stages of the ISD cycle. Back-end CASE tools focus on programming, implementation and maintenance stages of the ISD process.

Component development consists of the production of software components which then can be assembled to create software applications rather than constructing them from scratch (Turban et al., 2002, 493). Components are reusable pieces of software and have their origins in objects. However components are bigger pieces of code which can contain more than one function. "Each component is designed to work in a variety of contexts and work in conjunction with a variety of others" (Wills, 1999, 61). Components

allow information systems and software to adapt to changes in business as they can be replaced easily by new components with different functionality (Allen and Frost, 1998, 3).

g. Web approaches

As web development is relative a new kind of development. I have included this category to emphasise issues that have emerged in this area. Baskerville and Pries-Heje (2001) published a study on web development practices that suggest that the concept of *methodology* is changing. What they say is not that they have created a new development methodology but that in the minds of (web) developers a development methodology (in general) has now a different meaning that it used to have. Web approaches are relevant to this thesis because they address current issues in the development world that are affecting developers and that need to be considered to keep track of the changes in technology. The new internet or web kind of methodology is constrained by time and ambiguities. Some differences with the previous development approaches can be highlighted:

- Release orientation (or incremental approach), the application is released in bits starting with the pieces of software containing the core functionalities.
- Parallel development – a number of activities take place at the same time
- Coding your way out – developers hack their applications to meet time pressures
- Quality is negotiable – quality is based on expectations of customers and not on fulfilment of requirements or the quality of the development process.
- Need for new working or team structures – because traditional structures fail

Baskerville and Pries-Heje's study suggests that the practice of information systems development is becoming more pragmatic. Decisions about the development practices are made on a daily basis depending on the situation. "The procedures, stages, and tools cannot be predefined, only the logical relation between the properties of the setting." (Baskerville and Pries-Heje, 2001, 65)

GLOSSARY

The **Agile Manifesto** was written by The Agile Alliance, a group made up by software development practitioners who have created development methodologies with a focus on human factors and adaptability. The Agile Manifesto defined the term Agile as the label for all methodologies designed to respond to change and determined four values common to all of them. The Agile Alliance includes: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Stephen J. Mellor, Ken Schwaber, Jeff Sutherland, and Dave "Pragmatic" Thomas. (Cockburn, 2000a, 177)

Browser: Software application used to view different types of Internet resources from the World Wide Web

Data Mining tools find hidden information that is "unlikely to be uncovered by users with query tools, regardless of their degree of knowledge, creativity and intuition" (Kelly, 1997, 228)

Data Warehouse: "A database with reporting and query tools, that stores current and historical data extracted from various operational systems and consolidated for management reporting and analysis." (Laudon and Laudon, 2000, 247)

HTML: Hypertext Markup Language, is the language used in the WWW

Mantra: sacred syllables, words or utterance. In software development it is a characteristic of software or a technique that should be always religiously incorporated in (the development of) any software.

OLAP: Online Analytic Processing tools are specially designed for managers and analysts to gain insight into data through different dimensions each of them representing different views of the organisation.

Prosody: Phonological term "which capture the notion of 'it ain't what you say but the way you say it' – as expressed through vocal variations in pitch (intonation), loudness (stress), speed, rhythm, pause, and tone of voice" (Crystal, 2001, 32).

REFERENCES

A dictionary of philosophy, (1984). 2nd edn., London, Macmillan.

Ahmad, K. and Al-Sayed, R., (2006), 'Community of Practice and the Special Language "Ground"', in Coakes, E. and Clarke, S. (eds.), *Encyclopedia of Communities of Practice in Information and Knowledge Management*, Hersey Pa., Idea Group Reference.

Aldred, C., (1994), '"Workflow" automation of the desktop', in Spurr, K. e. a. (ed.), *Computer support for co-operative work*, Chichester, John Wiley & Sons.

Allen, C., (2004), 'Tracing the Evolution of Social Software' < http://www.lifewithalacrity.com/2004/10/tracing_the_evo.html >, accessed 25/10/04.

Allen, P. R. and Frost, S., (1998), *Component-based development for enterprise systems : applying the select perspective*, Cambridge, Cambridge University Press.

Arnold, S. E., (2003), 'Social Software', *Searcher,* 11, no.9: 30-31

Avison, D. E. and Fitzgerald, G., (1995), *Information systems development : methodologies, techniques, and tools*, New York, McGraw-Hill.

Avison, D. E. and Fitzgerald, G., (2003), *Information systems development : methodologies, techniques, and tools*, New York, McGraw-Hill.

Avison, D. E. and Wood-Harper, A. T., (1990), *Multiview: An Exploration in Information Systems Development*, Oxford, Alfred Waller.

Avison, D. E., Wood-Harper, A. T., Vidgen, R. T. and Wood, J. R. G., (1998), 'A further exploration into information systems development: the evolution of Multiview2', *Information Technology and People,* 11, no.2: 124-139

Bandow, D., (1998), 'Working with the borg: trust, system development and dispersed work groups', Proceedings of the 1998 ACM SIGGROUP conference on computer personnel research, 163-169.

Baskerville, R. and Pries-Heje, J., (2001), 'Racing the e-bomb: how the internet is redefining information systems development methodology', Realingning Research and Practice in Information Systems

Development - IFIP TC8/WG8.2 Working Conference, Russo, L., Fitzgerald, B. and De Gross, J. (eds), 49-68.

Baym, N. K., (1995), 'The Performance of humor in computer-mediated communication', *Journal of Computer-Mediated communication,* 1, no.2, <www.ascusc.org/jcmc/vol1/issue2/baym.html>.

Beck, C. T., (2003), 'Initiation into qualitative data analysis', *Journal of Nursing Education,* 42, no.5: 231-234

Beck, K., (1999), *Extreme Programming Explained: Embrace Change*, Reading Massachusetts, Adison Wesley Longman Inc.

Bentley, R., Hughes, J. A., Randall, D., Rodden, T., Sawyer, P., Shapiro, D. and Sommerville, I., (1992), 'Ethnographically-informed systems design for air traffic control', CSCW Research Centre Computing and Sociology Departments, Lancaster University, < www.informatics.ed.ac.uk/teaching/modules/hci/atc.pdf >, accessed 02/01/2003.

Beynon-Davies, P., (2002), *Information systems : an introduction to informatics in organisations*, Basingstoke : Palgrave.

Biesenbach-Lucas, S. and Weasenforth, D., (2001), 'E-Mail and Word processing in the ESL Classroom: how the medium affects the message', *Language Learning and Technology,* 5, no.1: 135-165, <http://llt.msu.edu/vol5num1/weasenforth/default.htm>.

Biesenbach-Lucas, S. and Weasenforth, D., (2002), 'Virtual office hours: Negotiation strategies in electronic conferencing', *Computer Assisted Language Learning,* 15, no.2: 147-165

Bodker, S. and Gronbaek, K., (1991), 'Cooperative prototyping:users and designers in mutual activity', in Greenberg, S. (ed.), *Computer-supported cooperative work and groupware*, London, Harcourt.

Boland, R. J. J., (1985), 'Phenomenology: a preferred approach to research on information systems', in Mumford, E., Hirschheim, R. J., Fitzgerald, G. and Wood-Harper, A. T. (eds.), *Research methods in information systems : proceedings of the IFIP WG 8.2 Colloquium ... 1984*, Amsterdam, North-Holland.

Boyd, D., (2004), 'Autistic Social Software' < http://www.danah.org/papers/Supernova2004.html >, accessed 21/06/05.

Brooks, L., (1997), 'Structuration theory and new technology: analysing organizationally situated computer-aided design (CAD)', *Information Systems Journal,* 7, no.2: 133-151

Browne, E., (2003), 'Conversations in Cyberspace: a study of online learning', *Open Learning,* 18, no.3: 245-260

Brownlow, C. and O'dell, L., (2002), 'Ethical Issues for Qualitative Research in On-line Communities', *Disability and Society,* 17, no.6: 685-694

Bucholtz, M., (1999), '"Why be normal?": Language and identity practices in a comunity of nerd girls', *Language in Society,* 28, no.2: 203-223

Burrell, G. and Morgan, G., (1979), *Sociological paradigms and organisational analysis: elements of the sociology of corporate life*, London, Heinemann.

Callon, M., (1986), 'The sociology of an actor-network: the case of the electric vehicle', in Callon, M., Law, J. and Rip, A. (eds.), *Mapping the dynamics of science and technology : sociology of science in the real world*, Basingstoke, Macmillan.

Callon, M., Law, J. and Rip, A., (1986), 'Glossary', in Callon, M., Law, J. and Rip, A. (eds.), *Mapping the dynamics of science and technology : sociology of science in the real world*, Basingstoke, Macmillan.

Campbell, T., (1997), 'Technology, multimedia, and qualitative research in education', *Journal of Research on Computing in Education,* 30, no.2: 122-133, <http://search.epnet.com/direct.asp?an=390335&db=buh&tg=AN>.

Carroll, J. M. and Swatman, P. A., (2000), 'Structured-case: a methodological framework for building theory in information systems research', *European Journal of Information Systems,* 9, no.4: 235-242

Carson, D., Gilmore, A., Perry, C. and Gronhaug, K., (2001), *Qualitative marketing research*, London, Sage.

Checkland, P. B. and Holwell, S., (1997), *Information, systems and information systems: making sense of the field*, Chichester, Wiley.

Chisholm, P. S. R., Constantine, L., Cunningham, W., Hohmann, L. and Kerth, N., (1996), 'Soft Issues and Hard Problems in Software Development', Conference on Object Oriented Programming Systems Languages and Applications - Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications

Chou, C., (2001), 'Internet Heavy Use and Addiction among Taiwanese College Students: An Online Interview Study', *CyberPsychology & Behavior,* 4, no.5: 573-586, <http://search.epnet.com/direct.asp?an=5513110&db=buh&tg=AN>.

Clarke, S. and Lehaney, B., (2000), 'Introduction: Information Systems as Constrained Variety - Issues and Scope', in Clarke, S. and Lehaney, B. (eds.), *Human Centered Methods in Information Systems: Current Research and Practice*, Hershey PA, Idea Group Publishing.

Clegg, C. W., (2000), 'Sociotechnical principles for system design', *Applied Ergonomics,* 31, no.5: 463-477

Coakes, E. and Clarke, S., (2006a), 'The Concept of Communities of Practice', in Coakes, E. and Clarke, S. (eds.), *Encyclopedia of Communities of Practice in Information and Knowledge Management*, Hersey Pa., Idea Group Reference.

Coakes, E. and Clarke, S., (2006b), 'The Concept of Communities of Practice', in Coakes, E. and Clarke, S. (eds.), *Encyclopedia of Communities of Practice in Information and Knowledge Management*, Hersey Pa., Idea Group Reference.

Coakes, E., Willis, D. and Lloyd-Jones, R., (2000), 'Grafitti on the Long Wall: A Socio Technical Conversation', in Coakes, E., Willis, D. and Lloyd-Jones, R. (eds.), *The New SocioTech: Grafitti on the Long Wall*, London, Springer.

Coakes, E. W., Bradburn, A. and Sugden, G., (2004), 'Managing and leveraging knowledge for organisational advantage', *Knowledge Management Research & Practice,* 2, no.2: 118-128

Coakes, J. M. and Coakes, E. W., (2000), 'Specifications in Context: Stakeholders, Systems and Modelling of Conflict', *Requirements Engineering,* 5, no.2: 103-113

Cockburn, A., (1995), 'Growth of Human Factors in Application Development', Cockburn, Alistair, < http://alistair.cockburn.us/crystal/articles/gohfiad/growthofhumanfactorsinsd.htm >, accessed 11-05-2005.

Cockburn, A., (1998), 'Software Development and Process (ECOOP 98 panle)', Cockburn, Alistair, < http://alistair.cockburn.us/crystal/articles/sdap/swdevandprocess.htm >, accessed 11-05-2005.

Cockburn, A., (2000a), *Agile Software Development*, Cockburn * Highsmith Series Editors.

Cockburn, A., (2000b), 'Characterizing People as Non-Linear, First-Order Components in Software Development', Alistair Cockburn, < http://alistair.cockburn.us/crystal/articles/cpanfocisd/characterizingpeopleasnonlinear.html >, accessed 05-07-04.

Cockburn, A., (2001), 'Agile Software Development 2: The People Factor', Cockburn, Alistair, < http://alistair.cockburn.us/crystal/articles/asdpf/asd2peoplefactor.htm >, accessed 11-05-2005.

Cockburn, A., (2003), *People and Methodologies in Software Development,* Ph.D Thesis, University of Oslo.

Constantine, L., (2001), 'Back to the Future', *Communications of the ACM,* 44, no.3: 126-129

Cooper, A. and Reimann, R., (2003), *About face 2.0 : the essentials of interaction design*, New York, Wiley.

Crystal, D., (2001), *Language and the Internet*, Cambridge, Cambridge University Press.

DeMarco, T. and Lister, T., (1999), *Peopleware: Productive Projects and Teams*, New York, Dorset House.

Denzin, N. K. and Lincoln, Y. S., (1998), 'Introduction: Entering the Field of Qualitative Research', in Denzin, N. K. and Lincoln, Y. S. (eds.), *Collecting and interpreting qualitative materials*, Thousand Oaks, Calif., Sage Publications.

DeSanctis, G. and Poole, M. S., (1994), 'Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory', *Organisation Science,* 5, no.2: 121-147

Dictionary of theories, (1993), London, Gale Research International.

Drucker, P. F., (1998), 'Management's New Paradigms', Forbes Magazine, < www.forbes.com/forbes/1998/1005/6207152a.html >, accessed 20/06/2003.

du Plooy, N. F., (2003), 'The Social Responsibility of Information Systems Developers', in Clarke, S., Coakes, E. W., Hunter, G. M. and Wenn, A. (eds.), *Socio-technical and human cognition elements of information systems*, Hershey, Pa., Idea Group.

Eckert, P. and McConnell-Ginet, S., (2000), 'Communities of practice: where language, gender and power all live', in Coates, J. (ed.), *Language and gender: A reader*, Oxford, Blackwell.

Elliott, M. S. and Scacchi, W., (2003), 'Free software developers as an occupational community: resolving conflicts and fostering collaboration', Proceedings of the 2003 acm SIGGROUP conference on supporting groupwork, 21-30.

Eom, S. B., Lee, S. M., Kim, E. B. and Somarajan, C., (1998), 'A survey of decision support system applications (1988-1994)', *Journal of the Operational Research Society,* 49, no.2: 109-120

Evans, J. and Brooks, L., (2005), 'Understanding Collaboration Using New Technologies: A Structurational Perspective', *The Information Society,* 20: 215-220

Fisher, J., (2003), 'Human Factors and the Systems Development Process', in Clarke, S., Coakes, E. W., Hunter, G. M. and Wenn, A. (eds.), *Socio-technical and human cognition elements of information systems*, Hershey, Pa., Idea Group.

Fisher, S., (1994), *Stress in academic life : the mental assembly line*, Buckingham, Society for Research into Higher Education & Open University Press.

Fitzgerald, B., Russo, N. L. and Stolterman, E., (2002), *Information Systems Development: Methods in Action*, Berkshire, McGraw-Hill.

Flick, U., (1998), *An introduction to qualitative research*, London, Sage.

Flynn, D. J., (1998), *Information Systems Requirements: Determination and Analysis*, Berkshire, McGraw-Hill.

Folkman Curasi, C., (2001), 'A critical exploration of face-to-face interviewing vs. computer-mediated interviewing', *International Journal of Market Research,* 4, no.43: 361-376, <http://search.epnet.com/direct.asp?an=5540844&db=buh&tg=AN>.

Ford, N. J. and Woodroffe, M., (1994), *Introducing software engineering*, New York, Prentice Hall.

Fowler, M., (2004a), 'Is design dead?' Fowler, Martin, < http://www.martinfowler.com/articles/designDead.html >, accessed 10/11/2004.

Fowler, M., (2004b), 'The New Methodology', Fowler, Martin, < http://www.martinfowler.com/articles/newMethodology.html >, accessed 03/11/2004.

Gachet, A., (2001), 'A Framework for Developing Distributed Cooperative Decision Support Systems - Inception Phase', 4th Informing Science Conference, June 19-22, 214-221.

Gadamer, H. G. and Linge, D. E., (1976), *Philosophical hermeneutics*, Berkeley, Calif., University of California P.

Garvin, D. A., (1998), 'Building a learning organization', in Review, H. B. (ed.), *Harvard Business Review on knowledge management*, Boston, Mass., Harvard Business School.

Geertz, C., (1993), *The interpretation of Cultures*, London, Fontana Press.

Giddens, A., (1976), *New Rules of Sociological Method: A positive critique of interpretive sociologies*, London, Hutchinson.

Giddens, A., (1979), *Central problems in social theory : action, structure and contradiction in social analysis*, Basingstoke, Macmillan.

Giddens, A., (1984), *The Constitution of Society*, Cambridge, Polity Press.

Giddens, A., (1990), *The Consequences of Modernity*, Cambridge, Polity press.

Gill, J. and Johnson, P., (1991), *Research methods for managers*, London, Paul Chapman.

Gioia, D. A. and Pitre, E., (1990), 'Multiparadigm Perspectives on Theory Building', *Academy of Management Review,* 15, no.4: 584-602

Goode, W. J. and Hatt, P. K., (1952), *Methods in social research*, New York, McGraw-Hill.

Greenberg, S., (1991), 'Introduction', in Greenberg, S. (ed.), *Computer-supported cooperative work and groupware*, London, Harcourt.

Greenfield, P. M. and Subrahmanyam, K., (2003), 'Online discourse in a teen chatroom: New codes and new modes of coherence in a visual medium', *Applied Developmental Psychology,* 24: 713-738

Grudin, J., (1991), 'Obstacles to user involvement in software product development with implications for CSCW', in Greenberg, S. (ed.), *Computer-supported cooperative work and groupware*, London, Harcourt.

Grudin, J., (1994), 'Groupware and social dynamics: Eight Challenges for Developers', *Communications of the ACM,* 37, no.1: 93-105

Halfpenny, P., (1979), 'The analysis of qualitative data', *The sociological review,* 27, no.4: 799-827

Hammersley, M., (1992), *What is wornq with ethnography?*, London, Routledge.

Hammersley, M. and Atkinson, P., (1995), *Ethnography: Principles in Practice.* 2nd edn., London, Routledge.

Handel, M. and Herbsleb, J. D., (2002), 'What is Chat Doing in the Workplace?' Proceedings of the 2002 ACM conference on Computer supported cooperative work, 1-10.

Handy, C. B., (1985), *Understanding organizations.* 3rd edn., Harmondsworth, Penguin.

Handy, C. B., (1989), *The age of unreason.* 2nd edn., London, Arrow.

Handy, C. B., (1995), *Beyond certainty : the changing worlds of organisations*, London, Hutchinson.

Harding, S., (1987), *Feminism and methodology : social science issues*, Bloomington, Ind., Indiana University Press.

Harrison, D. and Laberge, M., (2002), 'Innovation, Identities and Resistance: The social construction of an innovation network', *Journal of Management Studies,* 39, no.4

Harvey, L. J. and Myers, M. D., (1995), 'Scholarship and practice: the contribution of ethnographic research methods to bridging the gap' < www.qual.auckland.ac.nz/Harvey-Myers.pdf >, accessed 02/01/2003.

Herbsleb, J. D., Paulish, D. J. and Bass, M., (2005), 'Global Software Development at Siemens: Experience from Nine Projects', Proceedings of the 27th international conference on Software engineering, 524-533.

Hine, C., (2000), *Virtual Ethnography*, London, Sage.

Hirschheim, R. J., (1985), 'Information systems epistemology: an historical perspective', in Mumford, E., Hirschheim, R. J., Fitzgerald, G. and Wood-Harper, A. T. (eds.), *Research methods in information systems : proceedings of the IFIP WG 8.2 Colloquium ... 1984*, Amsterdam, North-Holland.

Holmes, J. and Meyerhoff, M., (1999), 'The community of practice: Theories and methodologies in language and gender research', *Language in Society,* 28, no.2: 173-183

Holsti, O. R., (1969), *Content analysis in the social sciences and humanities*, Reading, Mass., Addison-Wesley.

Holtham, C., (1992), *Executive information systems and decision support*, London, Chapman & Hall.

Hycner, R. H., (1999), 'Some guidelines for the phenomenological analysis of interview data', in Bryman, A. and Burgess, R. G. (eds.), *Qualitative Research*, Vol. III, London, Sage.

Jackson, M. C., (1985), *Systems methodology for organisational analysis*, Hull, Department of Management Systems and Sciences, University of Hull.

Jacobson, D., (1996), 'Contexts and Cues in Cyberspace: The Pragmatics of Naming in Text-Based Virtual Realities', *Journal of Anthropological Research,* 52: 461-479, <http://people.brandeis.edu/~jacobson/context_and_cues.pdf>.

Johnstone, B., (2002), *Discourse Analysis*, Malden, Mass., Blackwell.

Jones, M., (1999), 'Structuration Theory', in Currie, W. L. and Galliers, B. (eds.), *Rethinking management information systems : an interdisciplinary perspective*, Oxford, Oxford University Press.

Jones, M., Orlikowski, W. J. and Munir, K., (2004), 'Structuration Theory and Information Systems: A Critical Reappraissal', in Mingers, J. and Willlcocks, L. (eds.), *Social Theory and Philosophy for Information Systems*, Chichester, England, John Wiley & Sons Ltd.

Kelly, S., (1997), *Data warehousing in action*, Chichester, John Wiley.

Kettinger, W. J. and Grover, V., (1997), 'The use of Computer-mediated Communication in an Interorganizational Context', *Decision Sciences,* 28, no.3: 513-555

Kim, E. E., (2004), 'A Manifesto for Collaborative Tools', Blue Oxen Associates, < http://www.blueoxen.org/papers/0000D/ >, accessed 29/11/2004.

Klein, H. K. and Myers, M. D., (1999), 'A set of principles for conducting and evaluating interpretive field studies in information systems', *MIS Quarterly,* 23, no.1: 67-94, <http://search.epnet.com/direct.asp?an=1852786&db=buh&tg=AN>.

Laudon, K. C. and Laudon, J. P., (2000), *Management Information Systems. Organization and technology in the networked enterprise.* 6th edn., Upper Saddle River, NJ., Pretince Hall International Inc.

Laudon, K. C. and Laudon, J. P., (2005), *Essentials of Management Information Systems*, New Jersey, Pretince Hall.

Lea, M. and Spears, R., (1991), 'Computer-mediated communication, de-individuation and group decision-making', in Greenberg, S. (ed.), *Computer-supported cooperative work and groupware*, London, Harcourt.

Lee, A. S., (1991), 'Integrating positivist and interpretive approaches to organizational research', *Organisation Science,* 2, no.4: 342-365

Lee, F. S. L., Vogel, D. and Limayem, M., (2003), 'Virtual community informatics: A review and research agenda', *Journal of Information Technology Theory and Application,* 5, no.1: 47-61

Lucas, H. C., (1994), *Information Systems concepts for management.* 5th edn., New York, McGraw-Hill.

Madge, C. and O'Connor, H., (2004), 'Exploring the internet as a medium for research: web-based questionnaires and synchronous virtual interviews', Research Methods Festival

Mann, C. C., (2002), 'Why Software is so Bad', *Technology Review*, July/August, <http://sern.cpsc.ucalgary.ca/courses/SENG/513/F2003/papers/WhySoftwareIsSoBad.pdf>.

Marakas, G. M., (2003), *Decision support systems in the 21st century*, Upper Saddle River, N.J., Prentice Hall.

Marcoccia, M., (2004), 'On-line polylogues: conversation structure and participation framework in internet newsgroups', *Journal of Pragmatics*, 36, no.1: 115-145

Marshall, C. and Rossman, G. B., (1989), *Designing Qualitative Research*, Newbury Park, Calif., Sage Publications.

Mason, J., (1996), *Qualitative Researching*, London, Sage.

May, T., (1997), *Social research : issues, methods and process*, Buckingham, Open University Press.

McConnell, S., (1993), *Code complete : a practical handbook of software construction*, Redmond, Wash, Microsoft Press.

McConnell, S., (1996), *Rapid development : taming wild software schedules*, Redmond, Wash., Microsoft Press.

McDaniel, S. E., Olson, G. M. and Magee, J. C., (1996), 'Identifying and Analyzing multiple threads in computer-mediated and face-to-face conversations', Proceedings of the 1996 ACM conference on computer supported cooperative work, 39-47.

Metz, J. M., (1994), 'Computer-mediated communication: Literature Review of a new context', *Interpersonal Computing and Technology: An Electronic Journal for the 21st century*, 2, no.2: 31-49, <www.helsinki.fi/science/optek/1994/n2/metz.txt>.

Miles, M. B. and Huberman, A. M., (1994), *Qualitative data analysis : an expanded sourcebook*. 2nd edn., Thousand Oaks, Calif., Sage.

Millen, D. R. and Dray, S. M., (2000), 'Information sharing in an online community of journalists', Aslib Proceedings, 166-173.

Mingers, J. and Willlcocks, L., (2004), 'Preface', in Mingers, J. and Willlcocks, L. (eds.), *Social Theory and Philosophy for Information Systems*, Chichester, England, John Wiley & Sons Ltd.

Mumford, E., (1993), *Designing human systems for health care : the ethics method*, Cheshire, Eight Associates.

Myers, M. D., (2003), 'Qualitative Research in information systems', MISQ Discovery, < www.qual.auckland.ac.nz >, accessed 25/05/2003.

Nonaka, I., (1998), 'The knowledge creating company', in Review, H. B. (ed.), *Harvard Business Review on knowledge management*, Boston, Mass., Harvard Business School.

Norris, C., (1993), 'Some Ethical considerations on Field-Work with the Police', in Hobbs, D. and May, T. (eds.), *Interpreting the Field: accounts of ethnography*, Oxford, Clarendon Press.

Olle, W. T., Hagelstein, J., MacDonald, I. G., Rolland, C., Sol, H. G., Van Assche, F. J. M. and Verrijn-Stuart, A. A., (1991), *Information systems methodologies : a framework for understanding*, Wokingham, Addison-Wesley.

Orlikowski, W. J., (1992), 'The duality of technology: Rethinking the concept of technology in organizations', *Organisation Science,* 3, no.3: 398-427

Orlikowski, W. J., (1993), 'CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development', *MIS Quarterly,* 17, no.3: 309-340

Orlikowski, W. J., (1997), 'Learning from Notes: organizational issues in groupware implementation.' in III, R. L. R. (ed.), *Knowledge management tools*, Boston, Mass., Butterworth-Heinemann.

Orlikowski, W. J., (2000), 'Using Technology and Constituting Structures: A practice Lens for Studying Technology in Organizations', *Organisation Science,* 11, no.4: 404-428

Orlikowski, W. J. and Baroudi, J. J., (1991), 'Studying Information Technology in Organizations: Research Approaches and Assumptions', *Information Systems Research,* 2, no.1: 1-28

Orlikowski, W. J. and Robey, D., (1991), 'Information Technology and the Structuring of Organizations', *Information Systems Research,* 2, no.2: 143-169

Parker, D. and Stacey, R. D., (1994), *Chaos, management and economics : the implications of non-linear thinking*, London, Institute of Economic Affairs.

Parks, M. R., (1996), 'Making friends in cyberspace', *Journal of Computer-Mediated communication,* 1, no.4, <www.ascusc.org/jcmc/vol1/issue4/parks.html>.

Parrish, R., (2002), 'Conversation analysis of internet chat rooms' < http://www.polisci.wisc.edu/~rdparrish/Chat%20Rooms%20for%20Web%20Site.htm >, accessed 16/02/2004.

Perakyla, A., (1997), 'Reliabilityand Validity in Research based on Tapes and Transcripts', in Silverman, D. (ed.), *Qualitative research: theory, method and practice*, London, Sage.

Perry, D. E., (2004), 'Dimensions of Software Evolution', in Madhavji, N., Lehmann, M. M., Ramil, J. and Perry, D. (eds.), *Software Evolution and Feedback*, John Wiley & Sons.

Pfleeger, S. L., (1991), *Software engineering : the production of quality software.* 2nd edn., New York, Macmillan Pub. Co.

Phillips, E. M. and Pugh, D. S., (2000), *How to get a PhD: a handbook for students and their supervisors*, Buckingham, Open University Press.

Pressman, R. S., (2000), *Software engineering : a practitioner's approach.* 5th edn., London, McGraw Hill.

Priest, H., Roberts, P. and Woods, L., (2002), 'An overview of three different approaches to the interpretation of qualitative data. Part 1: theoretical issues', *Nurse Researcher,* 10, no.1: 30-41

Punch, K. F., (1998), *Introduction to social research: quantitative and qualitative approaches*, London, SAGE.

Remenyi, D., Williams, B., Money, A. and Swartz, E., (1998), *Doing research in business and management: an introduction to process and method*, London, SAGE.

Rheingold, H., (1994), *The virtual community : finding connection in a computerized world*, London, Secker & Warburg.

Rising, L., (2001), 'Agile Methods: What's it All About?' DDC-I Online News vol 2(9), < http://www.ddci.com/news_vol2num9.shtml >, accessed 09-7-2004.

Rose, J., (1998), 'Evaluating the contribution of structuration theory to the information systems discipline', Proceedings of the European Conference of Information Systems, BAETS, W. R. J. (ed), 1-22.

Rose, J., (2000), *Information Systems Development as Action Research – Soft Systems Methodology and Structuration Theory*, Lancaster, Lancaster University.

Rose, J. and Hackney, R., (2003), 'Towards a structurational theory of information systems: a substantive case analysis', 36th Hawaii International Conference on Systems Sciences, 258-266.

Rose, J. and Scheepers, R., (2001), 'Structuration theory and information systems development; frameworks for practice', European Conference on Information Systems, Smithson, S. and Avgerinou, S. (eds), 217-231.

Rosenberg, D. and Scott, K., (1999), *Use case driven object modeling with UML : a practical approach*, Reading, Mass., Addison-Wesley.

Savenye, W. C. and Robinson, R. S., (2001), 'Qualitative Research Issues and Methods: An Introduction for Educational Technologists', The handbook of research for Educational Communications and Technology, < >, accessed 04/08/2003.

Schrage, M., (1997), 'Collaborative tools: A first look.' in III, R. L. R. (ed.), *Knowledge management tools*, Boston, Mass., Butterworth-Heinemann.

Schultze, U. and Orlikowski, W. J., (2004), 'A Practice Perspective on Technology-Mediated Network Relations: The Use of Internet-Based Self-Serve Technologies', *Information Systems Research,* 15, no.1: 87-106

Sedelnikov, A., (2004a), 'Internet Forums: The Return', Seeking Usability, < http://usabilist.de/seeking/issue.php?article=40719&lang=EN >, accessed 20/09/2004.

Sedelnikov, A., (2004b), 'A usability of the Internet discussion forums', Seeking usability, < http://usabilist.de/seeking/issue.php?article=10705&lang=EN >, accessed 20/09/2004.

Selman, P. and Wragg, A., (1999), 'Local Sustainability Planning: From Interest-driven Networks to Vision-driven Super-Networks', *Planning Practice & Research,* 14, no.3: 329-340

Shami, N. S., Bos, N., Wrigth, Z., Hoch, S., Kuan, K. Y., Olson, J. and Olson, G., (2004), 'An experimental simulation of multi-site software development', Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research, 1-12.

Shim, J. P., Warketin, M., Courtney, J. F., Power, D. J., Sharda, R. and Carlsson, C., (2002), 'Past, present, and future of decision support technology', *Decision Support Systems,* 33: 111-126

Shirky, C., (2003), 'A Group is its own worst enemy', Shirky.com, < http://shirky.com/writings/group_enemy.html >, accessed 20/07/2004.

Shirky, C., (2004a), 'Group as User: Flaming and the Design of Social Software', Shirky.com, < http://shirky.com/writings/group_user.html >, accessed 30-11-04.

Shirky, C., (2004b), 'Situated Software', Shirky.com, < http://www.shirky.com/writings/situated_software.html >, accessed 21/02/2005.

Sink, E., (2004), 'Hazards of Hiring', MSDN, < http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoftware/html/software07072004.asp >, accessed 13/07/2004.

Smith, M., Cadiz, J. and Burkhalter, B., (2000), 'Conversation trees and threaded chats', Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, 97-105.

Smith, M. F., (1990), *Software prototyping : adoption, practice and management*, London, McGraw-Hill.

Spencer, L., Ritchie, J. and O'Connor, W., (2003), 'Analysis: Practices, principles and processes', in Ritchie, J. and Lewis, J. (eds.), *Qualitative research practice : a guide for social science students and researchers*, London, SAGE.

Spolsky, J., (2001), 'Don't Let Architecture Astronauts Scare You', Joel on Software, < http://www.joelonsoftware.com/articles/fog0000000018.html >, accessed 21/09/2004.

Spolsky, J., (2002a), 'Five Worlds', Joel on Software, < http://www.joelonsoftware.com/articles/FiveWorlds.html >, accessed 30-06-04.

Spolsky, J., (2002b), 'The Iceberg Secret, Revealed', Joel on Software, < http://www.joelonsoftware.com/articles/fog0000000356.html >, accessed 30-06-04.

Spolsky, J., (2003), 'Building Communities with Software', Joel on Software, < http://www.joelonsoftware.com/articles/BuildingCommunitieswithSo.html >, accessed 05-07-2004.

Spolsky, J., (2004), 'It's Not Just Usability', Joel on Software, < http://www.joelonsoftware.com/articles/NotJustUsability.html >, accessed 07/09/2004.

Spradley, J. P. and McCurdy, D. W., (1972), *The cultural experience: ethnography in complex society*, Chicago Ill., Henley on Thames.

Sproull, L. and Kiesler, S., (1986), 'Reducing Social Context Cues: Electronic mail in organizational communication', *Management Science*, 32, no.11: 1492-1512

Stacey, R. D., (2001), *Complex responsive processes in organizations : learning and knowledge creation*, London, Routledge.

Stacey, R. D., Griffin, D. and Shaw, P., (2000), *Complexity and management : fad or radical challenge to systems thinking?*, London, Routledge.

Stanley, L. and Wise, S., (1990), 'Method, methodology and epistemology in feminist research processes', in Stanley, L. (ed.), *Feminist Praxis: Research, theory and epistemology in feminist scholarship*, London, Routledge.

Sweet, C., (2001), 'Designing and conducting virtual focus groups', *Qualitative Market Research: An International Journal*, 4, no.3: 130-136, <http://search.epnet.com/direct.asp?an=9801730&db=buh&tg=AN>.

Tang, J. C., (1991), 'Findings from observational studies of collaborative work', in Greenberg, S. (ed.), *Computer-supported cooperative work and groupware*, London, Harcourt.

Tatar, D. G., Foster, G. and Bobrow, D. G., (1991), 'Design for conversation: lessons from cognoter', in Greenberg, S. (ed.), *Computer-supported cooperative work and groupware*, London, Harcourt.

Tatnall, A., (2003), 'Actor-network theory as a socio-technical approach to information systems research', Hershey, PA, Idea Group.

Taylor, S. J. and Bogolan, R., (1984), *Introduction to qualitative research methods*, New York, John Wiley.

Tesch, R., (1990), *Qualitative Research: analysis types and software tools*, New York, The Farmer Press.

The Cambridge dictionary of philosophy, (1999). 2nd edn., Cambridge, Cambridge University Press.

Thomas, J., (1995), *Meaning in interaction : an introduction to pragmatics*, London, Longman.

Thomsen, S. R., (1996), '"@ work in cyberspace: Exploring practitioner use of the PRForum"', *Public Relations Review,* 22, no.2: 115-131

Tice, T. N. and Slavens, T. P., (1983), *Research guide to philosophy*, Chicago, American Library Association.

Travers, M., (2001), *Qualitative research through case studies*, London, SAGE.

Turban, E. and Aronson, J. E., (2001), *Decision support systems and intelligent systems*, Upper Saddle River, N.J., Pearson Higher Education.

Turban, E., King, D., Lee, J., Warketin, M. and Chung, H. M., (2002), *Electronic commerce 2002 : a managerial perspective*. International ed. edn., Upper Saddle River, Prentice Hall.

Van der Aalst, W. and Van Hee, K., (2002), *Workflow management : models, methods, and systems*, Cambridge, Mass, MIT Press.

Walsham, G., (1993), *Interpreting Information Systems in Organizations*, Chichester, Wiley.

Walsham, G., (1995), 'Interpretive case studies in IS research: nature and method', *European Journal of Information Systems,* 4, no.2

Walsham, G., (1998), 'IT and Changing Professional Identity: Micro-Studies and Macro-Theory', *Journal of the American Society for Information Science,* 49, no.12: 1081-1089

Walsham, G. and Han, C.-K., (1991), 'Structuration theory and information systems research', *Journal of Applied Systems Analysis,* 17: 77-85

Warne, L., (2003), 'Conflict and Politics and Information Systems Failure: A challenge for Information Systems Professionals and Researchers', In Clarke, S., Coakes, E. W., Hunter, G. M. and Wenn, A. (eds.), *Socio-*

technical and human cognition elements of information systems, Hershey, Pa., Idea Group.

Weinberg, G. M., (1971), *The Psychology of Computer Programming*, New York, Van Nostrand Reinhold.

Wenger, E., (2003), *Communities of Practice: Learning, meaning, and identity*, Cambridge, Cambridge University Press.

Williams, M. and May, T., (1996), *Introduction to the philosophy of social research*, London, UCL Press.

Willis, D. and Coakes, E. W., (2000), 'Enabling Technology for Collaborative Woking: A Socio-Technical Experience', in Clarke, S. and Lehaney, B. (eds.), *Human Centered Methods in Information Systems: Current Research and Practice*, Hershey PA, Idea Group Publishing.

Wills, A. C., (1999), 'Designing Component Kits and Architectures', in Barroca, L. (ed.), *Software Architectures: advances and applications*, Springer.

Wilson, B., (1990), *Systems : concepts, methodologies, and applications*, Chichester, Wiley.

Wilson, P., (1991), *Computer supported cooperative work : an introduction*, Oxford, Intellect.

Winograd, T. and Flores, F., (1986), *Understanding computers and cognition : a new foundation for design*, Norwood, N.J., Ablex.

Wolcott, H., (1995), *The Art of Fieldwork*, Walnut Creek, AltaMira Press.

Yourdon, E., (1989), *Modern structured analysis*, Englewood Cliffs, N.J., Prentice-Hall International.

Yule, G., (1996), *Pragmatics*, Oxford, Oxford University Press.