Action in Context – Context in Action: Towards a Grounded Theory of Software Design

Brian Robert Webb

Thesis submitted for the degree of PhD.

Department of Computer Science
University College London
Gower Street, London, WC1E 6BT

July 2001

Dedication

The writing of this thesis has been a personal, as well as an intellectual, journey – and also an odyssey! It is dedicated to those who made that journey possible.

To my father and mother, Harry and Frances, who began my journey, to my wife Judith who has travelled the greater part of it, and to our children, Thomas, Kirsten and Sarah, for whom another journey has just begun.

Acknowledgements

In completing this work I am indebted to the following people,

Dr Janet McDonnell, my supervisor for her support and guidance throughout, and for her willingness to take over work-in-progress and see it through to completion. I wish her every success in the future.

Professor Russel Winder, Department of Computer Science, Kings College London, for initiating the research and for suggesting pragmatism allied to academic rigour.

Professor Gail Murphy, Department of Computer Science, University of British Columbia, for hosting a research visit in 1999 and for bringing a sceptical "software engineering eye" to the work.

Seamus Gallagher and Bride Mallon, two former research students, who have contributed through their special insights, in jointly authored papers and in the collection of empirical data.

Finally I would like to thank my wife, Judith, for her unstinting support and encouragement over the years – and for the *many* sacrifices that has entailed. Hers is indeed the greatest contribution.

"It is no linguistic accident that "building", "construction", "work", designate both a process and its finished product. Without the meaning of the verb that of the noun remains blank".

(John Dewey, Art as Experience, 1934, p. 51, quoted in Strauss and Corbin, 1990, p. 258)

To this we may add "design".

Abstract

This thesis develops a model and a theory of software design. Thirty-two transcripts of interviews with software designers were analysed using the Grounded Theory method. The first set of sixteen interviews drawn from the field of Digital Interactive Multimedia (Data-set A) was used to develop the model and theory, the second set of sixteen interviews drawn from one source of technical literature (Data-set B) was used to test and enhance the initial outcomes. Final outcomes are then grounded in the general literature on problem solving and design. The model is concerned to capture a rich, holistic picture of software design. It is descriptive rather than prescriptive, concerned to capture how software design *is* done rather than advocate how it *ought* to be done. The theory is a development of the model and is presented initially as a theoretical framework and then as a series of propositions. The theoretical framework is a function of the juxtaposition of specific properties or attributes of the "core category", which uniquely explains the phenomenon. Its outcome is four design scenarios. Each scenario is of interest as an explanation of software design practice but two scenarios wherein such practice does not "fit" the design context are of most interest. It is argued that these scenarios can be used to identify and explain design breakdowns. Finally, the thesis purports to explicate the "Meta-process" - the process through which the inductive model and theory was developed. This is an unusual objective for a piece of IS research but valid nonetheless and significant, given the complexity of the research method used and the dearth of good process accounts in the IS literature and elsewhere.

Contents

Figures

Chapter One: Introduction to the thesis

1.1 Introduction and overview

Action in context- context in action: towards a grounded theory of software design

Inasmuch as a title can convey content, the title of this thesis attempts to capture the essence of the research study reported on the pages that follow. The field of study is *software design* - a comparatively new field that promises much for our understanding of the way we develop and use software. Software design is a development out of the referent disciplines of computer science, software engineering, human computer-interaction and graphic design, and these too fall tangentially within the study's scope. Specifically however, the study is interested in how professional designers design software, and is focused on those at the sharp end of the field, where, as Winograd (1996:vi) points out, "the rubber meets the road".

A *grounded theory* of software design is proposed based on the inductive analysis of interviews with such practitioners. The theory is presented in its final form as a series of propositions but is given specificity through a theoretical framework identifying four design scenarios. Each scenario is explanatory of software design practice but those scenarios within which design practice does not "fit" design context are of most interest. It is argued that such scenarios can help explain when and how design breakdowns occur.

Yet this is not a 'final' theory. No presumption is made on the basis of the analysis of thirty-two interviews with practitioners in specific domains of software design to have developed a general or formal theory of the field. Rather the theory that is presented is

based upon the analysis of available empirical data but subsequently "grounded" in the literature. "*Towards*" is an important qualification. It also reflects something of the nature of the research effort and its outcomes. A claimed contribution of this thesis is to have explicated the "Meta-process" through which the theory, and underlying model, was developed.

The theory is a development from an inductive model of software design. This is not a normative model advocating how design *ought* to be done but a descriptive model documenting how design *is* done - or at least how it was observed to be done during this study. It is argued that the discipline of software design needs more and better descriptive models of design practice to better inform the development of appropriate methods, tools and languages to support that practice.

The construction *action in context – context in action* embodies the essence of the software design process as it was observed. Software design consists of a series of actions and interactions taken in a specific context. Context explains actions and interactions and their outcomes. It is suggested that interventions to improve the software design process are best directed at assisting designers to manage or control the context, or at least to mitigate the worst negative aspects of it.

1.2 Setting the scene

Through supervision of a research programme that identified two distinct design paradigms in the field of Digital Interactive Multimedia - DIMM (Gallagher, 1999) and the development of papers based on this research (Gallagher and Webb, 1997,

Gallagher and Webb, 2000) I became interested in the super-ordinate discipline of software design. In particular I became interested in the following questions

1. What is software design?
2. How do software designers design? (What do software designers do when they design and why?)
3. How may such knowledge inform interventions to improve software design practice?

Whereas Gallagher's work was concerned, primarily, to identify differences between Software Engineering and Graphic Design approaches to Multimedia development, I wished to study their commonality. I believed that a rich description of the common design process, from the point of view of the practitioners themselves, could make a significant contribution to the field. The current situation within DIMM, and within its super-ordinate discipline of software design suggested that basic research at a higher level of abstraction would be beneficial. For problems and issues in the sub field of DIMM see, inter alia, (Latchem et al 1993; Dospisil and Polgar, 1994; Fisher, 1994; Garzotto et al, 1995; Mulhauser and Effelsberg, 1996; Powell, 1998) and in the broader field of software design see (Kapor, 1991; Nielsen, 1993; Wiklund, 1994; Brown and Duguid, 1994; Brooks, 1995, Shneiderman, 1997).

In the chapters that follow a case is made for the development of models and theories as one, important, response to this situation. Chapter Two (2.4) notes a deficiency in traditional approaches to the design and development of software and increasing calls for new approaches (Blum, 1996; Carroll, 1997, Winograd and Flores, 1987; Winograd et al, 1996). A taxonomy of inquiring systems (Van Gigsch and Pipino, 1986) is used to set out the role and contributions of models and theories in IS

research (2.2.1). In addition the special value of descriptive models is highlighted (2.2.4). In Chapter Three a general case for theory generation over theory testing is made through a discussion of the Grounded Theory method. Finally, in Chapter Eight, the outcomes of this research are themselves "grounded" in existing theories and models of design and problem solving (8.3), and the contribution of this work evaluated (8.4 – 8.5).

A small but related piece of research motivated and informed the main study. Arising out of an early investigation into software engineering, four interviews were conducted, transcribed and analysed using the grounded theory method. These served as a pilot to the main study (Chapter Four, 4.4.1.6) and as a personal benchmark against which recommendations for further research into inter-disciplinary comparisons could be made (Chapter Eight, 8.6).

1.3 Scope of the thesis

Following Dasgupta (1991), Digital Interactive Multimedia is considered a sub-field of software design, which itself is considered a sub-field of design. In this thesis a hierarchical view of the design disciplines provides a theoretical and practical basis for

(A) Limiting the scope of the investigation to a small but defined area of a much larger field of investigation.

(B) Comparing design activity at different levels of granularity. For example, an inductive model of design based on the common activities of Software Engineers and Graphic Designers within the field of DIMM is developed (Chapters Five and Six) and then applied to the broader field of software design (Chapter Seven).

(C) Comparing disciplines within a sub field. Thus, although the focus of this thesis was to identify a common practice in software design, differences within and between disciplines can be accommodated as instantiations of a general class.

(D) Tracing the impact of paradigm on method. Although the actual interrelationships between levels in the design hierarchy may be quite complex (Wernick and Winder, 1994), for the purposes of this thesis, it is assumed that a paradigm is associated with particular methods and vice versa.

(E) Explicating the process, or Meta-process, which lies behind the development of the paradigm model. In this sense, the process behind the process, at the object level of inquiry as defined by Van Gigsch and Pipino (1986) and as described in Chapter Two (2.2).

The scope of the thesis is further limited because the objective of the study is to develop a *substantive* not a *formal* theory. A substantive theory "evolves from a study of a phenomenon situated in one particular situational context" (such as the study of executives in an organisation). On the other hand, a formal theory "emerges from a study of a phenomenon examined under many different types of situations" (such as a study of status in the family, work and in society) (Strauss and Corbin, 1990:174).

This thesis does not present a formal theory of design but investigates design as it is practised in a particular situational context – the field of software. In doing so many different cases of software design are examined – design in the small, design in the large, software engineering, graphic design, in different types of organisations, by different types of designer (age, education, experience), across different geographical locations and cultures. But the focus of the study remains, and is embedded within, software design. The generalisability of the theory is increased the greater the range of

cases studied but the theory remains substantive. This has practical limitations not only for any claimed contribution but also for the scope of data collection and analysis (see Chapter Four).

1.4 Claims made of the thesis

Three principal claims are made of this work

1.4.1 Claims to have developed an inductive model of software design

The process of model development is discussed in Chapter Four and the model is presented in Chapter Five. The model is "inductive" on two counts (a) its is derived from the inductive processes of the Grounded Theory method and (b) it may be used to analyse other design practice, within DIMM, or any other field of software design, or in other fields of design. This is discussed as further research in Chapter Eight (8. 6.3). It is argued that this model permits consideration of a wide range of factors acting and interacting upon design practice, and not normally included in traditional models and theories of software design, for example the importance of individual conditions and of the "pulls" of the design itself (Chapter Five, 5.5). In summary a rich, holistic picture of software design practice is presented. The model is adaptive – it can be used to describe design activity in a number of different contexts, and predictive, in that within a specified context it is possible to predict design strategies and their consequences.

1.4.2 Claims to have developed a theory of software design

A theoretical framework is developed from the model that identifies four design contexts or scenarios. Each scenario is of interest in explaining design practice but two in which design strategies do not match the context are particularly interesting. It is argued that these can be used to identify and explain design breakdowns. The

theoretical framework is summarised as a set of propositions that encapsulate the theory that has been developed (Chapter Six, 6.6). In Chapter Seven the model (and by implication the theoretical development of that model) is validated using another data set. However this does not constitute theory testing and this is left to other researchers.

1.4.3 Claims to have explicated the Meta-process

In this thesis a "Meta-process" is defined as the process behind the process. Taking as its starting point the dearth of process information in many good design studies and the need for such information to encourage more and better studies of software design, the thesis sets out to explicate the process used to generate the reported results. To the extent that this has been successful, other researchers will be able to replicate the research or at least to make an informed assessment of its value. Whilst no measure can be given here of either of these possible outcomes, a contribution to the literature is claimed in (a) making the intention to map out the process clear at the outset of the thesis (b) making the framework explicit throughout - and doing so a virtue[1] and (c) evaluating the results (Chapter Eight, 8.5.3).

In additions a number of secondary claims are made of the thesis

1.4.4 Claims to have contributed to a paradigm model of software design

Based on Kuhn's (1970) and Dasgupta's (1991) definition of the term 'paradigm' and on common usage of the term within the fields of computer science and information systems, two subsidiary claims are made.

(a) The model contributes to our understanding of a Disciplinary Matrix of software design

(b) The model is a basis for developing software design languages, methods and tools.

---

[1] An analogy may be made with Sir Richard Roger's Pompidou Centre wherein the design infrastructure is exposed and exploited as a virtue so that the design exhibits an inside out aesthetic.

1.4.4.1 The Model as a Disciplinary Matrix

Of the many definitions of 'paradigm' found in Kuhn's work, the concept of the Disciplinary Matrix is probably the most accepted. A Disciplinary Matrix is a set of beliefs and values shared by a scientific community. These beliefs and values comprise an epistemology and language that delineates the community from other communities. The Disciplinary Matrix includes a belief in metaphysical and heuristic models.

Whilst the model advanced in this thesis is not the result of an explicit identification of elements of the DM (as Gallagher (1999) and Wernick (1995) have done) it does touch upon many of the (same) elements tangentially. For example, the concept of design context includes the influences of personal and professional experiences and organisational environment. In this sense then, it is claimed that the model advanced in this thesis makes a contribution to the establishment of a Disciplinary Matrix for software design.

1.4.4.2 The Model as a heuristic model

Dasgupta (1991:142) likened a design paradigm to a heuristic or metaphysical model in the sciences. It was not a full Kuhnian paradigm (as defined by a DM) but a subset of it. He did however acknowledge that members of the design community may "come to believe in or become committed to" a particular model and that such a model "may become the reason for an identifiable or distinct community to form". Wernick and Winder (1994:5) point out similarities between the Kuhnian Disciplinary Matrix and Dasgupta's Design Paradigm but "agree with Dasgupta that the connection between the two concepts is that a design paradigm will often form part of the disciplinary matrix of a particular discipline."

In this more limited sense, Dasgupta's design paradigm is "an abstract prescriptive model of the design process that serves as a useful schema for constructing practical design methods, procedures, and tools for conducting design" (1991:141). It is this interpretation - as a general framework informing and underpinning the development of real world software products - that is most often encountered in the literature (see for example, Jayaratna, 1994). The model advanced in this thesis is consistent with

this interpretation save for one crucial aspect - the model advanced here is purely *descriptive* seeking only to document design practice, not to advocate it.

1.5 The main argument: action in context and context in action

Software design is described as a transactional system – a system of causal relationships, conditions and consequences. At the heart of this system lies action and interaction – strategies that designers use to manage, handle, carry out, or respond to design under a specific set of perceived conditions. The design context is often complex and much of what designers do when they design can be seen as a response to that complexity. Yet this is not software complexity, or not *only* software complexity (Brooks, 1975) but complexity in the entire design context including the personality of individual designers, communications between designer and users and between designers and designers, organisational culture and methods, and the complex interaction of these things. A holistic picture of software design is presented – one that is difficult to define and develop but one that is close to design practice.

1.6 Structure of the thesis

Chapter Two: Background to the Study

Two sources of background are given to the research. Firstly, a brief literature review identifies paradigms in software design. It is noted that the term paradigm is used inconsistently in the literature, in particular the distinction between a paradigm and process model is not clearly held. This chapter attempts to clarify the use of these terms and to establish a relationship between paradigm, process and software design environments, and to position the model developed here in a broader context. In addition, the value of description over enactment in process modelling is emphasised.

Chapter Three: The Research Method

In theory the research method should follow the research question (s). In this case the nature of the questions posed about the phenomenon - what is software design? what do software designers do when they design, and why? -suggested a qualitative approach to its study. In practice, although the researcher in his or her selection of research method may discriminate between alternatives according to pragmatic criteria (fitness for purpose, ease of use, social acceptability etc.) this process is not free of personal bias and prejudice. It is as well then for the researcher to state at the outset as much of his or her personal beliefs as are relevant in order that the reader is informed of the thinking that lies behind the choice of method.

The increased use of qualitative approaches in IS research is cited in justification of the method chosen. The grounded theory method (Glaser and Straus, 1967) is introduced and its process explained. Despite, or even because of, affording powerful data analysis techniques the method is not without its weaknesses. Some of these are reviewed and the case is made for a prescriptive application of the method. One such approach utilises a technique for data analysis called the paradigm model (Strauss and Corbin, 1990) and the chapter concludes with an explanation of this data analysis tool and its enhancement, the conditional matrix[2]. In this thesis, this combination of research tools is used not only to analyse the data but also to structure its presentation.

---

[2] Strauss and Corbin's description of their research tool as a paradigm model should not be confused with discussions of paradigms of software design elsewhere in this thesis or with the model developed as a result of this work.

Chapter Four: The Research Design

Bryman and Burgess (1994:3) note that the literature on qualitative research methods tends to focus on data collection over analysis. They advocate more and better expositions of data analysis, particularly autobiographical accounts of how the analysis was carried out in practice. Though this thesis goes some way along this path (primarily but not exclusively in its account of the Meta-process), it is not an easy path to follow. In qualitative research in general, and grounded theory in particular, data collection and analysis are so interdependent that it is almost impossible to write about one without also writing about the other. Notwithstanding this difficulty, a contribution to the literature is claimed in this respect.

Although the data collection process is described in some detail, greatest attention is given to data analysis and especially to the use of the paradigm model / conditional matrix. The question of validity and reliability of qualitative research is considered both in general and as it relates to this thesis. This serves to introduce a set of qualitative and quantitative reliability measures included in the Chapter Seven. The chapter also includes a discussion of the personal approach to data management taken during this phase of the research.

Chapter Five: Action in context – an inductive model of software design

An inductive model of software design is developed based on an analysis of sixteen interviews with Software Engineers and Graphic Designers active in the field of Digital Interactive Multimedia. Each of the essential elements of Strauss and Corbin's (1990) paradigm model - causal conditions, context, strategies, consequences and intervening conditions is described and then integrated into a holistic explanation of the observed phenomenon. This chapter is the kernel of the thesis. The description of

software design advanced here is the basis for the claim to have developed an inductive model of software design, first put forward in Chapter One and underpinning the theoretical development in Chapter Six.

Chapter Six: Context in action: towards a theory of software design

Whereas Chapter Five followed the method closely- indexing the data and presenting it in narrative form- this chapter cuts loose in an attempt to exploit the power of the generative method. The chapter discusses the core category of context-*complexity* that is significant in its explanatory power over the phenomenon. The analysis follows a different, and more difficult direction, slicing across the data set to build a composite picture of the category whilst also seeking to minimise the loss of contextual detail that such an approach usually entails. The outcome of this chapter is a set of theoretical propositions given specificity earlier in the theoretical framework.

The Conditional Matrix introduced in Chapter Three is used to build a more detailed description of software design, and to develop the theory. According to Strauss and Corbin, (1990) the conditional matrix is a "framework that denotes a complex web of interrelated conditions, action/interaction and consequences" (:161) and is predicated on the notion that grounded theory is a transactional system - " a system of analysis that examines action/interaction in relationship to their conditions and consequences" (:158) Here, a simplified matrix is presented as the number of levels within the framework is reduced from eight to four. At each level -action, interaction, individual conditions and organisational conditions - the strategies used to design are described and the important conditions shaping those strategies explained. The Conditional Matrix is enacted through the tracing of conditional paths and this is done in data collected on a medium sized Multimedia development project.

Chapter Seven: Further validation of the model and theory

In this chapter both the model and theoretical framework are validated in the analysis of another dataset. This is comprised of sixteen interviews sampled from those reported by Lammers in 1989. Two types of validation are sought. Internal validation refers to the accuracy of the data (does it reflect what is really going on in software design practice?). External validity, or generalisability, seeks to establish whether the results are applicable to another, different, dataset. With this analysis the study moves from design in the small to design in the large specifically focussing on the core category of context-complexity.

Chapter Eight: Conclusions and recommendations for further research

Glaser and Strauss (1967) encourage the integration of emerging theory with existing theory found in the literature. At this juncture, the research findings are considered in the context of Newell and Simon's (1972) general problem solving theory and in the context of other theories of design. The purpose of this comparison is to ground the inductive model in the general literature and thereby increase its relevance. The chapter includes an evaluation of the work based on criteria set out by Strauss and Corbin (1990) and concludes with some recommendations for further research.

Appendices

The Appendices afford further insight into the research process that resulted in the model and theory. These need not be listed here but are readily appraised by their listing in the Table of Contents.

1.7 A note on thesis presentation

Strauss and Corbin (1990) point out the value of analogy in presenting a grounded theory study. These include looking at a statute (Glasser 1978) and walking around a house (Strauss, 1987). It may assist the reader of this thesis to consider the thesis as itself a design, its outcomes structured according to a generic problem solving paradigm that describes its production. Chapters 1 – 4 identify and scope the problem and set out the means of its resolution [Analysis phase]; Chapters 5 and 6 develop the inductive model and the theoretical framework [Design build phase]; Chapter Seven validates the model and theory in another set of interviews [Design test phase] and Chapter Eight considers the implications of the work [Implementation phase]. Indeed the value of this analogy is greater than simple process comparisons suggest. As in the design it describes, the actual process was much more complex than that readily apparent in theories or models of that process.

The structure of the thesis permits multiple user or reader views, and need not be read in its entirety. Those readers interested only in the method are directed to Chapters One, Three, Four and Seven. Those readers concerned only with the implications of this thesis for the field of software design are directed to Chapters One, Two, Five, Six, and Eight. Table 1 (on the next page) summarises the inputs, processes and outputs of each chapter and these two routes through the thesis are identified by the use of shading.

Table 1: The development of the model and theory (overview of the thesis)

Chapter Two: Background to the main study [Research Questions 1 and 2]		
Inputs →	**Processes →**	**Outputs**
The general literature	Brief literature review. Identification of salient models and theories	Rationale for the study, Problem statement

Chapter Three: The Research Method		
Inputs →	**Processes →**	**Outputs**
Personal epistemology Glaser and Strauss (1967) Strauss and Corbin (1990)	Identification, clarification and evaluation of the research method used in this study.	Introduction and overview of the Grounded Theory method. Definition and description of the Paradigm Model and Conditional Matrix

Chapter Four: The Research Design		
Inputs →	**Processes →**	**Outputs**
Data-set A Data-set B	Detailed description of the processes used to reach, and justify, the findings of this thesis	Identification of data sources. Explication of data analysis processes including verification

Chapter Five: Action in context: an inductive model of software design [RQ1 and RQ2]		
Inputs →	**Processes →**	**Outputs**
Data-set A Strauss and Corbin (1990)	Open and axial coding of sixteen interviews with software designers. Identification of significant categories and of the relationships between these, using the Paradigm Model	An inductive model of software design

Chapter Six: Context in Action: towards a theory of software design [RQ1 and RQ2]		
Inputs →	**Processes →**	**Outputs**
Data-set A Strauss and Corbin (1990) Brooks (1975)	Selective coding of Data-set A, to identify and develop a core category that explains the phenomenon.	Theoretical framework. Theory stated as set of propositions

Chapter Seven: Further validation of the model and theory		
Inputs →	**Processes →**	**Outputs**
Data-set A Data-set B	Internal and external validation of outcomes using qualitative and quantitative tests	Partially validated model and theory

Chapter Eight: Conclusions and Recommendations for further research [RQ3]		
Inputs →	**Processes →**	**Outputs**
Research outcomes Literature of related work Strauss and Corbin (1990)	Conclusions, grounding research in existing theories of design and problem solving, evaluation of process and outcomes.	Conclusions Contribution Evaluation Recommendations for further research

1.8 Chapter Conclusion

This chapter is a synopsis of the entire thesis. The origins, nature and purpose of the investigation have been set out, including an outline of the research method. The major outcomes of the research have been identified explicitly as claims made of the thesis. In Chapter Eight these claims will be reconsidered in light of the presentation of process and outcomes, as thesis contributions.

Chapter Two: Background to the main study

2.1 Introduction

Since this thesis purports to develop both a model and theory of software design some definition and description of these terms is necessary at the outset. In particular it is necessary to reflect upon the relationships between the two, to distinguish between paradigm and process models and to establish a relationship between such models and practice. A citation driven review of literature then identifies some models and theories in the field of software design. In particular, the concept of design breakdowns is identified as one potential contribution of the thesis, and this is returned to in Chapter Eight (8.3 –8.4) in the discussion of the outcomes of this research.

2.2 On Paradigms, Processes, Models and Methods

2.2.1 A taxonomy

Van Gigsch and Pipino (1986:73) developed a hierarchical framework to investigate the discipline of information systems, based on the theory of multi-level systems (Mesarovic, Macko and Takahara 1970) and on the metasystems approach (Kickert and van Gigsch, 1979). The framework consists of "three basic levels which represent the discipline of IS and its respective inquiring systems"

The practice of IS "involves all those activities by which the theory, models, technology, in short, the state of the art of the IS discipline, are applied to the real world of organisations and systems". The science of IS is defined as "those activities by which the theory and models used to describe, explain and predict the behaviour of IS are developed". The epistemology of IS "involves the activities of inquiry which

seek to define the origin of knowledge of the discipline, to justify its methods of reasoning and to enunciate its methodology" (1986:72-73). These levels have been organised by type of inquiring system, respectively, the lower level of inquiry, the object level and the metalevel.

Level of inquiry	Inputs	Inquiring system	Outputs
Metalevel	Philosophy of science	Epistemology of IS	Paradigm
Object level	Paradigm from Metalevel and evidence from lower level	Science of IS	Theories and models of IS
Lower level	Models and Methods from object level and problems from lower level	Practice of IS	Solution to IS problems

Table 2: Van Gigch and Pipino's (1986) hierarchy of inquiring systems

Paradigms originate at the metalevel (Epistemology of IS), based on philosophical beliefs which embody the commitment shared by a particular community to accept a specific approach to scientific inquiry. These paradigms inform the object level (Science of IS) where along with evidence from the lower level (Practice of IS), theories and models are formulated and developed. Finally, at the lower level (Practice of IS) models and methods are applied to organisational problems.

Van Gigch and Pipino (1986:74-76) used their framework to classify papers published in academic journals "as indicators of the research emphasis and interests" of the IS discipline. They found that "the largest proportion of papers reflects work in the Practice and in the Science of IS, with a relatively smaller proportion classified at the epistemology level", agreeing with an earlier survey by Le Moigne (1985). They call for more work concerning the Epistemology and Science of IS.

However their framework is itself a theory, a meta theory of the role of theory, models and methods in the development of a discipline. What evidence is there that theories,

models and methods do indeed inform the way problems are solved? In the discipline of Information Systems van Gigch and Pipino suggest that too many models and methods are conceived of at too low a level of abstraction, the generality of most new models is suspect, hypotheses have limited scope and case studies have limited generality. It is not the purpose of this chapter to replicate van Gigch and Pipino's study in the field of software design. Rather, at the object or science level of inquiry, to consider "those activities by which the theories and models used to describe, explain and predict the behaviour of [Information] systems are developed" (ibid:72) and to evaluate the outcomes of such activities – in the context of software design.

Before doing so, it is worth pointing out that it is not necessary for software design to be considered a discipline (defined here in the Kuhnian sense of a community united by a common paradigm) in order to identify models within the field. Indeed, it is the case that any field that exists in a pre-scientific state - where there is no single unifying paradigm but the field is divided into a number of competing schools of thought - will exhibit more, theories, models and methods than a field considered to be in a normal science state. These will be less developed and will be distributed amongst the various schools of thought, each coalescing around their own paradigm. This thesis makes no claim based on empirical data for or against software design being a scientific discipline but observes that a single unifying paradigm is highly unlikely given the immaturity of the field. In addition it is noted that referent fields that inform software design are themselves the subject of debate concerning their claims to be 'scientific' disciplines (Gallagher, 1999 provides a useful summary of the relevant literature).

Nor does the probable pre-science state of the field rule out the identification of a community of software design. Although Kuhn equated community with paradigm his thoughts on this are ambivalent and he does recommend looking for a community first (Kuhn, 1970:176). Therefore it is not inconsistent to say on the one hand that it is highly unlikely that the field of software design could be considered a scientific discipline and on the other to search for theories, models and methods within the field based on an identifiable community.

Kuhn is quite specific on what constitutes a scientific community and how it may be identified. A scientific community is a distinct group of people united by similar educational experiences (Kuhn, 1970:177). The community tends to be small (25-100 people), and where larger groups exist, these naturally sub-divide into smaller groups. Communities can exist at numerous levels (Kuhn, 1970:177, 1977:296). Thus software design could be considered as part of a larger global community of design, which itself contains many sub-communities including software engineering and graphic design. Communities may be identified by the existence of scientific journals, conferences, textbooks and professional societies but also by formal and informal communication networks and citations. Of these textbooks are of key importance since "they expound the body of accepted theory" and form "the basis of a new tradition of normal science" (Kuhn, 1970:10,144).

2.2.2 Paradigm and Process Models

Van Gigsh and Pipino's framework provides a mechanism to identify and classify approaches within software design. Although paradigms can be identified in the literature (section 2.3 below) the boundaries between paradigm and process models are often blurred.

The predominant approach to software development is the life cycle model, or variations of it (Pressman, 1982). This model is based on a phased refinement

approach wherein systems functionality is specified early in the development process and subsequent phases add design detail. Although much criticised in its native form, the system development life cycle model underpins most of the more popular software development approaches including the waterfall, spiral, evolutionary and prototyping approaches.

But is the SDLC a paradigm or process or both? Pressman identified the generic software development process to consist of definition, development and maintenance. The systems development life cycle (and variations of it, noted above) is then a specific approach to this generic process. A similar position is taken by Agresti (1986:12) who makes a distinction between life cycle models and a "base development process for software development". Life Cycle models should be interpreted as only one class of response to a more fundamental concern – the development of an effective (flexible) process model for software development.

Elsewhere a process model is more specific and detailed. "A process model is a specification of a real world software process. A specification aimed at describing precisely and unambiguously, the requirements, the design and the implementation of a software system"(Jaccheri, Pico and Lago,1998:369). "A process model represents the knowledge of a specific process, including the process steps, their prerequisites and consequences and any synchronisation among concurrent steps defined in some process modelling language" (Israel Ben-Shaul, 1995:xii -3). It is "the orderly approach to applying methods and tools to software development" and includes activities carried out during development (both high and low level), local and global constraints on activities, partial ordering among tasks and synchronisation among current tasks.

A process model describes what will be done, how it will be accomplished, when it will be finished and who will use what to implement it (Charette (1986:38-40). A process is "the basic function to describe the chain of events required to create a particular software products" and a software engineering environment is "the process, methods and automation required to produce a software system".

Thus, in the literature a process model is used to describe the generic process of software development or to describe a real software process in some detail. Here van Gigsch and Pipino's framework is of some assistance. At the highest level of abstraction the generic software process describes the fundamental steps in the development of software. This model reflects basic beliefs and assumptions about the development of software but is non-prescriptive in that it does not advocate a particular approach to development. At the lower level, other process models (including the waterfall, prototyping and evolutionary) are prescriptive and detailed, but in varying degrees.

The model advanced in this thesis, developed at the object level of inquiry, is closer to the model at the higher level of abstraction. It seeks to describe the basic software design process and reflects the beliefs and values of practitioners in doing so. As such it reflects a Kuhnian paradigm. It makes no attempt to advocate a particular approach to software design (unlike Dasgupta's model) and does not map out in any detail the implications for particular approaches in particular instances (although a causal relationship between paradigm, process and methods is further explored below and in Chapter Eight).

2.2.3 Beyond process models to Software Development Environments (SDEs)

Goedicke (1990:1-2) takes as his starting point the original Greek meaning of the word paradigm– pattern, typical event or archetype – and goes on to set out the criteria for a software development paradigm. It must define a few basic steps (he identifies divide and conquer and separation of concerns), provide adequate descriptive means for defining the properties of a particular system in terms of the basic concepts (ie a language), provide guidelines which help to carry through certain development steps to yield the desired running system [methods], and provide (hopefully automated) tools to support the chosen method. This definition of paradigm goes much further than Dasgupta's (1991), to include elements of a process model and a software development environment.

According to Humphrey, (1989) Software Development Environments (SDEs) is a field of software engineering concerned with providing frameworks or infrastructures for supporting the development of software products. Within this field Software Processing Modelling (SPM) is concerned to understand the development process - a "way to represent the process with a formal notation in order to arrange and or partially automate support for it" (Ng and Yeh, 1990:3).

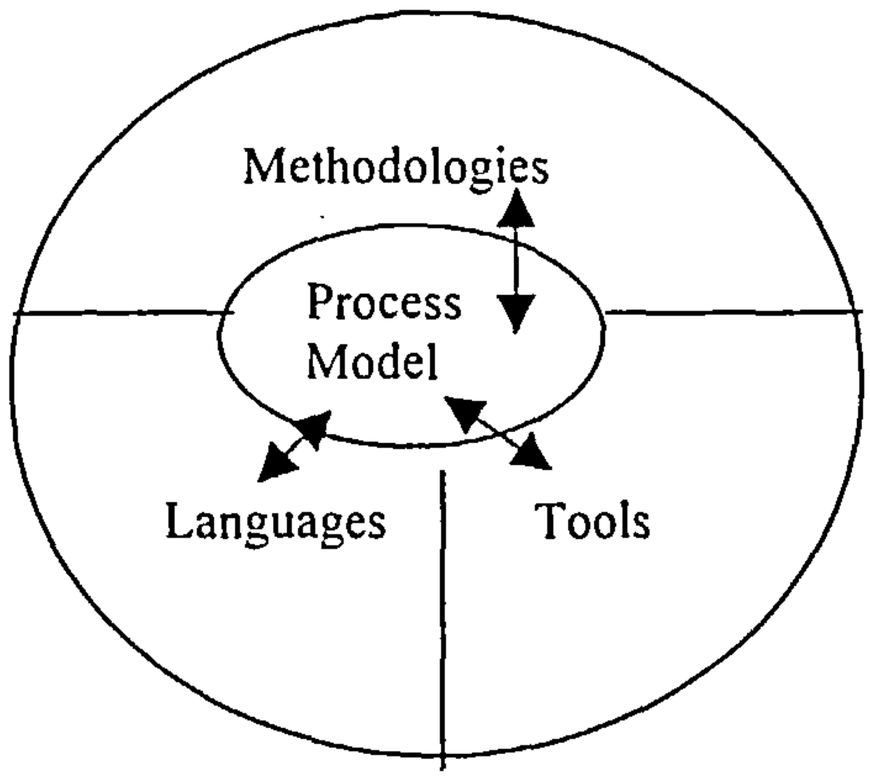


Figure 1: Elements of a software development environment (Ng and Yeh, 1990:13)

The process model informs methods, languages and tools. A method is a set of guidelines used to assist the development process. A method may act as a learning aid and facilitate communication and understanding within the development environment (Longren and Stolterman, 1999). Languages describe the functionality or specification of a system and include graphic languages, procedural languages, non procedural languages, functional languages, and object oriented languages. A software tool is any "program employed in the development, repair or enhancement of another program" (Dictionary of Computing, Oxford University Press, Third Edition, 1990) and therefore may be (inter alia) a text editor, a de-bugger, a user interface tool, a data management tool or a configuration management tool. In turn, the process model is supported and promulgated by particular methods, languages and tools. So for example SSADM both follows and develops the highly structured approach to software development inherent in the waterfall process model. The interdependence between all the elements of the software development environment is illustrated in the following comment about tools

> "the effectiveness of a computer-aided tool is dependent upon the environment of the user, in particular, the process model, the language, and the methodology that the user is using or with which the user is familiar" (Ng, 1990:1)

It is beyond the scope of this thesis to trace these relationships in any detail but ipso facto, their existence is significant to the relevance and contribution of the work. Chapter Eight will therefore discuss how the findings of this thesis may inform method development.

2.2.4 The value of descriptive models

Jaccheri et al, (1998) note that whilst the main goals of software process modelling are understanding, improvement and enactment, the software community has traditionally concentrated on improvement and enactment rather than description. They give as an example Process Centred Software Engineering Environments (PSEEs). A preoccupation with improvement and enactment is epitomised in management by Business Process Engineering (Hammer, 1995) and by quality improvements processes such as CMM and Bootstrap. Within Computer Science, the fields of requirements engineering and expert systems seek improvement, although in these, there is much greater attention paid to elicitation.

Jaccheri et al (1998:370) point out that the importance of process modelling for elicitation rather than enactment has been recognised and that

> "case studies have argued that the elicitation of a process model can be useful per se, even when the goal is not an enactable model but *rather the creation of a process description to be used in the context of an improvement strategy*". [my emphasis]

One goal of this thesis is a descriptive model of software design. No attempt is made to prescribe a particular process or to advocate the use of specific methods or tools.

2.3 The Meta-process

A Meta-process is the process of creating process models and process models will have different purposes depending on the Meta-process that produces it. Moreover

> "the elicitation of the software process model of an organisation is often the very first attempt to define it, trying to build a description that is as close as possible to how the real process is

> carried out" and that "consequently, during this activity, process information is gathered from several sources which is often incomplete, inconsistent and ambiguous" (Jacheri et al:1998:370-371)

Bandinelli, Fuggetta, Lavazza, Loi and Picco (1995) describe the key stages of such a process

(1) modeller describes information gathered by means of model fragments (data extracts)

(2) fragments are refined gradually - verified by modeller with model user

(3) model fragments are integrated to derive formalised model of process.

This is a reasonable description of the meta process followed in this research and reported in detail in Chapter Four, with the exception that verification was limited to the data collection phase in the first instance. However the model was subsequently tested with a second data set (Chapter Four, 4.3.6, 4.4 and Chapter Seven).

Whilst the primary goal of this thesis is to say something substantial (original and relevant) about the field of study, an important secondary goal is to explicate the research method used so that it may become more accessible to other IS researchers. This objective profoundly influences the structure and presentation of the thesis because it requires discussion of the research method to be thorough, transparent and pervasive. When the chosen method is grounded theory this becomes a significant challenge. Bryman and Burgess (1994) observe that

> "very often, the term [grounded theory] is employed in research publications to denote an approach to data analysis in which theory has emerged from data. Rarely is there a genuine interweaving of data collection and theorising of the kind advocated by Glaser and Strauss. As a result grounded theory is probably given lip service to a greater degree than is appreciated. Richards and Richards (1991:43) make a similar point when they observe that grounded theory 'is widely adopted as an approving bumper sticker in qualitative studies'. *Moreover the precise process whereby a grounded theory analysis was undertaken is often imprecise*" (:6) [my emphasis]

Yet it is unlikely that researchers have been deliberately evasive in this respect. Although some have undoubtedly used the method loosely, most researchers have sought to apply it diligently. That they have not been able to describe their methods in a way that makes them accessible to others may be due to a lack of clarity in

qualitative approach in general, and of grounded theory in particular, rather than any shortcomings on the part of individual researchers. As Bryman and Burgess (1994) note there are few personal accounts of how qualitative research is carried out (indeed this was the motivation for their collection of essays). Unlike quantitative research, qualitative research remains highly idiosyncratic. The coding of interview transcripts for example cannot itself be codified in the manner of quantitative techniques but must remain flexible in order to handle the nuances of the data as it unfurls. Despite attempts to make the 'doing' of grounded theory research more explicit (most notably by Strauss and Corbin, 1990, 1997) the actual process remains somewhat of a black art. Personal accounts of its application help but are currently too few and isolated to allow any general principles to be developed.

It remains a paradox of the grounded theory literature that those accounts that seek to lay bare the application of the method are, with a few exceptions, disappointing in their analysis (see for example, Konechi's account in Strauss and Corbin's (1997:131-146) book of 'exemplars'). While those that that present tightly interwoven data and conceptual depth in their analysis are not good at explaining how this was done. It is as if in the very act of documenting the application of the method its essential value is lost. At the very least this indicates that there are real problems in making transparent the myriad cognitive and reflective processes that take place during a grounded theory study.

The analogy with design is striking. The designer cannot easily convey or document how they design. Aside from obvious time limitations, they have no mechanism, no language even, to express the detail of the activity they are engaged in. It is left to others to try and explain what is going on and inevitably, these accounts are second

hand and second best. There are a few good accounts of the design process (Schon (1983) is an outstanding example) but even these do not make known the precise method of data collection and analysis. As with many good grounded theory studies, the reader is given little insight into the mechanics of the research process and left with few clues as to how such research may be replicated[3].

No particular claim is made of this thesis to succeed in the challenge of making the method accessible where many other, more competent, researchers have failed. A contribution to the literature is claimed however by making the intention to do so clear at the outset and by accounting for the success or otherwise of the attempt at the conclusion of the thesis (Chapter Eight, 8.5.3). Bryman and Burgess (1994:5) note the increasing popularity of autobiographical accounts of research practice wherein the authors discuss "the ways in which they actually conducted their research, *in contrast to the ways they might be supposed to conduct it*" (my emphasis)[4].

2.4 Identifying models and theories in software design - a brief review of the literature

A simple chronology of the development of the field of software design (see Table 3) indicates that whilst it came to prominence as an aspect of software in the 1990's, it antecedents can be found as far back as the mid-seventies. However, although the twenty year period between the publication of Fred Brook's Mythical Man Month in 1975 and its re-publication in 1995 neatly encapsulates the conception and birth of the discipline, the character of the discipline today is not shaped only by events during

---

[3] Protocol Analysis (discussed in Chapter Three) remains a general exception to this rule.
[4] Since beginning this research I have become aware of more and better studies in IS research. These are discussed in Chapter Eight (8.4.3)

that time. Indeed, the origins of "software design" may be traced back to the nineteen fifties.

1975: *The Mythical Man Month* is published
1990: Mitch Kapor delivers his *Software Design Manifesto*
1992: The Association of Software Design is formed
1992: Workshop on software design held at Stanford University
1993: ACM publish *Interactions*
1994: Special issue of *Human Computer Interaction* devoted to context in design
1994: SIGCHI devotes unprecedented attention to software design issues
1995: *The Mythical Man Month* is re-published
1996: Publication of *Bringing Design to Software*

Table 3: A chronology in the evolution of the discipline (adapted from Winograd, 1996)

2.4.1 A starting point (Winograd, 1996)

Here the theoretical development of the discipline is traced using the citations in Winograd's 1996 text to discover and discuss other key contributions. This approach to literature searching is most appropriate to a new, emerging field where the literature is evolving rapidly (Howard and Sharp, 1983).

Winograd (1996:xvii) links software design to software engineering citing "the substantial body of literature on software design as an engineering activity" including Brooks, 1975; Pfleeger, 1987; Rumbaugh, 1991; Blum, 1992; Brooks, 1995; and Blum, 1996. Of these, Blum (1996) in particular "addresses software engineering concerns from a design perspective".

2.4.2 Adaptive design (Blum, 1996)

In fact, Blum explicitly rejects software engineering as a paradigm for software design. "The present approach to software development is based on a faulty model of reality; it ignores the special properties that software affords" (Blum:1996:4) because "we view the program as a product and not as its design" (:12). He calls for a

paradigm shift in software development. He does so, undoubtedly from a software engineering perspective (as he readily admits he is a software engineer by training and by inclination) but this should not detract from his central thesis - a revolution is required in how we develop software. He recognises that the design model (within software engineering) "has been extremely successful" (:17) but advocates an alternative approach which exploits the unique features of software, this he calls "adaptive design".

The essential difference between adaptive design and the traditional software engineering approach is that whereas software engineering is concerned with models 'in-the-computer', adaptive design is concerned with models 'in-the-world'. That is, in software engineering "we perform an analysis on the world and then specify an artifact...we then fabricate or code a product that satisfies the specification....the specification defines the behaviour of what should be in the computer, and the process terminates when a correct realisation of the specification exists in the computer" (Blum, 1996:16). On the other hand in adaptive design this two step process is replaced with a one step process that models what goes on in the computer *as it affects us in the real world.* "Thus, we have a contrast between the careful analysis of what is to be built followed by its construction (ie. technological design) and the one step model of continuing response to changing requirements (ie. design by doing or adaptive design)"(1996:17).

Blum refers to the 'underlying tension in the design process'. Technological design addresses a need that exists in the real world but this need must be expressed formally in a model in the computer. This tension, "cannot be avoided, we must live with it and understand it" (:102).

2.4.3 A design rationale (Carroll, 1997)

Winograd (1996:xix) also recognises the contribution literature in the field of human - computer interaction (HCI) has made to software design (Card et al, 1983; Norman and Draper, 1986; Helander, 1988, Caroll, 1991) and in particular the value of the "cognitive analysis of human-computer interaction".

In a later account, Carroll reviews the history of HCI "as a step toward a science of design" (Carroll, 1997:501) Using Simon's (1969) *The Sciences of the Artificial* as a 'touchstone', he argues that HCI is not merely applied psychology but that it has "guided and developed the basic science as much as it has taken direction from it" (Carroll, 1997:502). He identifies software psychology (Shneiderman, 1980) as "the work that constitutes the historical foundations of HCI" (ibid:502).

According to Carroll, software psychology, the goal of which was to understand software design from a behaviourist point of view, was predicated on two - ultimately fallacious - assumptions. Firstly the waterfall model was accepted as the received view of software development. Secondly it was believed that psychological research would translate readily into usability practice. He argues that by the end of the 1980's it was clear that both these assumptions were problematic. The weaknesses of the waterfall method (Brooks, 1975) were cruelly exposed by the spread of smaller, distributed and user-centred systems requiring iterative development to meet considerably compressed development cycles. Research that produced general descriptions of users, based on artificial and unrepresentative laboratory experiments, proved to be of limited value in the real world. Nevertheless, Carroll claims that during the 1980's the origins of HCI in software psychology posed two central

problems for the field "*to describe design and development better, and to understand how it can be supported*" (Carroll:1997:503).

At the same time there was an increase in the number of empirical studies undertaken. This addressed both weaknesses in the software psychology approach. The waterfall method was further undermined since it was now clear that designers often need to do design in order to understand it; Brooks' maxim of "plan one to throw away" had been "a striking lesson to draw and carried with it many implications" (ibid:503). More research based on studies of practitioners in the field, helped bridge the gap between academic research and usability specialists, providing a more realistic foundation for design guide lines. Carroll concludes "through the decade of the 1980's, the inevitability of an empirical orientation towards system and software design rapidly evolved from a somewhat revolutionary perspective to the establishment view" (Carroll:1997:504).

In arguing for HCI as a science of design, Carroll advocates the development of a design rationale. This embodies a broader view of the design and the design process by explicitly identifying the issues that arise during the design, the alternatives considered in response to these issues, the reasons for chosing a particular alternative, the weighing of trade-offs and so forth. This has lead, variously, to the explication of a design space (MacLean, Young, Belloti and Moran: 1991), the articulation of the social and behavioural theory implicit in a design (Carroll and Rosson, 1991) and the application of the notion of issue based information systems (Rittel and Weber, 1973; Conklin and Yakemovic (1991) and Carroll, Alpert, Karat, Van Deusen and Rosson, 1994).

A design rationale is "a tool for managing the complexity of a process in which everything including the problem definition, constantly changes" (Carroll, 1997:510). Moreover it "creates an explicit design representation, a *theory* of the artifact and its use" (Carroll, 1997:511). Carroll points out that this theory is not a classic user model, it does not purport to describe general information processes and cognitive structures but specific situations of use - making it more powerful within the immediate design context. However, he has been involved, with others, in developing schemes for generalising from such situated theory (Carroll, Singley and Rosson, 1992).

By 1990, says Carroll "there was a clear consensus that the cognitive modelling approach (to HCI) had failed to provide a comprehensive paradigm"(1997:511) and he cites as evidence that in 1990 and 1991 the major international conferences in HCI featured panels addressed to the failure of theory. At the same time "many voices suggested that a more socially or organisationally oriented approach was required to supplement or replace the cognitive paradigm" (1997:511). These included anthropologists and sociologists, concepts from work psychology, and activity theory, which focuses on how people can negotiate with the social and technological environment to solve problems and learn (Wertsch, 1985; Engestrom, 1993: Nardi, 1995). "All this should be seen as part of a larger paradigmatic restructuring of social and behavioural science: traditions that had sought to study individuals in isolation from their contexts and social phenomena in isolation from individuals were declining"(Carroll, 1997:512).

2.4.4 A communications paradigm (Winograd and Flores, 1987)

Winograd and Flores propose an alternative design paradigm to that based on rationalist decision making. They reject Simon's (1972) model (except in narrow

domains with well-defined problem spaces) arguing that it fails to take sufficient account of context (even the concept of bounded rationality is deficient in this respect). Actual decision making is frequently irrational. One cannot avoid actions, one is thrown into them and even doing nothing has consequences (their concept of "throwness" is derived from the Heidiggerian view of "being in the world" and entails a 'transparency' or 'readiness to hand' that determines design activity); context makes structure and rationality less relevant. The crucial part of problem solving lies in the formulation of the problem and

> "much of what is called problem solving does not deal with situations of irresolution but takes place within a normal state of resolution...resolution concerns the exploitation of the situation, not the application of habitual means" (Winograd and Flores, 1987:150)

Design is not the result of individual cognitive operations in an objective world but participation in mutual behaviour in which language is essential. Here language is much more than description, it is a social act in which commitments are sought and given and is therefore essential to human discourse (1987:176). Language – and thought – are built on social interaction and knowledge is socially constructed. (1987:78);

> "every manager (designer) is primarily concerned with generating and maintaining a network of *conversations for action*, conversations in which requests and commitments lead to the successful completion of work" (Winograd and Flores, 1987:144)

However breakdowns in communication are inevitable, designers need to understand breakdowns, try to anticipate them and benefit from them in the form of self reflection. Breakdowns -along with language (shared commitments) pre-understanding, knowledge, experience and interpretation – shape the design context

> "The analysis of a human context of activity can begin with an analysis of the domains of breakdowns, and that in turn can be used to generate the objects, properties and actions that make up the domain" (Winograd and Flores, 1987:17)

But designers are also concerned to shape the context in which future action takes place, this they refer to as *conversations for possibilities* (1987:151). Chapter Eight (8.3) returns to this text to discuss the outcomes of this research study.

2.4.5 Discussion

Software design sits at the confluence of software engineering, HCI and design. Flows within and between the referent disciplines change the shape and character of the target discipline. These are (inter alia) the failure of traditional theory, a move from prescription to description, an increased emphasis on empirical studies, a shift from product to process focus, emphasis on situated or socially mediated design, the concept of design breakdowns, the importance of context and responsiveness to it. None of this is unique to software design. On the contrary, even a limited review of the general literature on design quickly suggests that such issues have much wider resonance. Indeed that their identification in software design is merely an instantiation of a general class of design problem. For example, the paradigm models put forward by Blum (1996) and Winograd and Flores (1987) are based on the philosophy of Martin Heiddeger that underpins Schon's (1983) seminal observations of design practice. The view of software design as a trial and error process can be identified in the work of, for example, Petroski, 1985; Rowe, 1987; and Lawson, 1997.

The relevance of such themes for this thesis is significant. The model of software design advanced here is predicated on the nature and role of design context and its impact on design strategies and their outcomes. Although no attempt was made to comprehensively cover the literature prior to this study (see Chapter Three, 3.3.2.1) the outcomes of this work prove to be consistent with existing theories and models of design. The relationship between the outcomes of this research and underlying theories of design and problem solving is explored in Chapter Eight. However no specific associations were planned, nor are now claimed, *a priori*.

2.5 The sampled sub discipline

Gallagher (1999) identified two major paradigms in Digital Interactive Multimedia (DIMM) - Software Engineering and Graphic Design - but acknowledged that others such as book production and film production are also significant (Webb, 1996). Within and between these major paradigms, and amongst others in the field, it is possible to identify approaches also identified at the level of the super-ordinate discipline. Multimedia development consists of a small number of generic steps - proposal, design and production – analogous to the analysis and design phases of the

software development life cycle (Bunzel and Morris, 1992, Vaughan, 1994); it involves iterative development (Ambron, 1990; Luther, 1994); prototyping (Cotton and Oliver, 1993; Fisher, 1994); and top down and bottom up design (Bunzel and Morris, 1992).

Since the purpose of this research was to generate an inductive model of software design, based not on existing theories but on an analysis of interviews with software engineers and graphic designers in the field of DIMM, it would be inconsistent to include a detailed account of these approaches at this juncture. The interested reader is referred to the general literature sampled above or to Gallagher's thesis or to two papers based on that research (Gallagher and Webb, 1997, Gallagher and Webb, 2000). Chapter Eight compares the outcomes of this inductive analysis with general theories of problem solving and of design.

2.6 Chapter Conclusion

Notwithstanding the nature of the research method, some theoretical background to the main study has been given. The relationship between theory, models and practice has been highlighted, as has the value of descriptive models and of the explication of the Meta-process. A brief review of the literature has identified some relevant models and theories of software design. In particular the concept of design breakdowns has been flagged as one potential contribution of the thesis. This is returned to in Chapter Eight (8.3 and 8.4).

Chapter Three: The Research Method

3.1 Introduction

Before setting out in detail the research process through which the data are further analysed and initial hypotheses are proved or disproved, confirmed or unconfirmed, a rationale and background to the chosen research method is given. The personal beliefs of the researcher that informed the choice and deployment of the method are given (consistent with a claimed contribution of the thesis) and a case is made for the use of qualitative methods in IS research. The chosen research method – Grounded Theory – is introduced and one data analysis technique – the Paradigm Model – is discussed in some detail. Finally the method is evaluated and an alternative approach – protocol analysis – considered.

3.2 An epistemological starting point

> "Behind every method lies a belief. Researchers must have a theory of reality and of how that reality might surrender itself to their knowledge-seeking efforts. These epistemological fundamentals are subject to debate but not to ultimate proof. Each epistemology implies a set of methods uniquely suited to it, and these methods will render the qualities of data that reflect a researcher's assessment of what is vital. I believe that researchers ought to indicate something about their beliefs, so that readers can have access to the intellectual choices that are embedded in the research effort" (Zuboff, 1988:423)

Zuboff's belief, her epistemological perspective, was that "feelings are the body's version of the situation". She used this belief, tempered by the "rigorous and systematic approach of a social scientist", to research human experience in dealing with new technology. I wish then to state the beliefs that have influenced my choice of research method.

- It is possible (and often, productive) to adopt a pragmatic approach to the selection of research method. The purpose of research is to study a phenomenon, gather some data and make sense of it. This activity permits a variety of approaches, methods and techniques from different epistemological and

ontological traditions. A method or set of methods may be associated with a given epistemological position but should not be tied to it; I do not need to be a phenomenologist to engage in social observation, nor an inductivist to use grounded theory.

- One can adopt and use a research method without necessarily signing up to the epistemological and ontological baggage that comes along with it. I have chosen the grounded theory method because of its intensive data collection and analysis techniques. I do not have to be an anti-positivist or a nominalist to support this position. Although a given epistemology may have certain methods more suited to it than others, method and epistemology are independent. Thus nomothetic methods can be associated with positivist epistemology and ideographic methods with an anti-positivistic epistemology.
- It is possible to believe in pluralism of method but not of outcomes. In research 'there are many ways to skin the cat' and one of a number of research methods can be chosen depending on the values of the researcher and the nature of the phenomenon to be studied. One can accept this without taking the relativist position that all research is unique due to the subjectivity of researcher and subject. This anti-positivist position has nothing to do with plurality of method.
- The views expressed here could be called a post positivist position, i.e. a rejection of scientific method for the concept of socially constructed knowledge and the notion that there is no one correct method but many dependent upon the problem being studied and the information sought. But I do not wish to embrace pluralism at the expense of positivism. It is perhaps better to see all methods on a epistemological continuum ranging from positivism to anti-positivism. Apart from the extremes, most methods will be a mixture of different traditions

- The terms qualitative and quantitative research are often applied inconsistently and incorrectly with each defined as the antithesis of the other. Thus quantitative research is associated with positivist science and qualitative research with an anti-positivist or interpretative approach. But qualitative research is not unscientific *de jure*, but, like quantitative methods, may become so *de facto* because of the way in which it is conducted. Hence qualitative research may be executed in a scientific manner (when for example scientific rigour is applied to the data collection/analysis /verification process) or in a non-scientific or pseudo scientific manner (when for example, data sources are not checked or the researcher does not acknowledge personal bias).
- Qualitative research is not one method but many methods for data collection and analysis and these methods do not exclude the quantification of data (interviews, case studies and content analysis methods for example all yield valuable quantitative data). Equally, quantitative methods do not exclude qualitative data and may even support it as an aid to comparison and generalisation. The Grounded Theory method is commonly referred to as a qualitative approach but its originators (Glaser and Strauss, 1967) do not rule out the use of quantitative data, stating that the generation of theory is independent of the type of data used. Rather than view qualitative research as an alternative to quantitative research we should recognise its complementarity. Quantification is a proven method of data collection and analysis permitting easy generalisations and cross study comparisons. Qualitative methods capture a rich picture of individual or social behaviour in context and afford convincing explanations of observed phenomena. A good research design will often include both approaches but equally, a research design that relies primarily, or even exclusively on one approach, is not

necessarily bad. In this thesis qualitative data analysis is set out in Chapter Four, and used in Chapters Five and Six. In Chapter Seven, quantitative analysis is introduced to validate the outcomes.

- In practice both the researcher and the people being studied are objective and subjective and data, which is termed objective or subjective, can be collected in an objective or subjective manner. In order to better understand a social phenomenon, we need the actor's viewpoint (commonly, but mis-leadingly referred to as subjective data) and the observer's or researcher's viewpoint (commonly, but mis-leadingly, referred to as objective data). The first is necessary if a rich picture of the phenomena is to emerge, the second is necessary if some sense is to be made of all this data through hypotheses or generalisations. Sandstrom and Sandstrom (1995) prefer the emic - etic distinction which they hail as " a key breakthrough in social scientific research of recent decades" (Sandstrom and Sandstrom, 1995:171) The researcher in emic mode records information from the point of view of the people engaged in social behaviour under investigation (for example in a user satisfaction survey) while in etic mode the researcher is an outside observer who decides on the concepts and categories necessary for descriptive analysis (for example in a time motion study or event analysis). One perspective is not better than the other, and one does not have to be an anti-positivist to conduct emic research - for example, in linguistic analysis, emic data can be gathered with scientific rigour. However, Sandstrom and Sandstrom also note confusion in the literature surrounding these terms and a tendency for discussions to degenerate into the old objective versus subjective dichotomy.

In this research both perspectives are used. A paradigm model of software design is developed based on the views of the software designers (emic mode). I then develop the model further through a higher level of analysis building a richer picture of the phenomenon (etic mode). A practical difficulty is to make the reader aware of the particular perspective being discussed at any point in time. In this thesis this is done through the explicit identification of sources and the use of "context" paragraphs.

- Pure or radical induction - such as that held by some phenomenologists and naturalistic inquirers, the idea that one can approach a phenomenon free from existing theory - is impossible. Even if we seek to avoid the literature, none of us are free from the beliefs and values that bias our work (we are subjective beings). This is recognised in natural science where empiricism - man's senses are the source of all his knowledge - is tempered by common sense. As Churchman (1971) points out, empiricism was doubted by Descartes (1596-1650) and other rationalists of the seventeenth and eighteenth centuries, "based on the very simple idea that the senses could tell us false things" (Churchman, 1971:38).[5] Kuhn (1970) believed that there was no such thing as an objective observation, that all observations were theory laden and that the scientist's method as well as his history was influenced by the scientific paradigm in which he was working; and O'Hear (1989), while defending the empirical method, acknowledges that our observations may not be true and accurate but only our "epistemological starting-point" formed by our sensory selectivity to the world around us (O'Hear, 1989:96). Thus observation under "certain circumstances" and the control of those

[5] Indeed, throughout this period, an empiricist, one acting without theory but relying solely on observation or experience, was also known as a charlatan and a quack.

circumstances to exclude extraneous causes and events is a vital element of any empirical study.

- Induction by itself, however many valuable insights it may offer, is not enough. Early anthropologists and ethnographers discovered this when they amassed data but couldn't do anything with them. Pure induction is incapable of building or verifying laws or theories. The construction of theory is a logico-deductive process in which emergent theory is continually tested against the data. This is what happens in the natural sciences and (contrary to popular belief) in qualitative methods such as grounded theory.

3.2 The method in context – a brief review of the use of qualitative approaches in IS research

Hirschheim (1985) has provided a synopsis of the growth, dominance and partial eclipse of positivist science, in an effort to " expose some of the hidden assumptions which lie behind our conception of valid research and valid research methods" (Hirschheim, 1985:28). Drawing mostly upon sociological treatments, in particular the work of Burrell and Morgan (1979) in organisational behaviour, Hirschheim traces the development of positivism from the seventeenth century, although he acknowledges its origins in the works of the classical Greek philosophers. He identifies Positivism's key philosophical canons as

- Objectivism - the separation of the observer and the observed based on Descartes' notion of the separation of mind/soul from the physical world.
- Empiricism - experiences of the senses as the one true source of knowledge.

- Naturalism - all phenomena, including social phenomena, can be explained in terms of natural causes and laws.

Anti-positivism, which also can be traced back to the Greek philosophers, re-emerged in the latter part of the nineteenth century from a concern that Positivist science did not reflect real life; individuals could not be treated in isolation but understood in a social and cultural context. Anti-positivists rejected the possibility of a value free theory emerging from positivist science and asserted that, because man was a free being, he could not be studied using the same methods as the natural sciences.

During the 1920's, Positivism was re-vitalised through the work of a group known as the Vienna circle and came to be known as Logical Positivism, Neo-Positivism or Logical Empiricism. This was to become "the dominant epistemology of contemporary science" (Hirschheim, 1985:51). Logical Positivism augmented phenomenalism (experience as the only source of data) with physicalism (data can also come from the outside world, not just private experience). Thus knowledge was no longer regarded as individualistic but could be derived through inter-subjective agreement. There was also a move away from individual explanations or laws to "theoretical networks of knowledge statements linked together through deductive logic and grounded in direct observation" (ibid.:51). The scientific model became the hypothetico-deductive model. Malinowski and Radcliffe Browne were just two of the many who adopted and applied Logical Empiricism to the social sciences. More recently, Logical Empiricism has been criticised because it does not deliver a value free theory and does not resolve the limitations of induction. Finally, Hirschheim identifies a group whom he calls post-positivists that favour replacing positivist science with socially constructed knowledge and methodological pluralism.

Hirschheim's motivation, based on dissatisfaction with positivist methods in Information Systems, was to "provide an overview of the key epistemological issues facing information researchers.. something which is long overdue"(Hirschheim, 1985:37) and to encourage methodological pluralism regardless of epistemological biases.

Hirschheim and Klein (1989) identified four paradigms of IS development using Burrell and Morgan's (1979) framework developed in the context of organisational and social research. A paradigm is defined as "assumptions about knowledge and how it is acquired". Epistemological assumptions are "those associated with the way in which system developers acquire knowledge needed to design the system" and ontological assumptions are "those that relate to their view of the social and technical world". These sets of assumptions are then given two dimensions. A subjective - objective dimension and a order - conflict dimension. An objectivist applies " models and methods derived from the natural sciences to the study of human affairs" and "treats the social world as *if it* were the natural world". A subjectivist "seeks to understand social life by delving into the subjective experience of individuals" and is principally concerned with "understanding how the individual creates, modifies and interprets the world in which he or she finds himself/herself". An order or integrationist emphasises a social world "characterised by order, stability, integration, consensus and functional co-ordination" whereas a conflict or coercion view "stresses change, conflict and coercion" (Hirschheim and Klein, 1989:1199-1217). These dimensions are mapped onto the two assumptions to produce four paradigms shown in Figure 2.

order	Functionalism	Social Relativism
conflict	Radical Structuralism	Neo-Humanism
	objective	**subjective**

Figure 2: Hirschheim and Klein's (1989) four paradigms of IS paradigms

These paradigms manifest themselves in the domain of Information Systems but are difficult to identify because they "are largely implicit and deeply rooted in the web of common sense beliefs which serve as implicit theories of action". Therefore, Hirschheim and Klein use generic story types "derived by interpreting pools of systems development literature that share the assumption of a particular paradigm" to illustrate each paradigm. They conclude that although different paradigms may co-exist within a school, one paradigm will tend to dominate and that "currently most research is focused only in the functional paradigm". However " although there is a strong orthodox approach to systems development, there are recently developed alternatives that are based on fundamentally different sets of assumptions". (Hirschheim and Klein, 1989:1199-1217)

Iivari (1991) identified seven major schools of thought in IS development - software engineering, database management, management information systems, decision support systems, implementation research, the socio-technical approach and the infological approach - based on the Kuhnian concept of the institutionalisation of the school in the scientific community, and on the existence of founders and followers. His analysis is based on a distinction between the ontology, epistemology, methodology and ethics of the research. Ontology "studies the assumptions made about the phenomena to be investigated' (Iivari, 1989:255); epistemology "concerns

the nature of scientific knowledge about the phenomena to be investigated" and methodology " is used in its original meaning to refer to the study of research methods' (ibid.:257)

All seven schools have similar assumptions. These include a positivistic epistemology, (only DSS had some anti-positivist tendencies), a view of information systems as a technical artefact with social implications and a structural view of organisations. Iivari concludes that the study supports Hirschheim and Klein's claim for an identifiable orthodoxy in IS development, "even though there is a certain variation between schools".

Morrison and George (1995) in their survey of research activity in the hybrid field of MIS/software engineering (which is defined as "all aspects of IS development from an organisational perspective"), found "many research methodologies used in social science research (such as laboratory experiments, case studies and field studies) are also important to MIS/SE research" (Morrison and George, 1995:90).

Davies and Myers (1994) acknowledge that "despite a clear historical preference for information researchers to adopt a classically scientific view of knowledge generation" (Davies and Myers, 1994:226), qualitative methods are becoming more popular particularly in the field of GDSS / CSCW. They emphasise the importance of context in qualitative research and point to the growth of ethnography in the study of IS in organisations including studies on IS development (Orlikowski, 1991; Orlikowski and Robey, 1991, Hirschheim and Newman, 1991), IT management (Davies, 1991) and transformation of organisational work through computer mediation (Zuboff, 1988).

Qualitative methods have also been used to discuss the epistemology of IS (Van Gigch and Pipino, 1986; Farhoomand, 1987; Banville and Landry, 1989; Hirschheim, 1989; Hirschheim and Klein, 1989; Iivari, 1991) and to combine discussion of epistemology with methodological development. Checkland (1981) has provided both a critique of the scientific method as applied to human activity systems and, in his Soft Systems Methodology, a qualitative approach to systems design. (Later he is concerned to place SSM firmly in the Phenomenological tradition rather than in the structural functionalism school where systems and systems thinking have traditionally been placed).

A summary of these and other approaches is found in Table 4 (on the next page).

Author/year	Key Words	Approach	Method(s)
Iivari, J. (1991)	IS development, Paradigms, Epistemology Ontology Methodology Ethics	Empirical, Epistemological Sociological	Content analysis of textbooks to analyse paradigmatic assumptions. Details of textbook analysis given elsewhere Iivari (1991)
Van Gigch, J.P Pipino, L.L. (1986)	IS Epistemology Paradigm, Kuhn	Empistemological Empirical	Literature (articles) review. Concludes that most papers at practice and science level, and that we need more at the epistemological level.
Banville, C; Landry, M. (1989)	MIS Whitley Kuhn (anti paradigm model)	Sociological, non-empirical, argumentative ?	Use of Whitley's (1984) model based on sociology of work organisations to classify disciplines. Emphasis on research rather than textbooks.
Farhoomand, A.F. (1987)	MIS, (Popper), Kuhn,	Empirical, Content Analysis	Thematic analysis of research strategies in 536 articles, 1977-1985
Hirscheim, R.A. (1985)	Epistemology, Scientific method, Positivism (we can study human phenomena using same methods we use in studying natural phen.)	Anti-positivism or Post- positivism influenced treatment. Non - empirical (in positivist sense).	Uses framework from organisational behaviour to structure an historical account of social science epistemology
Hirschheim & Klein (1989)	Paradigm Epistemology v Ontology Subjective v Objective Order v Conflict Functionalism Radical Structuralism Social Relativism Neo - Humanism	Sociological	Literature review Case study Analysis using Burrell and Morgan's (1979) framework developed for organisational & social research.

Table 4: A selection of qualitative approaches to research

3.4 Grounded Theory: introduction and overview

3.4.1 Introduction

The Grounded Theory method was developed in the mid 1960's by two sociologists at the University of California (Barney Glaser and Anselm Strauss) while studying the interactions of hospital personnel with dying patients (Glaser and Strauss, 1965). It was subsequently extended and refined both by the originators (Glaser and Strauss, 1967; Glaser, 1978, 1992; Strauss, 1987; Strauss and Corbin, 1990, 1997) and their students (for example, Charmaz, 1983, Stern 1994). The method has been widely used in social work and the social sciences and more recently has been used in the field of organisational research (for example, Sutton, 1987; Ancona 1990; Isabella, 1990; Kahn, 1990; Pettigrew, 1990; Elsbach and Sutton, 1992) and management (Giola and Chittipedi, 1991; Gersick, 1994; Hunt and Ropo, 1995; Brown and Eisenhardt, 1997; and Partington, 1997). Its use within the field of Information Systems research is much less documented, a notable exception being Orlikowski's (1993) study of CASE adoption in two organisations.

The approach to grounded theory development used in this study is based on Strauss and Corbin's (1990) book *Basics of Qualitative Research*. This text takes a highly prescriptive approach to theory development arguing that this is essential if novice researchers are to master an otherwise complex and idiosyncratic research method. Therefore the authors are concerned to "spell out the procedures and techniques in greatest detail" and "in a step by step fashion" (Strauss and Corbin, 1990:8). Whilst this approach has been criticised for departing from the original ideas put forward by Glaser and Strauss in 1967 (and there is evidence from his subsequent writings that

Glaser in particular was not keen on grounded theorists following an orthodox and authorised approach (Glaser, 1978)), it was readily adopted by this researcher.

"A grounded theory is one that is inductively derived from the study of the phenomenon it represents. That is, it is discovered, developed, and provisionally verified through systematic data collection and analysis of data pertaining to the phenomenon" (Strauss and Corbin, 1990:23) while "The grounded theory approach is a qualitative research method that uses a systematic set of procedures to develop an inductively derived grounded theory about a phenomenon". (ibid:24). However, although grounded theory is usually referred to as a qualitative method, even by its originators, the process of generating a theory is *independent* of the kind of data used. Theory can be generated from qualitative data, quantitative data or a mixture of both. (Glaser and Strauss, 1967:18).

Grounded Theory is often likened to ethnography, interpretative research, field research, naturalistic inquiry, observation, participant-observer method and case study method. Yet, while grounded theory shares some of the same philosophical traditions, it differs in one key respect -its emphasis on theory generation rather than data gathering and analysis.[6]

A comparison with case studies is instructive. Whereas a case study will typically seek to give an accurate description of a phenomenon through presentation of relevant data, a grounded theory study will seek to build theory (Strauss and Corbin, 1990: 21). The distinction, without theory is harder to make. Some case study researchers aim for

[6] Observation, participant observation and field research are research *techniques* common to a range of methods - both quantitative and qualitative- rather than approaches.

theoretical interpretations and some grounded theory studies do not produce a theory. A theory is based on related concepts. A case study is based on themes which may be conceptualisations but are more likely to be summaries of the data with little, if any, interpretation. So it very much comes down to the quality of the atheoretical study. Even if no theory has emerged, we should have greater interpretation of data. Also case studies are more likely to concentrate on a small number of cases, whereas grounded theory will typically examine multiple groups.

An inability to generalise from the specific is a criticism made of case studies and grounded theory. Walsham (1995) gives a number of examples of generalisations made from case studies where the generalisations have been tendencies rather than predictions. Orlikowski (1993) defends grounded theory on two fronts. She quotes Eisenhardt (1989) to argue that the constant comparative analysis of grounded theory is less likely to result in researcher prejudice than a theory built from 'armchair' induction. She notes the difference between statistical generalisation that generalises from a sample to a population and analytic generation which is the generalisation of theoretical concepts and patterns. One example is Zuboff's (1988) case studies of IT use in US organisations to develop the 'infomate' concept. Orlikowski extends her generalisation further "by combining the inductive concepts generated by her field study with insights from existing formal theory, in this case the innovation literature" (Orlikowski, 1993: 310).

Glaser and Strauss (1967) claim a number of specific benefits for the method

- grounded theory allows the development of an account of a phenomenon whilst simultaneously grounding that account in empirical observations or data. This

inductive method is particularly valuable where no theory exists or none is adequate.

- grounded theory provides a method for dealing with the complexity of social life and making it meaningful.
- the grounded theory approach is inherently processural - it deals with change and the consequence of change.
- grounded theory studies can have profound impact on the area studied. For example, Glaser and Strauss's (1965) study contained radical insights into improving medical education, the treatment of dying patients and their families.

However there are a number of acknowledged problems areas

- producing a grounded theory can be time consuming. Glaser and Strauss's (1965) seminal work, *Awareness of Dying,* was the result of nearly four years of intensive fieldwork in six urban hospitals. Orlikowski's (1993) study of the experience of two organisations with the use of CASE tools involved one hundred and fifty nine interviews each lasting an average of seventy five minutes, plus documentation, review and observation. (albeit, some data relating to one organisation had already been gathered as part of another research project).
- producing a grounded theory is difficult involving simultaneous collection and analysis of data, sampling on the basis of emerging theory and the generation and integration of abstract concepts.
- grounded theory is challenging to readers because it is dense and its boundaries are difficult to establish. However, one measure of a successful theory is that it should be readily understandable by the layman.

3.4.2 Overview

The following overview of the research method provides a background context to the detailed account of its specific application given in the next Chapter. It is informed by a glossary of technical terms included as Appendix 1. The key stages of the grounded theory method are shown in Figure 3

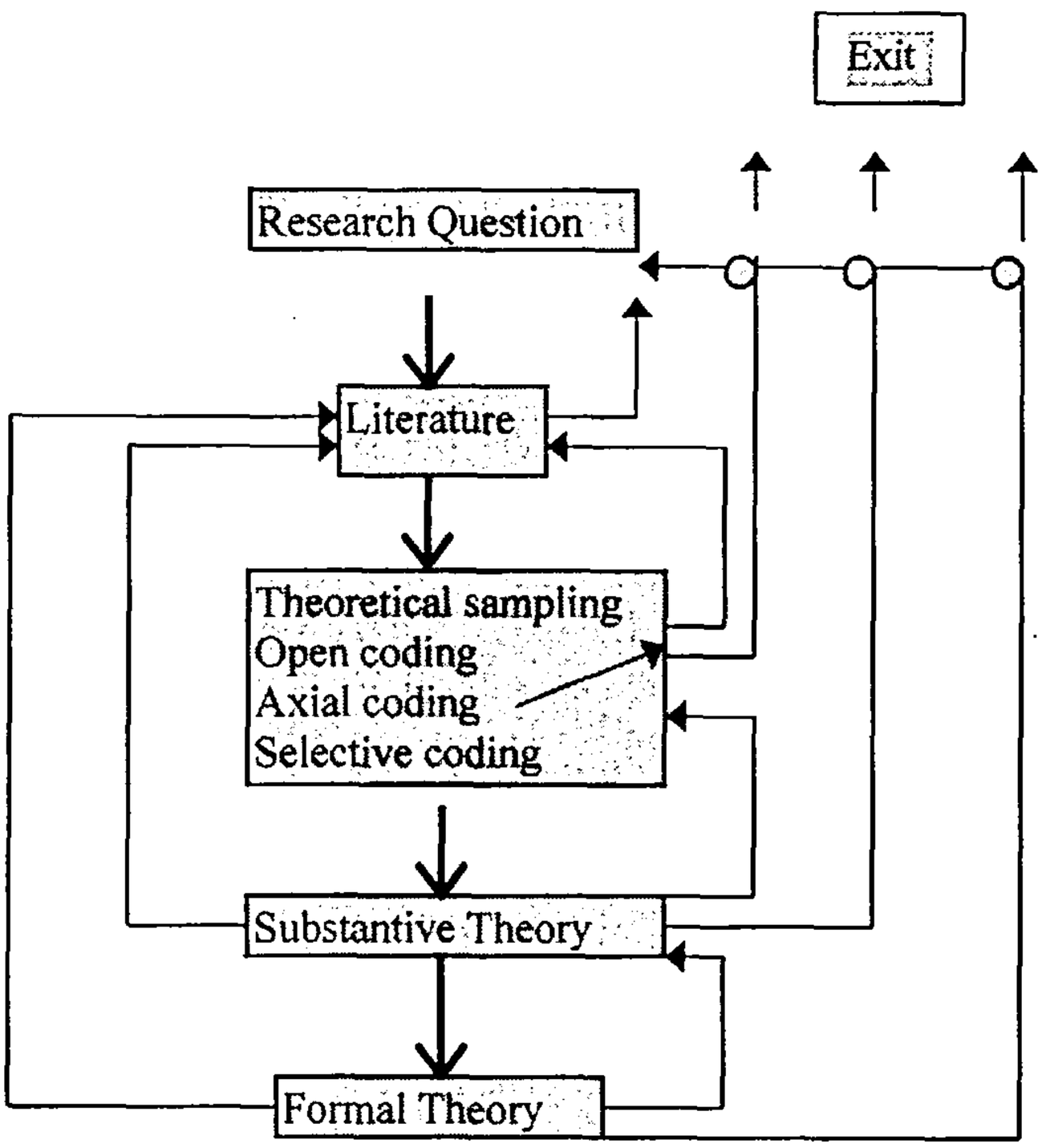


Figure 3: Key stages of the Grounded Theory approach

3.4.2.1 The research question(s)

The process begins with a research question(s) or statement that identifies the phenomena to be studied[7] The question may originate in personal or professional experience, it may be suggested by the literature, or it may be recommended by another researcher. Initially the question should be broad and flexible (within limits)

[7] In theory, the question should precede the method; we should select the method that best answers the research question. In practice, we are often biased, selecting a research question that fits our preferred method.

but become narrower and more focused as the research progresses. The question provides direction but should be reviewed in light of findings at each stage. For example, Hunt and Ropo's grounded theory study of leadership in General Motors began with the question "How do top-level philosophies, actions, and events impact and interact with selected aspects deeper within the organisation ? " (Hunt and Ropo, 1995: 389). They subsequently enumerated a number of propositions and integrated their findings with mainstream multi-level leadership theory. Orlikowski's study of CASE implementation in two organisations began with the question "what are the critical elements that shape the organisational changes associated with the adoption and use of CASE tools ?' (Orlikowski, 1993: 310) and ended with the integration of her results with formal theory taken from the literature on innovation. In this thesis, research questions first posed in Chapter One are investigated in Chapters Five, Six and Seven and "grounded" with existing theories of design and problem solving in Chapter Eight.

3.4.2.2 The use of Literature

Literature refers to primary and secondary data. Glaser and Strauss caution against a comprehensive literature review at the beginning "to carefully cover all the literature before commencing research increases the probability of brutally destroying one's potentialities as a theorist" (Glaser and Strauss, 1967: 251). However, Glaser and Strauss are not anti literature. They view the library as a valuable ethnographic source. Nor do they discount, as some believe (Walsham, 1995), the value of a firm foundation in literature *before* field work. The literature should be used at all stages of the research. Categories should be allowed to emerge and then checked out against the literature. Equally, literature should be grounded in data (emergent categories and

their relationships must be checked against primary data). The literature can be used as secondary source of data, to stimulate questions, to direct theoretical sampling, as supplementary validation, or to stimulate theoretical sensitivity.

Radical inductivists believe that if one avoids the literature, one will produce better work. Mellon (1990) for example, proposes that naturalistic inquiry - which attempts to focus on a phenomenon from the perspective of its participants- should begin with little or no awareness of existing literature. This, it is claimed, allows researchers to observe with no pre-conceived ideas or biases. But this view is not commonly held (Sandstrom and Sandstrom (1995) found little support for Mellon's position among naturalistic inquirers and ethnographers) and lacks common sense. Why should the researcher risk repeating past mistakes or re-discovering what is already known? Surely, there can be no justification for approaching field work unprepared.

Glaser and Strauss have been labelled inductivists and criticised for being anti literature. This, I believe, comes from a selective reading of their method and a consequent mis-representation of their position. Whilst they are concerned that potential theoretical sensitivity is lost when the sociologist commits himself "exclusively to one specific preconceived theory" (1967:46), they are not advocates of radical inductivism

> "no sociologist can possibly erase from his mind all the theory he knows before he begins his research. Indeed the trick is to line up what one takes as theoretically possible or probable with what one is finding in the field. Such existing sources of insight are to be cultivated, though not at the expense of insights generated by the qualitative research, which are still closer to the data. A combination of both are definitely desirable" (Glaser and Strauss, 1967: 253).

Their alleged anti literature position has to be put in context of a concern for

flexibility and theoretical sensitivity

> "some men seem to handle the precarious balance between the two source (existing and emerging theory) by avoiding the reading of much that relates to the relevant area until after they return from the field, they do this so as not to interfere with personal insights. On the other hand, some read extensively beforehand. Others periodically return to one or the other source for stimulation. There is no ready formula, one can only experiment to find which style of work gives the best results. *Not to experiment towards this end but carefully to cover all the literature before commencing research, increases the probability of brutally destroying one's potentialities as a theorist*" (Glaser and Strauss, 1967:253) [my emphasis]

They then go on to devote an entire chapter to the use of literature in grounded theory generation. This is based on an analogy between library research and field work. The library is "full of many voices waiting to be heard" and the desk researcher like the field worker needs intelligence, ingenuity and serendipity to make the most of his sources. He must know where to position himself (which resources to locate) and whom to talk to (the library is a wealth of recorded opinions, comments and conversations over time). He must not become overly reliant on one source (such as a cache of conference proceedings) but seek out comparative data.

Far from ruling out the use of literature then, Glaser and Strauss (1967:163) encouraged it, whether in the early days in order to better understand the substantive area being studied and to help formulate hypotheses, or as descriptive analysis (in the tradition of history or political science) or as highly empirical studies (such as content analysis or hermeneutics). For them "library materials are as potentially valuable for generating theory as are observations and interviews."

Glaser and Strauss are non prescriptive on the balance between field and desk research, between primary and secondary data. They do note however that while a substantive theory may be built entirely from desk research a formal theory will require the integration of the fieldwork of other researchers. Nor are they specific on

what type of literature should be selected or how. Inevitably some degree of arbitrariness is unavoidable, especially at the beginning of a research project.

In this research the general literature on software design and multimedia development was purposively avoided at the outset. This is reflected in the structure of the thesis. Some literature is called upon in Chapters One and Two to provide an introduction and background but this is limited. In fact, most of Chapter Two is based not upon the literature but upon an early analysis of the interview transcripts. Although relevant literature is used to corroborate empirical findings reported in Chapters Five and Six, the thesis does not present a literature review, per se. In Chapter Eight, under "related work", the outcomes of the research are "grounded" in relevant literature on the theory of design and general problem solving. In this the approach is consistent with that advocated by Glasser and Strauss (1967) and with other grounded theory studies within the Information Systems field (for example, Orlikowski, 1993). Moreover the use of a source of technical literature to generate Data-set B is supported.

3.4.2.3 Theoretical Sampling

Strauss and Corbin (1990:176) define theoretical sampling "as sampling on the basis of concepts that have a proven theoretical relevance to the existing theory" where the term " theoretical relevance indicates that certain concepts are deemed significant because (1) they are repeatedly present or noticeably absent when comparing incident after incident and (2) through the coding procedures they earn the status of categories"[8]

[8] The technical terms concept and category and others are explained in the Glossary, Appendix 1.

Sampling in grounded theory is unlike statistical sampling, one does not seek to identify and study a representative sample of a larger population and one cannot say in advance what the sample size and composition will be. Rather, one selects groups for study based on their purpose and relevance to the emerging theory. Relevance and purpose should ensure that the substantive area will be addressed and that the research objectives are met. Thus Orlikowski (1993), in her study of CASE implementation, selected two organisations that were both similar (in the substantive area of their selection and use of CASE tools) and different (in size, structure and culture). The latter to improve the applicability of the emerging theory.

In this thesis theoretical sampling was used within and across data sets. It was used within each dataset to select and order the data analysis (Chapter Four, 4.3.2) and across data sets where purpose and relevance to the emergent theory determined both data collection and analysis of the second data set. (Chapter Four, 4.2.3 and 4.3.6).

3.4.2.4 Coding

Data collection / coding / analysis are the critical stages through which a theory is built. The process is iterative with emerging concepts directing further data collection and new data often leading to a refinement of existing concepts. We cannot say in advance how much data will be gathered, how many groups sampled. The selection of groups and the collection of data is driven by the analysis itself and finishes only when we reach theoretical saturation, that is when enough data has been collected to explain the phenomena or when no new data can be found which adds to the concepts or categories (again, as is observed in the next chapter, practical constraints apply).

Open coding is a form of content analysis; it breaks down, examines, compares, conceptualises and categorises data. Axial coding puts the data back together again in new ways by making connections between categories using the paradigm model. This can happen immediately and periods of open coding are interspersed with periods of axial coding. Much of this happens automatically, with a constant interplay between deductive and inductive thinking (emerging concepts should be quickly verified in the data). Following axial coding, Strauss and Corbin (1990) suggest finishing the study if theme analysis or concept development is all that is required. However if one wishes to generate a substantive theory further analysis (through what they call selective coding) is necessary. In practice these different forms of coding are not discrete, sequential steps but rather proceed in parallel and the coder will be scarcely aware of the subtle transitions between one form and another. The coding procedure used in this research is detailed in the next chapter.

3.4.2.5 Theory

The grounded theory endgame is a substantial or formal theory. A substantive theory is specific to one context whereas a formal theory applies to many different situations. For example, a study of status in one organisation may lead to a substantive theory, the application of this substantive theory to different organisations - studying status in society for example - may lead to a formal theory. Elevating a substantive theory to formal theory increases generalisability but requires a great deal more work. One alternative is to integrate an emergent grounded theory covering a substantive area with an existing formal theory (see, Orlikowski; 1993). Of course, this should never be a bolted on, retrospective, attempt at validating an otherwise unconvincing

grounded theory. As made clear in Chapter One, the objectives of this research do not extend to developing a formal theory.

The grounded theory approach is inherently processural as well as comparative. Together with its inductive and contextual nature, these are its great strengths. Grounded theory facilitates "the generation of theories of process, sequence, and change pertaining to organisations, positions, and social interaction" (Glaser and Strauss, 1967: 14). This is in marked contrast to static, mainstream theory.

3.5 The Paradigm Model

In grounded theory the paradigm model has special significance. It is the means by which the data analysis is given depth and specificity. Strauss and Corbin (1990:99) are unequivocal in their advocacy of it "Unless you make use of this model, your grounded theory analysis will lack density and precision".

This section introduces the paradigm model by explaining its purpose and deployment. Each of its elements is then explained and the means by which these elements are related is discussed. Then the development of the model beyond the axial coding stage is discussed when a data analysis known as the Conditional Matrix is introduced. This section introduces the process detailed in Chapter Four and behind Chapters Five and Six and implicitly referred to therein.

3.5.1 Purpose and description

The paradigm model is used to link categories and sub-categories in a set of relationships. These relationships describe the phenomenon under study in terms of a set of conditions (causal, contextual, and intervening) and in terms of

action/interaction strategies and their consequences. A simplified form of this model would look like this

> **CAUSAL CONDITIONS [A] → PHENOMENON [B] → CONTEXT [C] → ACTION / INTERACTION STRATEGIES [D] → CONSEQUENCES [E]; INTERVENING CONDITIONS [F]**

Thus, causal conditions (A) lead to a phenomenon (B) which leads to context (C) which leads to action / interaction strategies (D) which then lead to consequences (E), under intervening conditions (F). This reflects something of the inductive nature of the paradigm model. Under conditions (A, C, F), which determine phenomenon (B), then strategies (D) are taken. This is quite different from a deductive model in which strategies (D) are determined by their relationship to the phenomenon (B) (and this relationship acts as a rule), while conditions (A, C, F) bound the already well defined phenomenon (B).

The purpose of the paradigm model is to enable the researcher to think systematically about the data and to relate them in complex (non-obvious) ways. The ability to do this is essential to the axial coding process within which the researcher is seeking to re-constitute the data in new and interesting ways following its decomposition during the open coding stage. Having identified categories and sub-categories during the open coding phase, the paradigm model is a mechanism to join these together. In theory this can be done for all categories, in practice this is usually impossible due to the number of categories involved. Each of its elements will now be explained.

3.5.1.1 Phenomenon

> *'the central idea or event or happening that a set of actions / interactions is directed at managing or handling, or to which the set of actions is related'.* (Strauss and Corbin, 1990:96)

Phenomena are identified during open coding, initially as concepts and then, perhaps, as categories. In fact it is the naming and categorising of phenomena through a process of asking questions and making comparisons ("the constant comparative method of analysis", Glaser and Strauss, 1967:101-116) during the open coding stage that first gives the method its precision and specificity. The paradigm model is really an extension of that process.

Many phenomena are identified during the open coding process and the researcher must decide which are the most important. They will then seek to group other phenomena or concepts around these (categorizing) which significantly reduces the number of units to be worked with. These categories will be developed in terms of their properties (characteristics or attributes of the category) and dimensions (locations of a property along a continuum). This is essential to the identification of relationships between categories and between categories and their sub-categories.

Eventually the researcher reduces the number of phenomena to a few major categories. Then one core category or phenomenon must chosen.

> "Sometimes two phenomena in the data strike the investigator as being equally important or of interest. It is essential however, to make a choice between them in order to achieve the tight integration and the dense development of categories required of a grounded theory. To fully develop two core categories, then to integrate the two, and to write about them with clarity and precision is very difficult. This is so even for the experienced writer and researcher" (Strauss and Corbin, 1990:121)

They offer some advice on how to choose the core category. The core category need not itself be a process but it should be capable of incorporating process for the grounded theory method is action oriented and processural. (The concept of awareness used in Glasser and Strauss' (1965) seminal study of the relationship between dying patients and hospital staff is given as an example). The core category should fit and describe the phenomenon and must be broad enough to encompass and

relate as subsidiary categories, the other categories. "The core category must be the sun, standing in orderly systematic relationships to its planets" (Strauss and Corbin, 1990:124).

The core category in this analysis – **context-complexity – action-interaction** is first identified in Chapter Five and further developed in Chapters Six and Seven. The rationale behind this process is set out in Chapter Four (4.3.4)

3.5.1.2 Conditions

Each and every phenomenon is subject to a number of conditions. These conditions are vital to our understanding of a phenomenon, they can explain its presence, account for the strategies taken to manage it and their consequences. The process of identifying conditions, which themselves are categories and phenomena, and relating them to the core phenomenon (and to each other) can be quite complex, in particular because conditions change over time.

Conditions can be defined by *type* and by *level*. Conceptually conditions can be represented at a number of different levels, some conditions will be specific or close to the phenomenon in question, others will be broad and distant from it. Levels of conditions will be discussed further as the paradigm model is developed using the Conditional Matrix (see section 3.6).

There are three types of condition defined by the impact they exert on the phenomenon - causal, contextual and intervening.

3.5.1.2.1 Causal Conditions

"Events, incidents, happenings that lead to the occurrence or development of a phenomenon"

Also known as antecedents, this set of conditions can be anything that causes a phenomenon. In reality a phenomenon is rarely produced by a single causal condition but by a combination of such conditions. The researcher must identify these conditions and develop them in terms of their properties and dimensions for it is this that gives precision and specificity to the phenomenon and which, in turn, influence the strategies taken to manage it. Strauss and Corbin give the example of a study of the phenomenon of pain. There are many possible causes of pain, one of which is having a broken leg. It is the properties and their dimensions of the broken leg that determine strategies taken to manage pain. For example the broken leg may have multiple fractures or a compound fracture.

Strauss and Corbin suggest two approaches to identifying causal conditions. The first is to locate action verbs within the data and the second is to focus on a phenomenon and systematically search back through the data for those events, incidents or happenings that seem to precede it. Both approaches were used in this study but many of the causal conditions included in the final model were located originally within another condition type but re-designated as the logic of the model developed. This is discussed further in Chapter Four (4.3.2)

3.5.1.2.2 Context

"Context represents a specific set of properties that pertain to a phenomenon. That is, properties or attributes relating to a phenomenon are located along a dimensional range. Context is therefore also the particular set of conditions within which the action / interaction strategies are taken to manage, or respond to, a specific phenomenon" (Strauss and Corbin, 1990:101).

This definition requires some clarification. A phenomenon is instantiated at any point in time by its properties and the location of these properties along a dimensional range. It is these properties and their dimensional value that gives the phenomenon specificity. For example, a phenomenon may be defined by its properties and their dimensions to be a, b, c. These properties and their dimensions define the context in which action/interaction strategies are taken to manage the phenomenon. So we can then say that under conditions of a, b, c, strategies 1, 2, 3 are employed or under conditions of d, e, f, strategies 4, 5, 6 are employed.

What determines the context? Causal conditions lead to a phenomenon but it is the properties and the dimensional values of the properties of the causal conditions that determine the phenomenon's context. There is a direct relationship between those conditions that cause the phenomenon and those conditions that define it. Returning to the example of pain caused by a broken leg, one could say that under conditions of multiple fractures and a compound break the pain is intense and of long duration. In this study it was observed that ill defined, uncertain and volatile user requirements caused a complex design context which influenced design strategies, and in turn was influenced by these. These relationships are developed in Chapters Five and Six.

But it is not just causal conditions that determine context, intervening conditions, action/interaction strategies and consequences also help shape it.

3.5.1.2.3 Intervening Conditions

> *"These are broad and general conditions bearing upon action / interaction strategies. They may be regarded as a broader structural context pertaining to the phenomenon. Intervening conditions act to either facilitate or constrain the action / interaction strategies taken within a specific context"* (Strauss and Corbin, 1990:103)

Intervening conditions stand between context and action/interaction strategies. As such they too must be managed. Intervening conditions range from those distant to the situation, to those close in and include such conditions as time, space, culture, economic status, technological status, career, history, and individual biography. In the example of a broken leg leading to pain, relevant intervening conditions would be where the leg was broken (geographical location), age and health of the victim, his or her attitude towards pain, knowledge of first aid etc. These conditions, and others, influence the strategies taken to manage the pain. In the case of software design intervening conditions include (inter alia) the personality of the individual designer, his or her education and experience, the methods used by the organisation and the organisational culture.

The level and number of intervening conditions included in a study is determined by the phenomenon itself and by the researcher's skill in recognising what is relevant, but is limited by practical considerations. To be relevant a condition must be verified by data as having a direct or indirect effect upon the phenomenon. The researcher should try to ensure that all relevant conditions are included so that the phenomenon is precise but he or she should not include so many conditions that the phenomenon becomes vague and specificity is lost. Intervening conditions make relating categories to each other more difficult because they get in the way of simple cause leads to consequence logic. They explain variations, for example why one person chooses a certain strategy and another doesn't or why one person is successful and another isn't.

As such they are an essential element of the paradigm model.

3.5.1.3 Action/Interaction

> "Strategies devised to manage, handle, carry out, respond to a phenomenon under a specific set of perceived conditions" (Strauss and Corbin, 1990:97)

Strategies are usually, but not always, purposeful or goal oriented - they are carried out for a specific reason - to manage or respond to a phenomenon. Sometimes a strategy will be reflexive or taken for reasons unrelated to the phenomenon under study but have consequences for it. Strategies are also processual , they evolve or change over time. Also, failed action / interaction or a failure to engage in action / interaction can be just as important (descriptive) as success.

Strauss and Corbin comment that "Of all the paradigm features, action and /or interaction lie at the heart of grounded theory" (1990:159) Action and / or interaction is the essential activity on which all other conditions and consequences are based. First there is action "the active, expressive, performance form of self and / or other interaction carried out to manage, respond to and so forth, a phenomenon" (1990:164). Action is carried out through action processes such as performing an operation, conducting an experiment, writing a thesis, drawing an E-R diagram, writing a requirements specification, programming.

Interaction on the other hand means "people doing things together or with respect to one another in regards to a phenomenon (Becker, 1986) and the action, talk and thought processes that accompany the doing of those things" (Strauss and Corbin:1990:164). Interaction is taken to include "even things done alone" which require interaction in the form of self -reflection. The example given is managing an illness that requires interaction with others to obtain medical supplies etc. Examples of interaction processes include negotiation, domination, teaching, and debate. In the

field of software design interaction includes communication and collaboration, and related use of prototypes and storyboards.

Strauss (1987) gives the example of the division of labour to illustrate the differences between action and interaction but also their inter-dependence. The division of labour refers at one level to the action process for the carrying out of the phenomenon of work, different people doing different tasks to some end. On another level it involves negotiations, discussions and legitimisation of boundaries and so forth that take place in order to arrive at and maintain a division of labour and accomplish its associated tasks. The two levels of strategy are equally necessary, but the interaction level is often overlooked.

A practical difficulty for the researcher in the application of Strauss and Corbin's approach to grounded theory is that their definitions of "action" and "interaction" are not clear, despite, or perhaps because of, the examples they give (the division of labour, p164; the Head nurse, p. 169). What exactly is "action"? – the "active expressive, performance form of self"?. When and how does action become interaction? Since self reflection is held to be one form of interaction it is clear that most non trivial tasks will contain some element of interaction. This is certainly the case in design where the opportunities for, and the significance of, self reflection are well documented (Schon, 1983). Moreover even if a task could be identified as action (only) it is always possible that through process or change action will become interaction through self reflection. Therefore in this thesis the expedient of taking action and interaction together as a single process was adopted, whilst also recognising that (a) any task or strategy may consist of action and interaction and (b) may exhibit more action than interaction or vice versa. This is significant to the

analysis in Chapter Six and subsequently to the development of a theoretical framework based on identifiable levels of interaction.

3.5.1.4 Consequences

"Outcomes or results of action and interaction" (Strauss and Corbin, 1990:97)

Action / Interaction strategies taken in response to, or to manage, a phenomenon have certain outcomes or consequences. These might not always be predictable or what was intended. The failure to take action / interaction also has consequences. Consequences may affect people, places or things, they may be events or happenings, or they may take the form of a responsive action / interaction (for example, if someone asks you to do something you don't like doing, your response may be to do it badly or not at all). Also, the consequences of one set of actions may become part of the conditions (causal, contextual or intervening) of the next set of actions / interactions or of a set of actions / interactions sometime in the future. Of interest in this study are consequences that impact on the quality of the design (process and outcome) and the process through which such consequences influence future design strategies and outcomes.

3.5.1.5 Summary

Thus far the paradigm model has been presented as a set of interrelated elements.

Action / Interaction is the key element for it is "the manner in which any phenomenon is expressed" (Strauss and Corbin, 1990:159). But action / interaction (and its related phenomena) is embedded in a set of conditions and it is these conditions (causal, contextual or intervening), that, to a greater or lesser extent, determine the expression of the phenomenon. Action / Interaction has certain consequences and these too may

become part of the conditions for a future set of actions / interactions. Table 5 summaries the relationships between the elements of the paradigm model.

	Causes	**Context**	**Strategies**	**Consequences**	**Intervening Conditions**
Causes		Primary determines through properties and dimensional values of these	Precede or lead to (strategies to manage) design	No direct influence.	No direct influence. Together shape Context. Share many common categories
Context	No direct influence. However in its broadest sense Context subsumes Causes		Primarily determines. Context explains why certain strategies are taken	Primarily determines the outcomes of a given strategy	No direct influence. However in its broadest sense Context subsumes ICs
Strategies	No direct influence but indirectly influence Causes via Consequences	No direct influence but indirectly influence Context via Consequences		Directly result in. However a strategy's outcome is moderated by Context	No direct influence but indirectly influence ICs via Consequences
Consequences	Directly influence. Become part of future inputs	Directly influence. Become part of future inputs	Directly influence. For example a successful strategy is more likely to be repeated		Directly influence. Become part of future inputs
Intervening Conditions	No direct influence. Together shape Context. Share many common categories	Directly influence. Of secondary importance to Causes in this respect	Facilitate or constrain strategies taken in a given Context	Indirect influence via context and strategies. However may be critical to outcomes. For example education and experience	

Table 5: The relationships between elements of the paradigm model.

The development of a paradigm model for any phenomenon is a concrete outcome of the axial coding process. Strauss and Corbin (1990:115) emphasise that this does not mean that the researcher can ignore other means of discovery but that he or she should "continue to look for additional properties of each category and to denote the dimensional location of each, incident, event or happening". They then go on to describe ways in which the analysis may be made more abstract (through selective coding) and to define a framework that summarises and integrates the analysis (the conditional matrix).

3.6 The Conditional Matrix

Strauss and Corbin propose the Conditional Matrix as a means of more systematically relating conditions, action/interaction and consequences to a phenomenon. The conditional matrix is a framework that "denotes a complex web of interrelated conditions, action/interaction, and consequences that pertains to a given phenomenon" (1990:161) and is predicated on their view of grounded theory as a transactional system – "a system of analysis that examines action / interaction in relationship to their conditions and consequences" (1990:158). This transactional system is made up of interactive and interrelated levels of conditions, which range from the broadest, or more general features of the world at large, to the more specific - those closest to the phenomenon under investigation. Central to the transactional system, and located within the range of conditions, is action / interaction[9].

---

[9] It should be noted that the organisation of a phenomenon into different layers is not unusual in qualitative research. Lofland (1971) for example, has provided a classification of social phenomena that can be used as a coding scheme (acts-activities-meanings-participation-relationships-settings).

They represent the conditional matrix as a set of circles, one inside the other, each (level) "corresponding to different aspects of the world around us" (1990:161). The outer rings include those conditional features most distant to the action / interaction, while the inner rings include those features most closely related to the action/interaction sequence. At the core of the circle stands action and interaction.

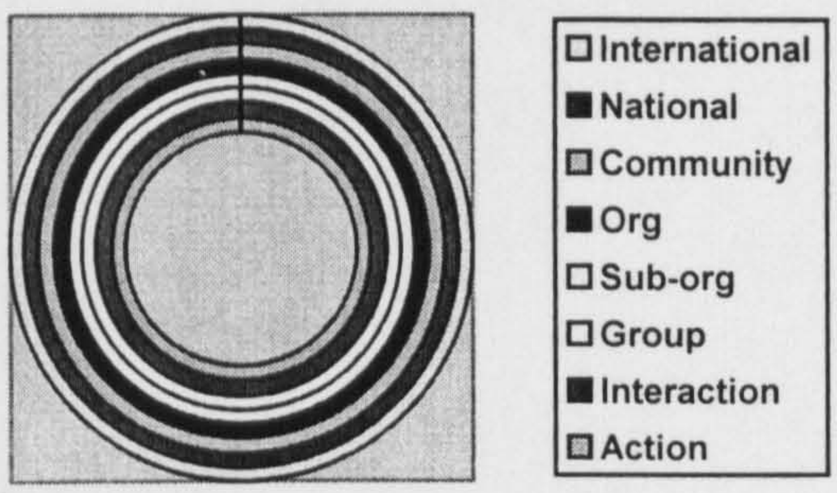


Figure 4: The Conditional Matrix (Strauss and Corbin, 1990)

Strauss and Corbin make three important points about this matrix

- conditions at *all* levels have relevance to *any* study. For example the values and attitudes that actors bring with them to a particular action or interaction are located at the outer level.
- the researcher can study *any* phenomenon at *any* level of the matrix. They give the example of world hunger which can be studied at the individual, group, community, national or international level.
- regardless of the level within which a phenomenon is located, it will stand in conditional relationship to levels above and below it, as well as within the level itself.

Each researcher needs to give specificity to each level of the matrix by relating each condition to the particular phenomenon under study. What is to be included will be influenced by the type and scope of the phenomenon under investigation. Ultimately the researcher must decide what is to be included and what emphasis each level should receive in the overall analysis of a given phenomenon. In turn, practical considerations associated with the pursuance of the research will influence the decision. In this study the conditional matrix was considerably simplified with the number of levels reduced from eight to four. The next chapter discusses the rationale for this and the outcomes of the analysis are presented in Chapter Six.

3.7 The method evaluated

> " Strauss and Corbin's (1990) ... attempt to present that original approach (Glaser and Strauss, 1967) in a straightforward, proceduralised form but without loosing any of its comprehensiveness and intellectual complexity. This uncompromised intent has resulted in a step by step 'method' which is difficult to follow in practice except in a loose, non-rigid, non-speciable fashion which inevitably draws it back toward the original version. This difficulty is borne out by published grounded theory studies in the field of organisation and management, which are characteristically vague about their interpretation of grounded theory methodology and method. It is supported by the author's experience with doctoral students who have abandoned the approach because of its bewildering complexity" Partington, (1998:8)

Partington argues that the paradigm model as advocated by Strauss and Corbin is overly complex for researchers "seeking to build grounded theories of managerial cognition using interview data". This stems from the models origin in "the constructivist philosophical assumptions of symbolic interactionism" which "exerted a powerful sway over the development of the grounded theory approach" (Partington, 1998:10). He goes on to propose a simplified model consisting of just three elements and a conditional matrix simplified from eight concentric rings to four.

By so doing he undermines his own criticism. The paradigm model and the conditional matrix, even when simplified cannot be used in isolation from the

particular approach to grounded theory analysis that informs them. In rejecting the specific approach on the one hand but embracing the tools of that approach on the other, Partington is being disingenuous. In this thesis and elsewhere (see for example Konechi ,1997) simplified versions of the paradigm model and the conditional matrix are used without rejecting the overall approach. Indeed, in this case, the "straightforward, proceduralised form" was welcomed and, it is argued, used precisely because it did not mean abandoning the essential richness of Glaser and Strauss's original method.

It is beyond the scope of this thesis to detail further criticisms of the Grounded Theory method, or specific interpretations of it. Much of literature degenerates into positivist versus anti-positivist rhetoric wherein the particular biases of individual researchers are represented rather than any objective evaluation of methods. The interested reader is directed to general criticisms of inductive methods (see for example, Bateson, 1973, and Archer, 1998).

3.8 Why was protocol analysis not used?

In the introduction to an edited collection of essays on protocol analysis, Cross, Christianns and Dorst (1996) observe

> "Of all the empirical observational research methods for the analysis of design activity, protocol analysis is the one that has received the most use and attention in recent years. It has become regarded as the most likely method (perhaps the only method) to bring out into the open the somewhat mysterious cognitive abilities of designers" (1996:1)

Hinrichs (1992) suggests why this may be so

> "the method favours the study of tasks that are neither too easy (routine) nor too hard (creative problems)..that class of design problem for which potential design components are known but design plans are not....best described by Newell and Simon's Problem Space Hypothesis" (1992:180)

Protocol analysis is based on (concurrent or retrospective) verbal accounts given by subjects of their own cognitive activities. It has its origins in psychological research methods of the 1920's and the first report of a study of design activity was Eastman's (1970) study of architects. However it was not until the late 1980's that protocol studies of engineering design began to appear. (Ullman et al, 1988; Adelson, 1989; Whitefield and Warren, 1989; Ennis and Gyeszly, 1991; Ehrlenspiel and Dylla, 1993, Lloyd and Scott, 1994). During this period the protocol analysis method was extended to include team design activity (Tang, 1991; Minneman and Leifer, 1993, Visser, 1993). Most engineering studies have been of mechanical engineering but electronic engineering has also been studied (Colgan and Spence, 1991). Protocol analysis has also been used to study software design (Jeffries et al, 1981; Guindon, 1990; Davies and Castell, 1992) including the design of computer supported co-operative work systems (Olson, 1992). Although protocol analysis has traditionally been applied within one domain, a few inter-disciplinary studies have also taken place (Thomas and Carroll, 1979, Goel and Pirolli, 1992).

Protocol analysis has been used to develop theory exclusively (Ericsson and Simon, 1984) and in tandem with the grounded theory approach (for example Ancona's, 1990 study of consulting teams). [10] Cross at al (1996) suggest that protocol analysis is best used with other techniques, for example using participant observation and interviews to examine longer term design processes and using protocol analysis to study, specific, short term activities.

---

[10] The literature on protocol studies is a specific exception to the general rule observed in chapter one that the Meta-process behind the development of models and theories is not well documented.

Why was protocol analysis not used in this study? Ignorance of the method cannot be offered as an excuse in this case. Although the author had no prior experience of conducting a protocol study, he had some knowledge of the purpose and procedures of the method. Anyway, prior to this research, the author had no experience in conducting a grounded theory study. Neither can the principal weakness in the method be used as an explanation. Protocol analysis is ill suited to collecting non-verbal data but this is hardly a reason to reject it in favour of an approach based almost entirely on the analysis of interview transcripts! Grounded theory is more suited to the study of longer term design processes but this was neither the principal intent nor the obvious outcome of this study. In truth, no convincing scientific or pseudo scientific reason can be formulated in response to the question. Indeed none is offered, nor need it be.

As indicated at the beginning of this research method was chosen because of (a) fitness for purpose - it lent itself well to answering the research questions and (b) philosophical fit - the epistemological and ontological basis of the method were consistent with the researcher's disposition toward qualitative approaches to IS research problems. At the beginning of the research there was no expectation that Simon and Newell's problem space hypothesis would be employed to contextualize the findings (Chapter Eight) and therefore no suggestion that protocol analysis would be a good way to explore such a theory in this instance. It is a moot point whether or not protocol analysis would have resulted in a better study since the outcome of any research is never certain. In one respect at least however, it would not. Grounded theory studies of design are much less common and each contribution to the literature, potentially at least, disproportionately increases our appreciation of the diversity and utility of design studies.

3.9 Chapter Conclusion

This chapter provided some epistemological and historical background to the chosen research method. This method – grounded theory - was introduced and its major stages described. Within this method a prescriptive approach to data analysis (Strauss and Corbin, 1990) was preferred and two tools used in the analysis – the paradigm model and the conditional matrix – were outlined. The reader is reminded however that the model of software design produced as an output of this analysis is descriptive rather than prescriptive, it is not proposed as an approach to *doing* software design but only as a one means to *study* it.

Chapter Four: The Research Design

> "You have to explain yourself well (be articulate) and if you are honest then this helps" (graphic designer)

4.1 Introduction

This chapter explains in detail the application of the research methodology introduced in the previous chapter. Following Bryman and Burgess (1994:3) the chapter seeks to articulate the research as it was done, as opposed to how it should have been done and this (autobiographical) account is as candid as time, space and memory permit.

Firstly the data sources are introduced and the procedure for data collection, including sampling strategy, outlined. Then the data analysis approach is described in detail illustrated where necessary with examples from the data. Consistent with one objective of the research, and with a claimed contribution identified in Chapter One, each phase of the analysis is explicitly referenced to the procedural steps recommended by Strauss and Corbin (1990). This is followed by a review of the ways in which the validity and reliability of the data have been, or can be, established. Finally the approach taken to data storage and management is outlined. The chapter concludes with a summary of those aspects of the research design that uniquely inform the morphology of the study.

4.2 Data collection

4.2.1 Scope, rationale and overview

In grounded theory the process of data collection is driven by emerging theory.

> "The basic criterion governing the selection of comparison groups[11] for discovering theory is their theoretical relevance for furthering the development of emerging categories" (Glaser and Strauss, 1967:49)

---

[11] a comparison group may be a social unit of any size, a single person may be designated a group. A group may pre-exist or may be created (according to emerging concepts)

Therefore, the researcher does not know at the outset which groups will be selected nor even how many. However Glaser and Strauss do offer some guidance

> "The sociologist may begin the research with a partial framework of local concepts, designating a few principal or gross features of the structure and processes in the situations that he will study ....the researcher chooses any groups that will help generate, to the fullest extent, as many properties of the categories as possible, and that will help relate categories to each other and to their properties" (1967:49)

Yet some control over data selection, beyond relevance to emerging theory, is necessary if the research is to be completed satisfactorily and on time. The researcher can exercise some control over the choice of groups *a priori* through deciding whether he or she wishes to build a substantive or formal theory. For example, a substantive theory on software design needs only a study of software designers, a formal theory of design needs a study of many different types of design professionals. So in aiming for a substantive theory rather than a formal theory, the task of data collection is made easier, or at least limited.

Further control over data collection can be exercised through the techniques of comparative analysis.

> "Control over similarities and differences is vital for discovering categories and for relating their properties" however "When beginning his generation of a substantive theory the sociologist establishes the basic categories and their properties by minimising differences in comparative groups" (Glaser and Strauss, 1967:55-56)

In fact, good substantive theory can result from the study of only one group, if the analyst carefully sorts the data into comparative subgroups. Only when the basic categories are established should the researcher seek to maximise differences between groups in order to stimulate the further generation of theoretical properties.

> "By maximising or minimising differences among participating groups the sociologist can control the theoretical relevance of his data" (Glaser and Strauss, 1967:55).

Glaser and Strauss are non-prescriptive on the data to be collected and the techniques to be used in analysing it "there are no limits to the techniques of data collection, the way they are used or the types of data acquired" (1967:64).

In theory, sampling should stop only when theoretical saturation is reached, that is when no additional data can be found to further develop the properties of a category. In practice, there is a need to limit data collection. Glaser and Strauss themselves acknowledge this when they say

> "in comparative studies of more than two groups, the sociologist usually tries to compare as many of the groups for which he can obtain data *within the limits of his own time and money and his degree of access to those groups*" (1967:48) (my emphasis)

The researcher can reduce the amount of data to be collected as the research develops. Initially they may need to collect data on the entire group but when the main categories have emerged he needs to collect data only on the categories that are relevant to the emerging theory. Also it is possible to collect data on many categories simultaneously.

In summary, the main points that guided data collection in this study were

1. The exercise was appropriate to the stated objectives of the research.
2. The exercise was carried out under constraints of time and access to data.
3. The exercise was driven by theoretical sensitivity but tempered by practical constraints (referred to in 1 and 2).

With this in mind, the major steps of the data collection process will now be discussed.

4.2.2 Data-set A

These interviews were conducted and transcribed during a separate but cognate research project investigating Multimedia systems development methods (Gallagher and Webb,1997; Gallagher and Webb, 2000). The purpose of these interviews was to identify disciplinary paradigms within the field but the focus was design. Consequently a large part of each interview was given over to the subject of design

covering (inter-alia) definitions, motivations, influences, processes, techniques, tools and methods. In all twenty three interviews were completed. Collectively these interviews constituted a valuable record of what local software designers operating in the field of Digital Interactive Multimedia (DIMM) said about design.

It is readily acknowledged that the secondary analysis of interview transcripts, whether obtained in the literature or gathered by proxy, is devoid of the richness of "being there". However in this instance three factors minimised or offset the loss of technique

1. The researcher was already familiar with the subject area, and in particular with professional practice as the result of previous research studies (Webb and Booth, 1995; Webb, 1996). This provided a background context that, itself, grounded the data analysis.
2. Detailed background and contextual information for each interview was provided. For example, by including annotations relating to humour, pauses and hesitation, gestures and other cues.
3. Following Strauss and Corbin (1990:31) "regardless of whether you transcribe all or part (or none) of the transcripts, it is important to listen to the tapes" the original recordings were readily available during the analysis and proved essential to resolving issues unclear from the transcripts.

Support for the approach taken is claimed from the literature. As noted in the previous chapter (3.4.2.2) Glaser and Strauss (1967) do not rule out the generation of theory based solely on secondary data. Strauss and Corbin (1997) include in their collection of grounded theory "exemplars" a number of studies that relied heavily on secondary

data. Elsewhere, Turner provides a description of how he developed a grounded theory by the coding of data drawn entirely from official public records into a fire at Summerland Leisure Centre, Douglas, Isle of Man, in 1973 (in Bryman and Burgess; 1994). Strauss and Corbin (1990:189) refer explicitly to such analysis. They employ Glasser's (1967) definition of secondary analysis whereby the researcher employs sampling and coding to the "collected interviews or field notes of another researcher".

Secondary analysis is well documented in the field of hermeneutics. Originating in the study of ancient texts, hermeneutics offers a powerful analogy for primary research whereby "reading of text provides a model for reading human behaviour" (Lee, 1997). Yet it is also the case that, even in its more limited, original application, hermeneutics can uncover much about people and organisations, documented in interview transcripts or other sources, recorded first hand or not. Specifically, through a careful reading of the text we may establish a context for action, whilst also drawing upon other sources, including our own experiences.

Finally one practical benefit of *not* doing the interviews is noted. Since there is less danger that the coder reads too much into each and every statement, coding is easier, quicker and more objective.

4.2.2.1 Sampling strategy

Interviewees were selected on the basis of purposive / judgement sampling and snowball sampling. Purposive / Judgmental sampling (according to Babbie, 1995) is when a sample is selected based on the researcher's own knowledge of the population, and the nature of the research. Snowball sampling is when a small sample is selected (by whatever means) and then expands based on referrals from the initial sample. So,

an initial list of interviewees was generated via local knowledge and contacts and then enlarged based on personal recommendations. The purposive / judgement sampling method and the snowballing sampling method necessarily involve some degree of theoretical sensitivity, consistent with the grounded theory approach.

4.2.2.2 Format and procedure

The interviews were semi-structured, and an interview format consisting of a series of open-ended questions was used. The advantages of this approach - in particular, the greater clarity and control which it offered - were deemed to outweigh the disadvantages (the danger that it may lead respondents towards answers that they feel are 'correct'). The approach also permitted a copy of the interview schedule to be sent to participants prior to the interview. A number of benefits accrue from this. It permits participants to consider in advance how they may respond and reduces the anxiety experienced by respondents. The combined effect of which is likely to increase the quality of responses. It also facilitates short, or shorter, interviews. These benefits were observed during the interviews. Many respondents had prepared written notes in advance of the interview with the result that their responses were much more free-flowing. Also the willingness of the majority of participants to allow the interview to be recorded was, at least in part, attributable to the fact that they had had the interview schedule beforehand.

Two other benefits of semi-structured interviews became clear during the data analysis stage. Firstly the format allows triangulation of the data requested through multiple questions on the same, or broadly similar, topic. For example designers were asked "to define design" and "to describe what you do when you design". Secondly, semi-structured interviews are much easier and quicker to code than unstructured

interviews as the coder does not have to search through the entire transcript when looking for themes and concepts. This is particularly important when multiple readings of a transcript are required.

Each interview lasted from between 60 to 120 minutes. During the interview participants were encouraged to be as expansive as possible but to base their responses to specific questions on one or two recent projects. This helped "anchor" the discussion on the participant's recent experience and therefore, helped improve the clarity and relevance of the response. Although formal visualisation techniques were not used an A4 pad was made available and a number of participants drew diagrams or scribbled notes to clarify particular points.

The interview transcript was sent to the participant to allow him or her to review the material for accuracy. This is recommended practice in such approaches (Walsham, 1995). Clear instructions were given not to arbitrarily amend content but to raise any concerns with the researcher. Minor amendments, including annotations relating to style and emphasis, were permitted however. In the event, few (less than fifteen percent of interviews used in the data analysis) were annotated in this manner, and these involved only minor changes. Whilst it cannot be assumed that those that failed to return transcripts or otherwise comment upon them had no significant amendments to content, neither can it be assumed that they did. That is, a failure to return a transcript does not automatically invalidate it. Where a transcript was read, it may be that the participant was happy with the record and felt no further response was necessary. In fact such a scenario was confirmed by two software engineers, verbally (and independently) to the researcher, some time after the interviews.

Walsham (1995:78) advocates making interviews the "primary data source" since they provide the researcher with the "best access to the interpretations that participants have regarding the actions and events which have or are taking place". Agar (1980:110) suggests that the researcher should treat interviews "as the core of ethnographic fieldwork", making them the central source of data to be corroborated by other data. An attempt was made to triangulate the data in this case by requesting company documentation but this met with a limited response. However during the interviews, the interviewer was frequently shown designs, sketches and storyboards. Where appropriate such information was included with the transcript, either as annotations on the main text or as supplementary material.

Partington (1998:10-11) has described a grounded theory study based solely on interview data. He raises the essential problem with this research technique - data is not based on observed events but on the informants second hand account of those events and that "reality is a stage further away from the immediate reality of the interviewees words and two stages further away from that which is observable by the interviewer". He purports to deal with this problem by "developing an improved grounded theory framework" (in essence a simplified conditional matrix) and by "aligning this more centrally with causal aims" (anchoring his research in a theory of reality described by Bhaskar's (1975) critical realist ontology).

The limitations of the interviews as a data collection technique are well documented in the literature (see for example, Weick, 1967; Judd, Smith, and Kidder, 1991). Often interviewees are asked to explain something that is not readily explained by language, or as “doers” rather than “thinkers”, have little time to reflect on practice. How can one be sure that the data are both comprehensive and representative? Where the unit

of analysis is an explicit word or statement how can one be sure that something significant was not missed, something not stated explicitly but assumed implicitly?

Of course one cannot ever be sure. Rather it falls to the researcher to convince him or her self, and others, that the problems are known and addressed. In this case, it is acknowledged that practitioners may not have said something because they believed it to be so evident that it didn't need to be said and that this something may be significant to a description of software design. Some reassurance is offered through triangulation of data (section 4.4.1.1) in that other evidence pertaining to a phenomenon was considered, but this is limited. Further assurance may be derived from some work described in Chapter Seven. This shows that there is little variance between the essential elements of the paradigm model presented in this thesis and Wernick's (1995) and Gallagher's (1999) Kuhnian analysis of software engineering and graphic design textbooks.

All (twenty three) of the original transcriptions were read to establish the quality and scope of the available data. An equal number of transcripts from each of the two disciplines (eight graphic designers and eight software engineers) was then selected for more detailed analysis. The selection was made on the basis of the simple quality criteria of the breadth and depth of the material relevant to the objectives of this research study and directed by the stated research questions. No attempt was made to ensure the sample was representative, either of the larger sample or of the population from which it was drawn.

4.2.2.3 The Interviewees

One significant epistemological starting point for this research was that, following Winograd et al (1996), a software designer is defined as one who designs software whether that be interface design, database design, algorithm design or whatever. All the interviewees selected for analysis in this study are, or have been recently, active, at various levels, in one or more aspect of software design (as here defined) in the field of digital interactive multimedia (Data-set A) or more generally (Data-set B).

Craig (1991), referring to the field of interface design, indicates the essential difficulty of exclusive definitions.

> "Because interface design is complex, it is difficult to find a single person, with the abilities and skills that are necessary for a designer. Together, the graphic designer and the software engineer function as a designer. Their two perspectives balance the design of the user interface. The work of these professionals blends into a finished product that will satisfy the user's needs. The graphic designer does not create the interface in its entirety and neither does the software engineer, but both work together as craftsperson, artist, and designer" (Craig, 1991: 139-140)

More recently, Rijken (1999:46) includes graphic design and software development in a definition of media design, along with journalism and interaction design but in contrast to architecture (which includes urban planning and landscape design).

Of course, it is not axiomatic that because a graphic designer is an equal partner in the design of an interface that he or she is necessarily a *software* designer. If one defines software design as designing (planning or developing) algorithms and code then a graphic designer does not, ordinarily, design software (although some do and it is a moot point whether the design and development of icons for example can be excluded from any definition of software design). However if software design is interpreted in the much broader sense to mean the total design of the software artefact, as Winograd et al suggest, then graphic designers engaged in the design and development of user interfaces are software designers. Thus in Winograd's book, in the interviews used in

this research and in interviews reported elsewhere (Lammers 1989), individuals who have little or no involvement in the writing of actual code call themselves software designers. It is the broader definition of software design that underpins and informs the remainder of this thesis[12].

Interviewee	Experience*	Education
GD1	12 years	BA Graphic Design
GD2	24 years	BA Graphic Design
GD3	20 years	BA Graphic Design
GD4	11 years	BA Graphic Design
GD5	08 years	BA Graphic Design
GD6	03 years	BA Visual Com.
GD7	04 years	BSc Comp & Design
GD8	01 year	BA Visual Com.
SE1	18 years	HND Computing
SE2	10 years	BSc Eng. MSc Comput.
SE3	14 years	BSc Eng. MSc Comput.
SE4	13 years	BSc Computer Science
SE5	06 years	BSc Computer Science
SE6	10 years	BSc Computer Science
SE7	01 year	BSc Computing & IS
SE8	03 years	BSc Computer Science

Table 6: Data-set A – Interviewee Profile

* experience refers to total years experience in the referent field of graphic design or software engineering. As the number of years experience in the sub-field of digital interactive multimedia was not always clear, this is not shown. In general those with the least experience overall tend to be those with experience only in the DIMM field.

It should be noted that although each interviewee is classified as either a software engineer or a graphic designer these labels are indicative of the interviewee's formal education and the type of work that they do, or have recently done. The sample included a wide range of education and experiences and not all those classified as a software engineer or a graphic designer would necessarily describe themselves as such. For example some graphic designers prefer the title of "Multimedia designer" or "information architect", some software engineers would prefer to describe themselves

[12] In the context of the argument set out in Chapter One, software engineering and graphic design are two sub-disciplines of software design, studied here (Data-set A) in the field of Digital Interactive Multimedia, and more generally (Data-set B).

as a "programmer" or a "system analyst". Moreover a range of skill and experience levels is represented in both disciplines and some interviewees have progressed to managerial positions. Nevertheless, each interviewee was assigned to one or other of the two major referent disciplines in the field (what Gallagher (1998) has identified as paradigms or communities). The purpose of this assignment was to maximise the generative power of the grounded theory coding procedures based on the constant comparative analysis of data. The way in which this was done will be explained later in the chapter.

4.2.3 Data-set B

Lammers (1989) set out to record the "experiences, approaches, and philosophies of software designers in a personal, in-depth manner", to "look into the minds and personalities behind the software" using interviews in which the "individuals speak for themselves" (1989:1). She sought a cross section of "specialities and experiences" but primarily focused on the programming. Significantly she states

> "for the purposes of this book, the word 'programmer' is defined as a developer of software or a designer of software, often but not always involved in the actual writing of code" (1989:3)

The interviews comprised a set of open ended questions designed to "highlight the similarities and differences in approaches to programming" and to "allow the personality and special interests of each programmer to come to the fore". Interviews were transcribed, edited and refined and returned to the interviewee "so that they could read what had been said and re-work the interview, so that it expressed exactly what they meant". In the end, nineteen interviews "from a very large pool of talented programmers" was published (Lammers, 1989: 2-3).

All nineteen interviews were read. Of these, sixteen were selected and coded. Sampling was based on purpose and relevance to emerging theory, and guided by theoretical sensitivity. Following the analysis of Data-set A complexity was identified as a powerful explanatory category (Chapter Six). Further interviews were sought that would develop this category, wherein descriptions of larger and more complex projects could be found. Therefore relevance to emerging theory guided the selection of the source and of interviews from within that source. In fact the process was more iterative and messy than this account implies. Insights gained during the analysis of Data-set B often occasioned a return to Data-set A, for example to develop concepts and categories. Strauss and Corbin (1990:181) note that such iteration is a feature of increased theoretical sensitivity.

The interviews selected were; Charles Simonyi; Butler Lampson; John Warnock; Gary Kildall; Bill Gates; John Page; C. Wayne Ratcliff; Bob Frankston; Jonathan Sachs; Ray Ozzie; Peter Roizen; Bob Carr; Jeff Rankin; Toru Iwatani; Scott Kim; Jaron Lanier.

Of these, all but four could be considered software engineers. Twelve were either formally educated in Computer Science and / or had worked in the field of software engineering. Rankin, Iwatani, Kim, and Lanier have a variety of (sometimes quite bizarre) backgrounds including, inter alia, musician, games designer, graphic designer. For the purposes of this study they are grouped as graphic designers.

The purpose of analysing this data was to develop emerging theory by extrapolating from the sub discipline of DIMM to the super-ordinate discipline of software design, consistent with the objectives of the research and making the outcomes of the analysis

of Data-set A more relevant and more generalisable. In addition the analysis of the Lammer's data set permitted one measure of validity and reliability. Many concepts and categories identified as a result of the analysis of transcripts of Data-set A were also identified in Data-set B (section 4.3.6 describes this process, its outcomes are reported in Chapter Seven)

4.3 Data analysis

4.3.1 Rationale and process outline

Boyatzis (1998) identifies three ways to develop a thematic code (a) theory driven (b) prior data or research driven (c) inductive or data driven. The three approaches form a continuum from theory driven to data-driven approaches. The continuum can be said to reflect increasing uncertainty and ambiguity of the analysis of code, increasing time to develop the code and increasing discomfort for the researcher. Thus whilst the theory-driven approach is comfortable for many researchers,

> "when entering the path of the data driven approach, researchers must have a great deal of faith that they will arrive at a desirable destination, especially because they do not know where it will be, what it will look like once they are there, and how long it will take" (1998:29)

The approach to coding used in this study is a hybrid of Boyatzis's (1998) data-driven (or inductive) approach and the research or prior data driven approach. Early coding of the interview transcripts generated data driven codes (concepts and categories). It soon became clear however that a purely inductive approach would be too protracted. Axial coding afforded a framework to organise and make sense of emerging data in the form of the paradigm model. This was readily adopted. The researcher was re-assured of the legitimacy of this model by its application to related phenomena (for example Partington (1997) and Konechi (1994) used the model to describe different aspects of management) and by its application to categories generated during early

coding. At the same time, this researcher was aware of the importance attached to context in design (for example by Schon, 1983; Seely Brown and Duguid, 1994; Lawson, 1997; Beyer and Holtzbalt, 1998) and felt confident that the model could be used to make a contribution to knowledge in this area. Figure 5 provides and overview of the data analysis process used in this study.

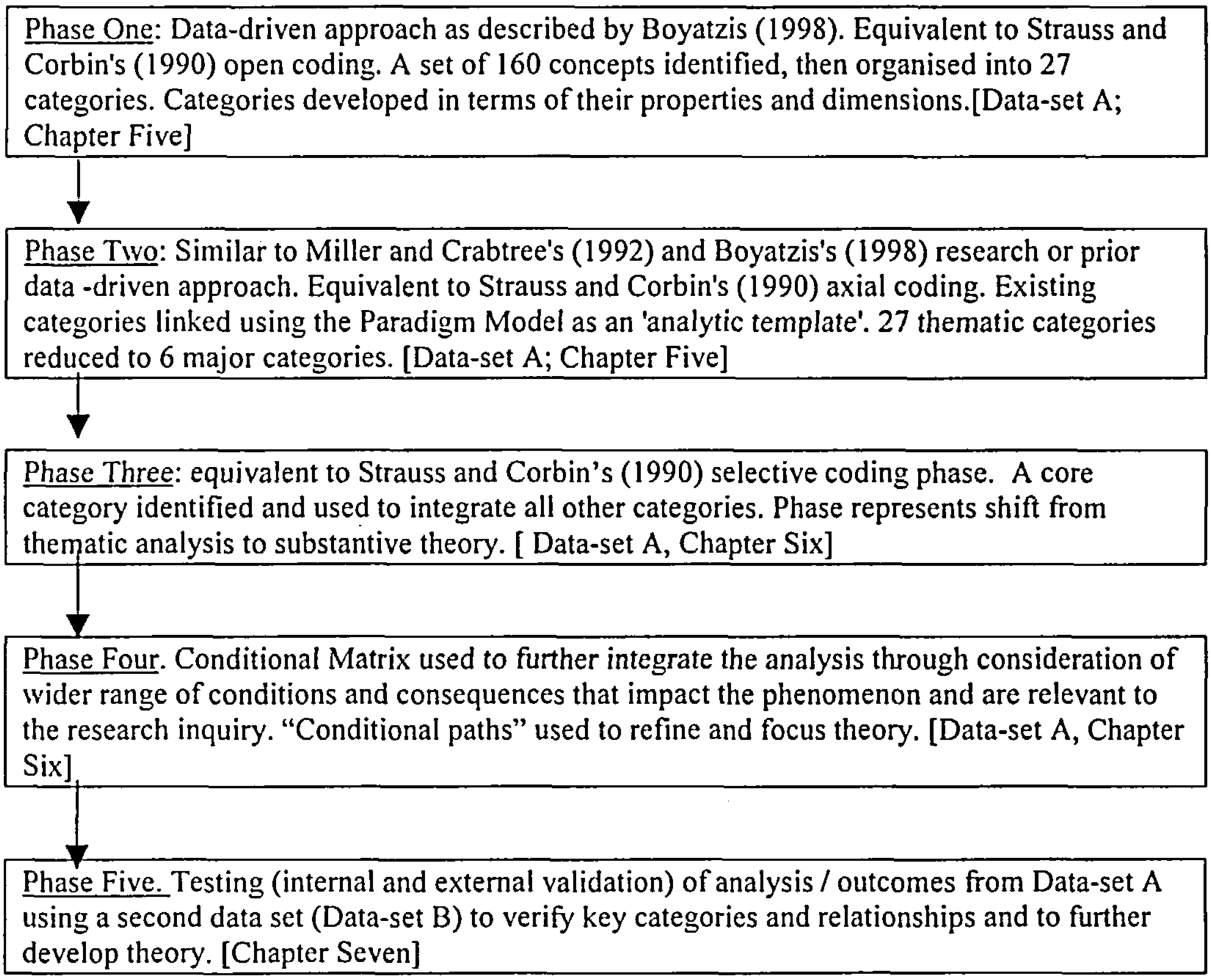


Figure 5: Overview of the data analysis process

In Phase one, concepts were identified and grouped into categories and categories were developed in terms of their properties and dimensions. Phase Two involved using the frameworks of the Paradigm model to relate and group the categories at a higher level of abstraction. During Phase Two gaps in the analysis often required a return to the original transcripts. When this happened, the transcript was read from the perspective of the paradigm model. So for example, when it became clear that both

causes and consequences of software design were poorly represented in the model, all the transcripts were re-read to identify instances of causes and consequences in the data. Here the analysis was driven by the paradigm model and not by the data. As such the approach used is similar to what Miller and Crabtree (1992) describe as a "template analytic technique" in which the researcher uses a pre- existing code or framework to process and or/ analyse the data, and to Boyatzis's (1998) prior data or research driven approach.

In practice the process was more complex than that conveyed by this simplified outline. Bechhofer's (1974:73) comment that qualitative research " is not a clear cut sequence of procedures following a neat pattern, but a messy interaction between the conceptual world and the empirical world, deduction and induction occurring at the same time" and Ritchie and Spencer's (1994:218) description of the coding process as "invariably associated with the cutting and pasting of transcripts or notes; whereby chunks of text are cut out and pasted with other items that fit under a certain heading" give a better flavour of what actually went on. With this in mind the data analysis procedure will now be explained in detail.

4.3.2 Phase One data analysis (open coding)

This phase is broadly equivalent to Strauss and Corbin's (1990) open coding and Boyatzis's (1998) process of thematic analysis and code development. Boyatzis (1998:4) defines a code as "a list of themes; a complex model with themes, indicators, and qualifications that are causally related, or something in between these two forms". Where a theme is "a pattern found in the information that at minimum describes and organises the possible observations and at maximum interprets aspects of the phenomenon". According to this definition the paradigm model itself is a form of

coding (wherein causal relationships are established between 'themes'). However for the purposes of this discussion a code is considered equivalent to a *concept, a category (or a sub-category)* and *the properties* and *dimensions* thereof[13]. Table 7 summarises the steps involved in this phase of the analysis

Inputs	Process	Outputs
16 interview transcripts Data-set A	Sample Data-set based on simple quality criteria	4 inter disciplinary sub samples. Sequence of coding determined
4 sub samples	Code transcripts to generate concepts. Organise concepts into categories. Develop categories in terms of their properties and dimensions	Set of concepts Set of thematic categories
Thematic categories	Amend thematic categories to accommodate differences in the data (where necessary)	Final set of thematic categories tolerant of significant differences in the data

Table 7: Steps taken during Phase One data analysis

4.3.2.1 Phase One - Step 1: Create sub-samples

All the transcripts were read in their entirety and the quality of each graded according to loosely pre-determined criteria including clarity, consistency, relevance and perceived contribution. Based on this approach, transcripts were organised into four sub samples. The purpose of this was to maximise the opportunity for generating useful codes as early in the analysis as possible[14]. Thus the sub-sample with the highest perceived quality was coded first and so on until all four sub-samples were coded. As the analysis progressed the number of new codes being identified diminished. Figure 6 shows a Pareto type distribution of codes by transcript.

---

[13] these terms are further defined in Annex 1 (Glossary)

[14] The benefits of this approach were deemed to outweigh any attendant disadvantages, such as the loss of theoretical sensitivity in not coding each transcript in random order. Moreover since it was considered impossible to completely set aside previous codes when coding a new transcript (despite what the textbooks urge) far better to be prejudiced by good code than bad. The procedure adopted is akin to that described by Strauss and Corbin (1990:187) as discriminate sampling – the choice of data that will maximise the analysis. Discriminate sampling is normally associated with the selective coding stage and sampling during the open coding stage is normally indiscriminate.

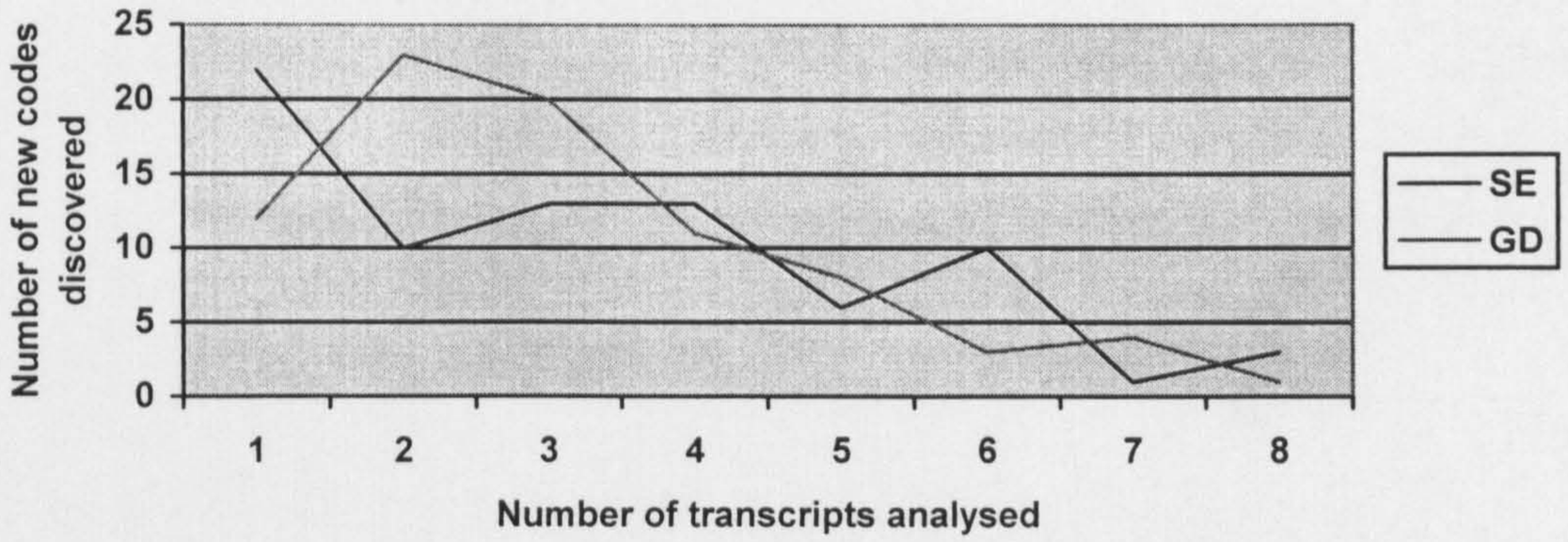


Figure 6: Data-set A – Distribution of new codes by transcript

Each sub-sample consisted of two software engineering transcripts and two graphic design transcripts. Within each sub sample, one transcript was read and coded then one from the other discipline. Alternation was designed to maximise "theoretical sensitivity" to the data (Glaser and Strauss, 1967) [15]. Boyatzis (1998) refers to this process as "criterion referencing" or "anchoring" the data. This sequence was repeated until each sub sample (and the entire sample) was coded.

SE1>GD1>SE2>GD2 {A}

SE3>GD3>SE4>GD4 {B}

SE5>GD5>SE6>GD6 {C}

SE7>GD7>SE8>GD8 {D}

The sampling of the data based on design disciplines is not inconsistent with the view that software engineers and graphic designers are software designers. Rather,

[15] An early attempt to analyse the data in disciplinary blocks of four was abandoned as being insufficiently sensitive to differences *between* disciplines. No significance should be attached to the fact that a software engineering transcript was the first to be coded in each sub sample and overall.

following Schon (1983) a common core of design is anticipated but with differences arising out of different technical environments.

4.3.2.2 Phase One - Step 2: Generate code

Each selected transcript was read and re-read and concepts (words, phrases or paragraphs) that seemed important (i.e. descriptive of the phenomenon, relevant to the purpose of the study) were extracted. An entry was created in the database by cutting and pasting the original text or by keying a summary of what was said. Each entry was given a label or name, its exact location in the transcript was identified and three keywords recorded. (The mechanism used for storing and retrieving codes is described in the section on data management, 4.5 and the codes are listed in Appendix 3). In addition a memo was often written further describing the concept (what Strauss and Corbin call an "operational memo"), or developing it analytically and conceptually (a "theoretical memo") and this was referenced to the concept (database) record. Appendix 4 gives an example of such coding. Thus whilst the concept records themselves were descriptive each fostered analytical memos within which the original concept was further developed. Moreover the three keywords per concept provided a quick and easy way to compare concepts and an initial lead to categorisation.

When a transcript had been coded for concepts an attempt was made to organise these concepts into categories. Categories are higher level and more abstract entities that subsume two or more concepts. This was rarely successful at the individual transcript level but became more significant at higher levels of granularity. Thus after a pair of transcripts had been coded for concepts and certainly after all four transcripts in a sub sample had been coded, a number of categories were generated which subsumed and explained many (but not necessarily all) of the concepts generated. As noted above,

most concepts (and therefore categories) were generated across sub-samples A and B. In practice, the process was quasi-deductive in that concepts and categories identified from the better quality transcripts were sought in the remaining transcripts. New concepts (and categories) were still being generated but at a much reduced rate. As a general rule of thumb where a concept was simply repeated (within the same source or elsewhere) and did not add anything new to what was already coded, it was ignored. Otherwise it was included but with only the value added described.

During this stage supported categories were further developed in terms of their properties (and the values of the dimensions of each property)[16]. This was an important step. Because "properties and dimensions form the basis for making relationships between categories" and " to understand the nature of properties and their relationships is a requisite task for understanding, in turn, all of the analytical procedures for developing a grounded theory" (Strauss and Corbin, 1990:70). In practice the step was easier (and quicker) than first anticipated as the major categories expanded to subsume weaker categories as sub-categories or properties. This proved effective in reducing the total number of categories to be considered, albeit simultaneously adding to the complexity of each remaining category, and to the relationships between categories.

4.3.2.3 Phase One - Step 3: validate code

During this stage categories were developed to take account of differences. This often involved going back and re-reading the transcripts in each sub-sample and amending the categories to take account of any exceptions or exclusions in the data. Where two

categories were found to describe essentially the same phenomenon then the category that best differentiated the data was retained and the other was rejected. Thus, when the analysis produced two categories - **balancing** and **coping** – that described a similar strategy, only balancing was retained and further developed since it was an in vivo code that described more accurately the observed phenomenon.

Where a category did not tolerate broad variation in the data, and could not be amended to do so, it was rejected. For example, although a process of **seduction** (by technology, and to a lesser extent by complexity) was identified in many interview transcripts, the category **pulls** was used instead as this better incorporated a similar response to aesthetics. Significantly data that was found to be inconsistent or contradictory did not automatically invalidate a category. Rather this was looked upon as an opportunity to develop the category, adding richness to it through the addition of new properties or a greater dimensional range for existing properties.

One example of this is the development of the category **structure.** Based on an early analysis of software engineering transcripts this referred only to data structures but was quickly amended to accommodate references in graphic design transcripts to information content. In other studies vagueness in the data is welcomed and included as a feature of the phenomenon under investigation (eg. Galal, 1996) and is not inconsistent with the grounded theory approach as set out by Strauss and Corbin (1990). Appendix 5A lists the final set of thematic categories

---

[16] An analogy can be made here with defining the domain range of possible values of an attribute in database design.

4.3.3 Phase Two data analysis (axial coding using the paradigm model)

The first phase of data analysis resulted in a set of (27) thematic categories that each described one (or more) aspect (s) of the phenomenon of software design. This phase is equivalent to Strauss and Corbin's (1990:96-115) axial coding stage, the purpose of which is to develop several main categories of the phenomenon in some detail. In a later phase, these main categories will be related under a core category to further integrate the analysis and formulate a substantive theory (see 4.3.4 and Chapter Six).

In this phase of the analysis, the paradigm model was used at three distinct but inter-related levels

- At the *macro* level to order all 27 thematic categories in relation to the phenomenon. Each category was further classified as a condition, strategy or consequence. This facilitated an immediate overview of the types of categories involved and the relationships between them, and prompted further questions. What is the main story here? What are the key categories and relationships in explaining this story? Do these categories (and their relationships) describe uniquely and specifically the phenomenon under study?
- At the *meso* level to examine the relationships between categories, within classification. Thus all the categories already classified as conditions, strategies or consequences were further examined. Within each classification, a single integrative category was identified and all the other categories within that classification related to it, as sub-categories – again using the paradigm model[17].

This level reduced the total number of thematic categories to be considered (from

[17] Effectively, a core category was sought that integrated all other categories within that classification. At this level, not all elements of the paradigm model were represented– nor needed they to be. The value of the model was really in getting the researcher to think about the data in a structured way.

27 to 6) but added to the complexity of each remaining category, and to the relationships between categories. These categories are set out in Appendix 5B.

- At the *micro* level to examine each integrative category in terms of its properties (sub-categories) and to examine the relationships between integrative categories based on the dimensional values of these. This final level of analysis was intended to add specificity and density to the analysis through the examination of variation in the data.

Chapter Five discusses the outputs from each of these levels, and their aggregation.

Sometimes a category could legitimately belong to more than one aspect of the paradigm model. For example, we can say that **user requirements** *cause or lead* to a specific context. But we can also say that user requirements are an *intervening condition* - that is they *come between* action/interaction and consequences. Should this category be identified as a cause, an intervening condition or both? On such occasions the transcripts were re-read to contextualise the category. For example, causes would often contain action verbs implying antecedence to a phenomenon, or could be identified by tracing back from the phenomenon. Sometimes it proved very difficult to follow a causal relationship in the data.

Tracing a path from an action / interaction to its consequences was difficult given intervening conditions, elapsed time and process. When the function of a particular category was not obvious theoretical sensitivity to the data became crucial and the researcher assigned the category according to 'best fit' to the emerging analysis. Thus for example, the category of **user requirements** became a causal condition rather than (the equally legitimate) intervening condition because the category was

considered to have greater power in explaining the phenomenon when considered as a causal condition. This did not mean that the category was not considered as an intervening condition only that its function as a causal factor was given precedence in the paradigm model.

Relating categories by means of their properties using the paradigm model as an analytical template was a deductive – inductive cycle, as recommended by Strauss and Corbin (1990:107). A relationship was proposed on the basis of some data. Further evidence of this relationship was sought elsewhere in the data. In particular negative or invalidating data was looked for. If the relationship was corroborated (or at least not refuted) it became a propositional rule (to be further tested elsewhere). Otherwise the relationship was revised, thus adding richness and variation to the analysis, or rejected outright.

The development of the category **context-complexity** illustrates this process. Early in the analysis it became clear that, in many situations, the interaction of conditions (causal, contextual and intervening) created a complex design environment (characterised by uncertainty and volatility). However the analysis of two transcripts of interviews with software engineers showed that this was not always the case, other design scenarios existed that were non-complex, or at least less complex, due to the interaction of a different set of dimensional values attached to the same properties of the category. This context was found in engineering or "back end" design projects and provided a significant and timely qualification to the emerging general category.

Strauss and Corbin (1990:107) caution that the process of axial coding is quite complex because the analysis is performing four distinct analytical steps simultaneously. These are

> "(a) the hypothetical relating of subcategories to a category by means of statements denoting the nature of the relationships between them and the phenomenon (b) the verification of those hypotheses against actual data; (c) the continued search for the properties of categories and subcategories, and the dimensional locations of data indicative of them; (d) the beginning exploration of variance in phenomena, by comparing each category and its subcategories for different patterns discovered by comparing dimensional locations of instances of data."

In this study much potential complexity was avoided by limiting the number of categories under (a) and thus verification steps (b), further search (c) and the exploration of variation (d).

Finally it is noted that, at this level, the integrative categories themselves had to be grounded in the data. Those that worked best – those that were most successful in integrating all other categories within a classification – were found in the transcripts. Attempts to apply integrative categories formulated purely on logic or on deduction based on weak empirical evidence were unsuccessful. Thus, for example, an attempt to range action and interaction strategies along a spectrum ranging from visual to cognitive was abandoned because it was both unhelpful and unsustainable[18]

4.3.4 Phase Three data analysis: (selective coding)

This analysis builds immediately and directly upon that of the axial coding stage. In fact, selective coding is really a more advanced form of axial coding wherein the data is analysed at a higher, more abstract, level. The approach to this phase of the analysis

---

[18] Winograd and Flores (1987) contend that cognition is not a separate identifiable act and to see it as such is to belong to the rational tradition.

is based on the selective coding procedure set out by Strauss and Corbin (1990:118 – 142) but is discussed here under three broad headings.

4.3.4.1: Phase Three: Step 1 - Identify and develop the core category

> "To achieve integration, it is necessary first to formulate and commit yourself to a story line. This is the conceptualisation of a descriptive story about the central phenomenon of the study" (Strauss and Corbin, 1990:119).

In practice this comes down to writing down, as clearly and as succinctly as possible, the essence of the research findings – as they appear at this stage of the analysis. Then, this "general descriptive overview" is developed conceptually and analytically through the identification of a core category and relating all other categories to it – again using the paradigm model. The core category identified in this study was in fact an amalgam of two thematic categories based on a common property. Thus **context complexity (level of complexity) - action/interaction (level of interaction)** became the core category. The outcomes of Step One of this phase of the analysis, and the logic behind its implementation, is set out in Chapter Six (6.2)

4.3.4.2 Phase Three: Step 2 - Identify patterns of relationships in the data

The core category is defined and developed by means of its properties. It is the dimensional values of each of the properties of the core category that give it specificity. Moreover it is the specific value of each property of each category that makes possible the linking of categories (around the core category) using the paradigm model. Based on such linkages it is possible to identify patterns ("repeating relationships between properties and dimensions of categories") in the data, and account for variation. "It is very important to identify these patterns and to group the data accordingly, because this is what gives the theory specificity" (Strauss and Corbin, 1990:130). Patterns in the data may present themselves during the analysis or

may be deduced (through a basic process of asking questions), in which case they must be verified in the data.

Earlier it was argued that context is primarily determined by the properties and dimensional values of causal conditions. Thus user needs that are vague, uncertain and volatile contribute to a complex design context. Strauss and Corbin also point out that context is an arrangement "of properties of the general phenomenon, ordered in various combinations, along their dimensional range to form patterns" (1990:131). In this case, deduction produced four design scenarios or patterns derived from the intersection of the properties and dimensional values of two major categories – context complexity and action/interaction (Ch 6.3). Each was then validated or grounded in the data.

Once identified these patterns or design scenarios can (and should) be used to group the other major categories. This entails examining the properties and dimensional values of each category related to each of the identified patterns or design scenarios (Chapter Six, 6.4). The objective of this is that "the data are now related not only at the broad conceptual level, but also at the property and dimensional levels for each major category .. [the] rudiments of a theory!" (Strauss and Corbin:1990:133)

4.3.4.3 Phase Three: Step 3 – identify and develop the theory

The final step of this phase of data analysis involves "laying out the theory", making statements of the relationships (propositions) and verifying these in the data. Also any gaps in the data should be addressed. For example if one context is poorly represented in the data. Under step 4.3.4.2 above one design scenario [low context complexity-high interaction context] was weakly supported by the data and no amount of further

analysis changed that situation. The scenario was included (rather than rejected) because the logic of the deduction suggested that such a scenario could exist (at least in theory). However in the discussion of the inductive model and in the layout of the final theory this weakness had to be acknowledged and its implications explored (Chapter Six).

On the other hand poor support in the data for proposed or emerging categories lead to their rejection. For example, during the analysis it became clear that **negotiation** was an important process in design but the lack of direct evidence - on when, where, and how it took place, on the types and levels of negotiation, and on the key players and their interactions – excluded this category from the final list. Instead the data relating to negotiation was absorbed into the categories of **communication** and **collaboration**, and their interaction. Similarly, although both a logical and empirical outcome of communication and collaboration, **compromise** was not included as a separate category but absorbed under the category of **balancing**.

Although presented here in sequence, in reality there was considerable iteration between steps, as Straus and Corbin readily acknowledge. The outcomes of each of these steps and of this phase of coding overall is presented in Chapter Six

4.3.5 Phase Four: using the Conditional Matrix to further integrate the analysis

Strauss and Corbin (1990) provide an additional framework to further develop the analysis. The Conditional Matrix (CM) summarises and integrates categories through a number of "interactive and interrelated levels of conditions". Strauss and Corbin are non-prescriptive on the number of levels to be considered, they outline seven, but state that each researcher must give specifity to his or her own analysis. In this study,

four levels of conditions are included in the CM, action, interaction, individual conditions and organisational conditions. These levels are grounded in the data and they encompass those factors that most describe individual design activity. Although discussion did occur on broader issues, such as the competitive environment, the future state and direction of the field, these were not included as a separate layer in the CM but considered only as they impacted upon lower and more specific layers.

The conditional matrix is used to

(a) integrate the analysis outlined in phases one to three and presented in Chapter Five

(b) explore further the conditions and consequences that impact upon the phenomenon under study (based on the view of software design as a transactional system

(c) Explicitly expose the role and impact of process or change on the data

(d) Summarise and present the analysis in a final explanatory framework or theory

The Conditional Matrix is operationalised through the tracing of a conditional path. A conditional path is "the tracking of an event, incident, or happening from action/ interaction through various conditional and consequential levels, and vice versa, in order to directly link them to a phenomenon" (1990:158). It links concepts systematically, giving specificity to the analysis (wherein conditions are identified as causal, contextual or intervening), and scopes the investigation since it is only possible to trace the most relevant and interesting paths. In this study, conditional paths were considered for each of the four design scenarios. However only one of these was traced in some detail. The outcome of this analysis is presented in Chapter Six (6.4 and 6.5).

4.3.6 Phase Five: tests for internal and external validity using Data-set B

Finally in the process of data analysis, a second data set, Data-set B, was used to validate the analysis of Data-set A. Strauss and Corbin (1990:187) note that such discriminate sampling "maximises opportunities for verifying the storyline, relationships between categories, and for filling in poorly developed categories". Therefore whilst testing is built into each step of the primary analysis –comparing hypotheses against data, making modifications and testing again – a separate set of tests where undertaken. Strauss and Corbin (1990:190) caution that

> "remember, a theory is just that – a theory. To find out through further testing that a propositions does not hold up, does not necessarily indicate that the theory is wrong; but that its propositions have to be altered or expanded or encompass additional and specifically different conditions"

These tests are further introduced in the next section and presented in Chapter Seven, along with tests applied to Data-set A.

4.4 Further tests for validity and reliability

> "Qualitative researchers have no single stance or consensus on addressing traditional topics such as validity and reliability in qualitative studies. Early qualitative researchers felt compelled to relate traditional notions of validity and reliability to the procedures in quantitative research (eg. Goetz and Le Compte, 1984). Later qualitative writers developed their own language to distance themselves from positivist paradigms. Lincoln and Guba (1985) and, more recently, Erlandsen, Harris, Skipper, and Allen (1993) discuss establishing quality criteria such as "trustworthiness" and "authenticity". These are all viable stances on the question of validity and reliability" (Creswell, 1994: 157-158)

Creswell's own approach to this question is allied to that of Merriman (1988) and Miles and Huberman (1984). He distinguishes between internal validity, external validity and reliability. Internal validity is the accuracy of the information and whether it matches reality and involves such procedures as triangulation of data, getting feedback from informants and including participants in all phases of the research. External validity relates to the generalizability of findings, which in qualitative studies is limited. Reliability is concerned with the replication of the study in another context. This too is limited in qualitative research but the researcher can

help by making available as much relevant information as possible. For example, statements about research positions, central assumptions, selection of informants, biases and values of the researcher all improve the chances of a study being replicated in another setting.

Rather ironically, Creswell's initial point about the lack of consensus among qualitative researchers is underlined by his own text. He includes his discussion of validity and reliability under the heading of "verification steps". Boyatzis (1998:144), however, states that reliability "is not verification, which is a pure, positivistic notion". Boyatzis goes on to comment on the causal relationship between validity and reliability " [the] validity of findings cannot conceptually exceed the reliability of the judgements made in coding or processing the raw information" (1998:144). Black (1993:72) on the other hand notes that "while it is possible to have an instrument that is valid but not reliable, an instrument that is not valid will never be reliable". Later Boyatzis appears to acknowledge a syllogism in the use of the two terms when discussing the issue of confidence, which he identifies with reliability "the word can also be applied to a researcher's belief or trust that he or she has captured the phenomenon under investigation and that his or her judgements are sound" (1998:170). Black points out that "the discussion of validity in the literature is littered with controversy" (1993:67) but that a valid research instrument should measure what it is intended to measure. He also distinguishes between internal validity ("the design of the study and the collection of variables"(1993:171)) and external validity ("generalizability"(:172)).

This next section of this chapter is concerned with internal validity (the accuracy of the information and whether it matches reality (Creswell, 1994) and reliability

(consistency of judgements over time and space (Boyatziz)). Discussion of external validity (generalisability (Creswell, 1994; Black, 1993)) is deferred until the Chapter Seven of the thesis, and is thus informed by the findings of the analysis of Data-set B.

4.4.1 Internal validity

Some steps taken to ensure internal validity have already been discussed in the sections on data collection and analysis but will be summarised again here for the purposes of transparency and completeness.

4.4.1.1 Triangulation of source data

During the data collection phase, before, during and after each interview, every opportunity was taken to acquire source documentation that could be used to validate the main research instrument. This met with limited success. Aside from the perennial issue of confidentiality, many of the companies visited simply did not have well documented systems. Most documentation that was gathered was generated as a result of, and during, the interview. Thus the interviewer would often return from an interview session armed not only with the crucial audio-tape but with sketches and drawings produced during the conversation. In one case a process description developed for the same company during a previous research exercise (Webb and Booth, 1995) was used to clarify some points of the narrative.

Triangulation within or between data of the same source type was more valuable. In this way this researcher attempted to deal with perceived inconsistencies within single or multiple interview transcripts. For example, one software engineer in the early part of the interview claimed that coding was not design but later talked at length about coding in the context of design. Faced with this discrepancy (and in the absence of an

opportunity to clarify this directly with the interviewee) the researcher attached greater weight to the latter interpretation due its prominence (length and emphasis) in that transcript.

4.4.1.2 Getting feedback from informants

This was primarily achieved by sending the transcript to the participant for comment but as discussed earlier in this chapter (Section 2) this procedure met with limited success. In addition, some limited feedback was received informally and verbally.

4.4.1.3 Involving participants in all stages of the research

This was not possible in this study due to obvious practical constraints related to the nature of the research and the type of participant sought. However, steps were taken to ensure as wide an input as possible on the research design. Thus, in addition to the literature, fellow colleagues, research students, and grounded theorists all provided guidance and direction. Some technical knowledge about Multimedia development carried over from previous research studies (Webb and Booth, 1995) and informal contacts with the software design community in Northern Ireland provided a context throughout the duration of this study.

4.4.1.4 Measuring what it is intended to measure

At the beginning of the research study a number of informal (pilot) interviews with software designers were conducted. These were not recorded for transcription but focused on defining and refining a set of questions around which subsequent semi-structured interviews could be conducted. Theoretical sensitivity ensured that themes and concepts emerging from the data drove the analysis. In addition, the method of analysis was not used here for the first time but had been developed through the

analysis of four interviews with civil engineers conducted as part of an earlier inquiry into the differences between software engineering and other engineering disciplines[19]

4.4.1.5 Validating the code.

Boyatzis (1998:30) argues that the coding continuum typically reflects the *decreasing* likelihood of achieving higher inter-rater reliability, or consistency of judgement. That is, although the data driven approach is characterised by uncertainty and ambiguity "the closeness of the code to the raw information increases the likelihood that various people examining the raw information will perceive and therefore encode the raw information similarly"(1998:30). Also "because a data driven code is highly sensitive to the context of the raw information, one is more likely to obtain validity against criteria and construct variables". Moreover, validity increases in research studies of multiple units of analysis and comparative studies and as the number of studies using the same type of information increases. Chapter Seven presents one measure of validity wherein a comparison is made between the outcomes of this study and Wernick's (1995) and Gallagher's (1999) Kuhnian analysis of software engineering and graphic design textbooks.

Individual researchers can increase the likelihood of higher validity and reliability by producing good thematic codes as " a good code will have the maximum probability of producing higher interrater reliability and validity scores" (1998:31). Boyatzis goes on to outline the elements of a good thematic code.

1. A label that is close to the data, conceptually meaningful but is clear and concise (requiring minimum interpretation).

---

[19] This line of inquiry was subsequently abandoned when the main focus of study became software design.

2. A definition of what the theme or concept concerns
3. A description of how to know when the theme occurs
4. A descriptions of any qualification or exclusions to the identification of the theme
5. Examples, both positive and negative, to eliminate possible confusion when looking for the theme

These criteria have been used to structure the presentation of the final set of thematic codes in Appendix 5.

4.4.1.6 Use of a Pilot Study

Early in the research project a small number of interviews (4) with civil engineers were conducted, transcribed and analysed using the grounded theory method. This line of inquiry was not subsequently continued but the exercise meant that the transcripts used in the main study were better coded and greater confidence was held – *a priori* – that the instrument was valid.

4.4.2 Reliability

Boyatzis (1998:147-149) points out why the test-retest paradigm so popular in positivist science, has limited application in qualitative studies. Since the goal of qualitative research is to discover something unique about the phenomenon being investigated, using a rich variety of experiences and data sources, it is highly unlikely that the phenomenon would be sufficiently stable over time to allow a retest to take place. In fact "administering the assessment instrument or method twice to the same people at different times or in different settings may in and of itself create an arousal and an additional field experiment".

This does not mean however, that a sample should not be tested comprehensively in the first instance (we must be sure that what we are measuring is representative) or that the qualitative researcher can be negligent in relation to the question of reliability. Boyatzis defines reliability as consistency of judgement over time and space. As such it is closely linked to the concepts of confidence, dependability and credibility.

Black (1993:72) employs Mehrens and Lehmann's (1984) definition of reliability "the degree of consistency between two measures of the same thing" where the measures may be tests, instruments, data sources or methods. Creswell (1994:174) notes that (following Jick, 1979) the concept of triangulation is based on the assumption that any bias inherent in particular data sources, investigator and method would be neutralized when used in conjunction with other data sources, investigators and methods. Boyatzis (1998) points out that whilst steps can be taken during the research to improve reliability measurement, most if not all of that measurement takes place after the completion of the initial research, post facto.

4.4.2.1 Steps taken to improve reliability measurement during the research

4.4.2.1.1 Audio-taping and transcriping interviews

The audio-taping of each interview means that other researchers can have access to exactly the same data sources used in this study, thus facilitating double coding. The transcript of each interview provides a more accessible (albeit less rich) record of the data.

4.4.2.1.2 Clustering of concepts and categories.

Coffey and Atkinson (1996) note that clustering themes can be a useful way to organise code. Boyatzis (1998) observes that clustering can be important for the

presentation of findings and in the formulation of further research. He goes on to identify two broad approaches to the clustering of qualitative data, conceptual and empirical, and suggests that a researcher's preference will be influenced by the coding paradigm adopted (data driven, prior research driven or theory driven). Conceptual clustering involves organising the data around related characteristics, underlying constructs or a hierarchy and is primarily a qualitative approach. Empirical clustering is primarily a quantitative approach employing descriptive or multi-variate statistical techniques.

The approach taken to clustering in this study is, in hindsight, somewhat eclectic, reflecting the hybrid coding paradigm adopted. Nevertheless the process was sufficiently developed to facilitate some check on the consistency of the data analysis, in this thesis and beyond. However the extent to which this can be done is crucially dependent on the quality of the underlying data.

The first phase of data analysis, the open coding stage, used clustering based on related characteristics and underlying constructs to develop categories from concepts. The organisation of concepts into categories and categories into thematic categories and thematic categories into meta-categories (of the paradigm model) is clear evidence of hierarchical clustering. The paradigm model itself is a hierarchy. At its apex is the core category and all other categories are related to this through a series of cause and effect relationships.

The significance of clustering at this juncture in the thesis lies in the opportunity to use quantitative techniques to test the reliability of the qualitative analysis. Empirically created clusters may expose flaws in the original argument, reveal

relationships or underlying constructs not previously detected, or may simply allow the initial analysis to be developed and expanded. Since these tests are conducted *post facto* they are described in the next section.

4.4.2.2 Reliability measurements post facto

Two approaches to reliability measurement are offered, one included in Chapter Seven, the other identified as further research in Chapter Eight (8.6.2). These are categorised according to Jick's (1978) broad interpretation of triangulation, and by Mehrens and Lehmann's (1984) and Boyatzis' (1998) equation of reliability with consistency.

Two sources of reliability measurement are offered in Chapter Seven, derived from Wernick's (1995) study of Software Engineers, and Gallagher's (1999) study of Software Engineers and Graphic Designers. These are summarised in Table 8 and discussed briefly below.

Source data	Investigator	Method	Triangulation of
8 SE texts	Wernick (1995)	Content Analysis	Source data, investigator and method
8 GD texts 4 SE texts	Gallagher (1999)	Content analysis	Source data, investigator, and method

Table 8: Mixed methods reliability measurements (post facto)

This study has identified aspects of the inductive model that are consistent with factors identified by Wernick in his analysis of software engineering texts and by Gallagher in his study of software engineering and graphic design texts. (Chapter Seven). In fact this is also a measure of validity as according to Boyatzis's definition (1988:30) it constitutes another unit of analysis of the same type of information.

Creswell (1994) observes that Denzin (1978) first used the term triangulation in 1978 to argue for the combination of methodologies in the study of the same phenomenon. This combination of methods may be drawn from 'within methods' approaches or from 'between methods' approaches, such as with the mixture of qualitative and quantitative methods. Triangulation remains an important reason to combine qualitative and quantitative methods, including triangulation in the classic sense of seeking convergence of results (Cresswell:174-175). Thus a final set of reliability measurements is discussed wherein statistical analysis is used to further test the qualitative data presented.

Creswell identifies three models of 'between methods' approaches found in the literature. The two phase design approach keeps the approaches completely separate. The dominant-less dominant design has one dominant paradigm with one small component of the overall design drawn from an alternative paradigm. The mixed methodology design mixes aspects of the qualitative and quantitative paradigms at all or many methodological steps in the design. The approach implemented in Chapter Seven and suggested as further work arising from this thesis is closest to the dominant- less dominant design model but is quite specific in that the less dominant quantitative study is being used as a triangulation on the dominant qualitative study.

The basis of the mixed method triangulation suggested in this thesis is the clustering of categories during data analysis. Two levels of triangulation are proposed as further research.

(a) further the use of descriptive statistical techniques to describe the data (for example, the use of histograms or scatter plots to arrange the data along one or more numeric distributions)

(b) the use of multivariate non parametric statistical techniques such as factor and cluster analysis to test and develop the data at a more advanced level. This is crucially dependent on sample size.

Creswell reminds us that such a pragmatic approach is unlikely to please either qualitative or quantitative purists who believe that paradigms should not be mixed. Boyatzis (1998) observes that attempts to quantify approaches such as naturalistic inquiry and hermenuetics may "violate conditions of the methodologies or values embedded in these methods". The same point is made, but more forcefully, by Lee (1999:21) who states

> "how would I respond to a critic who insists that he or she does not understand or agree with my interpretation, and demands conclusive proof? I would argue that such a demand is inappropriate"

Nevertheless, Boyatzis points out that whilst a purely qualitative account is sufficient when "there is no desire to generalise" (as is often the case with the postmodernist approaches mentioned above), descriptive or interpretive methodologies do not *preclude* quantitative analysis. He suggests that even when the intention is not to generalise the researcher should anticipate the readers desire to do so (1998:129). With this in mind the use of quantitative tests to validate the data reported in this thesis is is presented in Chapter Seven and discussed as further research in Chapter Eight.

4.5 Data management

> "each analyst must develop his or her own style for memoing and diagraming. Some may choose to use computer programs, others color coded cards, while others prefer putting typewritten pages into binders, folders, or notebooks. The method you choose is not important, as long as it works for you. What is salient, however, is that your memos and diagrams remain orderly, progressive, systematic, and easily retrievable for sorting and cross-referencing" (Strauss and Corbin, 1990: 200)

It was natural for this researcher to choose a computer-based solution to this data management problem. Even at the outset of the research programme, alternative paper based approaches appeared simultaneously too restrictive and unwieldy (and their obvious limitations need not be documented here).

Barry (1998) cautions against the outright rejection of non computer based approaches and the blind adoption of what she terms Computer Assisted Qualitative Data Analysis Software (CAQDAS). She points out that such action is often associated with inexperienced researchers who use CAQDAS software as a prop, "a way of dealing with the anxiety of those unused to dealing with complex and unstructured datasets" (ibid:3). However she also makes clear the advantages of using such software - it frees the researcher from much of the tedium of data administration and archiving, it provides a powerful tool for data manipulation and analysis and it acts as a medium for communication between researchers.

In this study, the choice of a *specific* computer-based solution was also natural. The researcher at that time was developing a number of relational databases using Microsoft Access RDBMS and it seemed both logical and challenging to seek to apply this technology to grounded theory coding and analysis. Details of the database system used are given in Appendix 2.

Before, during and after the selection of this technical solution, a number of alternative, and uniformly more sophisticated solutions came to light. Features of two of these solutions (relative to the chosen solution) are discussed here in order to provide some context. However, this should not imply that any kind of product appraisal (formal or informal) took place. Rather, a RDBMS approach was decided upon early on, for reasons of convenience, intellectual curiosity and the belief that, to paraphrase Strauss and Corbin, 'it would work for me'.

Barry (1998) has identified two outstanding CAQDAS products. She quotes Weitzman and Miles (1995) who put Atlas/ti and Nudist 'at the head of the field' of coding and theory building software with "the choice between the two are not clear cut". Barry goes on to conceptualise the differences between the products along two dimensions - the structure of the software and the complexity of the research project. The details of her framework and the conclusion of her analysis are not of primary interest here. What is of interest is a feature comparison between the two products and the low tech, homegrown solution developed and used in this research study.

Feature	***Atlas/ti* & Nudist approach**	**DBMS based approach**
User friendly interface	Yes/Yes (but less so)	Yes (but needs customisation)
Predominantly visual	Yes/No (predominantly verbal)	No (predominantly textual)
Sophisticated searches	Yes/Yes	Yes (but requires SQL)
Data admin and archiving	Yes/Yes	Yes
Support for multiple data types	Yes/Yes	Yes
All features on screen at once	Yes/No	No
Limits on coding (size & links)	No/Yes	Yes
Supports hypertext	Yes/ No	No
Project management tools	Yes /Yes	No
Supports multiple researchers	Yes /Yes	No (but may be developed)

Table 9: A comparison of data management solutions

One would expect the highly rated CAQDAS products to out feature the in-house solution. Yet the comparison does show that the in-house solution contains the essential features of such software - a user friendly interface, data administration and

archiving, support for multiple data types and sophisticated searching. As stated previously, the decision to develop an in-house, DBMS based solution to the data management problem was not based on any such *a priori* product comparison. In retrospect however some objective support for the decision taken has been demonstrated.

4.6 Chapter Conclusion

This chapter set out the procedures for data collection and analysis used in this study. Greater attention was given to data analysis reflecting a research objective and claimed contribution of the thesis. A number of assumptions were made that uniquely inform the morphology of the research . These are (a) data not collected by this researcher was used as a primary input (b) software engineers and graphic designers are equally held to be software designers (c) the data analysis was inductive and deductive, data driven and research/ prior data driven (d) a second data set derived from published interviews with software designers (Data-set B) was used to validate the process and outcomes of the main analysis (Data-set A). (e) measures for data validation and reliability are explicitly set out (e) a "home grown" approach to data management was followed. In the next chapter, the first of two that present the outcomes of this study, the research design set out in this chapter is applied to the data.

Chapter Five: Action in context: An inductive model of software design

> "If you watch designers then you will see them scribbling on paper or playing on the computer, but I wouldn't tell them how to work or even pretend to understand how they work" (software engineer)

5.1 Introduction

This chapter presents the outcomes of the open and axial coding procedures described in Chapter Four (4.3.2 and 4.3.3). Given that one of the claimed contributions of this thesis is to make the data analysis process transparent the chapter attempts to (a) integrate the outcomes of the open coding process –first order codes or concepts – into a narrative of software design and (b) develop this narrative analytically using the axial coding procedure of the paradigm model. Notwithstanding the comment made in Chapter One (1.4.3) that many such accounts are elusive, a practical difficulty remains the balance between the two sets of outputs, between description and analysis. In the end it was decided to structure the chapter around the paradigm model framework but, under each major category, to give as full an account as possible of the first order codes or concepts identified and developed during the open coding phase. Thus in this chapter the outputs of the open coding phase are nested within that of the axial coding phase.

Each of the 27 thematic categories generated as a result of the open coding of Data-set A was assigned as a condition, strategy or consequence of the paradigm model (at the macro level of inquiry as described in Chapter Four, 4.3.3). This produced the following organisation

Phenomenon [Functionality, Structure, Presentation, Entropy, Problem Solving]

Causal Conditions [User-Requirements, Designer-Motivation, Designer-Influences, Design-Pulls]

Context [Context-Complexity]

Strategies [Notations, Prototypes, Abstraction, Separation, De-composition, Refinement, Re-use, Storyboards, Iteration, Pragmatism, Communication, Collaboration, Balancing]

Consequences [*Good Design, Bad Design*]

Intervening Conditions [Design-Constraints, Methods]

Within each classification (condition, strategy, consequence) the assigned categories were further examined (at the meso level of inquiry described in Chapter Four, 4.3.3] . One integrative category was identified and all other categories within that classification related to it as properties or sub categories. This produced the following six '*meta*' categories

> **Strategies [action / interaction; problem solving / framing]→ consequences [quality outcomes]; under a set of conditions which are causal [User – Designer] contextual [context-complexity], and intervening [constraints]**

Each of these 'meta' categories and the relationships between categories was then developed further under the micro level of inquiry (as described in 4.3.3). This chapter presents the outcomes of this analysis. Appendix 1 and Appendix 5 list and describe the codes (concepts and categories) used in this account. The chapter concludes with a summary of the analysis in the form of an inductive model of software design and a discussion of its essential features.

5.2 Action and interaction

The designer employs a range of strategies to design. At the lowest level these strategies are a set of "action" processes including, but not limited to, doodling, sketching, drawing, diagramming, flowcharting, abstraction, separation, de-

composition, and refinement. **Notations** are applied at different levels of abstraction and at different stages of the design process. For example, doodling is used to develop initial ideas or concepts (described as "back of the envelope design"[GD1-3]) but also to detail some flowchart or process. The strategies are subjective and relative, not absolute. What is doodling to one designer may be scribbling, sketching or drawing to another. Diagramming and Flowcharting includes Data Flow Diagrams and Object Diagrams that have more defined outcomes and formal execution [SE3-11; SE4-6; SE5-5; SE6-8]

Written documentation is important. This may be something as simple as an outline of what the design is going to do, as in database design [SE8-2], or it may be a more detailed and formal document agreed between designer and user and signed off as a requirements specification [GD2-23]. The purpose of the specification is to capture the design requirements but the document will go through a number of drafts as the design evolves. In one extreme case the designer starts off with a written specification, develops a prototype based on this and goes back and rewrites the written specification based on user reaction to (and his own experience of) the prototype. Only when the written specification is agreed as "correct" does coding commence [SE6-6].

Many designers still sketch or diagram or write on paper, and prefer to do so [for example, SE8-2], but one graphic designer commented " I like to keep things in my head, I hate writing things on paper, it's not necessary" [GD3-6]. There is evidence of increased use of computer based tools. The benefits of using computers to do design notations are recognised but so to are the dangers. A graphic designer drew a parallel with the advent of desktop publishing when the ready availability of technology to

produce documents resulted in a general deterioration of quality and warned, "the computer is just a tool it will not design for you" [GD1-3].

Despite the widespread use of **notations** it is clear that much design knowledge is not written down. This may be the result of a deliberate strategy, when for example, the designer feels that he or she has sufficient experience and cognitive faculty to work solely from memory [GD3-1]. It may be a consequence of time constraints that exclude documentation or it may simply be a feature of the design process itself, as for example, when a small group of designers share knowledge but only because "it is in people's heads" ([SE1-18]).

Unsurprisingly given the nature of the field, there is a good deal of evidence of, and support for, **prototypes**. Prototypes are used for a number of purposes – to capture requirements, to communicate the design to others, to act as a quasi contract - " If a customer signs off the prototype then you've got him. If they approve something then they can't come back later and say that they don't like it" [SE3-5].

In fact this is what users do frequently and ironically the same designer went on to detail a design approach that appeared to be based on just this eventuality [SE3-11]. However, the comment does illustrate a feature of the prototypes described and demonstrated by many of the interviewees. Prototypes were rarely thrown away (although some were) but invariably used for further development. This was the case whether the prototype was an interface design, a running program or a storyboard. This departure from Brook's (1975) maxim of "build one to throw away" is in part explained by an innate reluctance by designers to abandon code once written ("we're all hackers at heart"[SE4-1]). In part by recognition of the importance of prototypes

in a visual environment and of the significance they assume as the design evolves. This approach has broader implications for practice and these are discussed in Chapter Eight [8.2.3.3].

The relative importance of the prototype to the overall design will influence who develops it. If it is a small 'skeletal' prototype then it will be developed by the graphic designer using for example HyperCard or Hyper-Studio or Director but if it's a larger prototype requiring 'complicated multimedia' then it is probably left to a programmer [GD1-7].

Abstraction and **separation** are particularly important to software engineers. "Look at a problem, identify the key elements, separate these out", "think about one thing at a time" "decide which to tackle first and which to give overall priority" [SE1-3; SE1-13]. Software engineers seek to separate functionality from implementation or presentation. This should result in a, preferably formal, functional specification. Doing this is becoming more difficult because the interface is becoming much richer and more complex, certainly it is much more difficult than in conventional software (where "we don't design a new keyboard every time we design a new system" [SE1-20]). For example it is becoming more difficult to recognise when something causes a change of state on the system as opposed to a change of state on the interface [SE1-17].

Nevertheless, software engineers claim to be better able to do this than graphic designers. This is a key professional skill (along with abstraction, generalisation, re-use and de-bugging skills). Software engineers have "a certain level of confidence with dealing with the technology" and they bring "an awareness of how complexity

impacts on and is impacted by technology" [SE1-21] This is much more than the programming knowledge or technical knowledge they are commonly assumed to have, it is the "appreciation of programs as interactive things" [SE1-21]. Whereas the "media oriented people won't have the same degree of confidence" and the graphic designer "who is good at designing a page, hasn't even touched on the kind of complexity that we are used to dealing with" [SE1-21].

Solving a problem by progressively breaking it down into smaller and more manageable pieces [**Decomposition**] is a generic design strategy, one that was referred to directly and indirectly in the transcripts. One graphic designer described the design process as follows. The designer does the research, collects the content, develops the initial menu and outline modules, then develops a series of thumbnails that are transferred to the computer. "The design of it is this [points to thumbnails] taken to a more complete, finalised level of detail" [GD3-7]. When it was suggested that this was really a process of adding detail to an architectural design or broad structure, he agreed but added that the process was not linear

> "Yes there are different levels, but I try to work through them together. These buttons here [points to screen design], I didn't know if they were going to be in it" [GD3-13].

indicating that this was a function of both logic and invention

> "I also try to develop different screens at the same time only for the reason that you can get stuck on one thing for a week which can be quite boring" [GD3-7]

Design is often a long and difficult process, with no quick fix, but rather a period of "wrestling with the problem" [GD2-20] until a satisfactory (to the designer) solution emerges. Knowledge is often tacit and sometimes it is difficult for the designer to explain how or why a solution is reached

> "When a designer gets to this point it is difficult to explain why he or she is at that point, but they themselves will just know. This is a problem that most students face - it takes them a

> while before they realise that they have actually solved the problem. After a certain period of design time a designer will know that it is right" [GD2-20].

One designer likened design to driving a car "once you learn you never really think how you do it" [GD2-5]. Another appeared to allude to a bike riding analogy observing that aspects of design (in this case the technique of separating out buttons on the interface) "might be really tedious, but if you've done it once you can do it again" [GD3-8].

Given a design problem the designer seeks to solve it through **de-composition**, **abstraction** and **separation** but these strategies themselves rely on the strategy of **refinement.** Abstraction and separation is only possible after the problem has been made more manageable through a process of refinement, **iteration** and trial and error [SE4-9]. In this sense then, refinement is not an independent strategy but one that occurs in conjunction with, and which facilitates, other strategies [GD3-10].

Design is an iterative process through which the design evolves over a period of time. Frequent iterations occur between the designer and the user, or client or customer[20], between designers, and between designers and managers. These iteration start at the very beginning of the design process and continue until the design is delivered and beyond. The purpose of each is to develop the design further - by clarifying user requirements for example or to test a piece of code. A graphic designer likened the entire development process to a process of design **refinement** [GD4-5]. A software

---

[20] In the transcripts the term 'customer', 'client' and 'user' are used frequently without being defined. Webb (1996) defines a user as someone who actually uses (or will use) the system. A customer or client is someone who purchases or commissions the system. A customer or client may also be a user but the respective roles are distinct. He also notes that, based on his research into Multimedia product design processes (Webb and Booth, 1995), designers rarely get to talk to actual users. For the purposes of this study then, although the terms customer, client and user are used frequently by the interviewees, the reader may assume that in the majority of cases the designer is referring to a customer or client.

engineer described design as a process of trial and error – the designer will try out various things to see if they work. This involves "throwing ideas back and forward" and gradually "building up a picture of the solution" [SE4-9]. This approach makes accurate estimations of time and budget difficult if not impossible. Refinement is most commonly facilitated through the use, and re-use, of **prototypes**.

In addition to prototypes, **re-use** is made of code, methods, tools, techniques and experience - and also the design itself. Code re-use may be "haphazard", possible only because in a small team "it's in peoples heads" [SE1-18] or, in one case, when as a result of a quality assurance initiative (ISO9000) "a library of routines which are well and truly tested" had been built up [SE8-1]. Alternatively code is re-used on an individual basis

> "I would just be trying out different bits of code based on past experience - I know that this is similar to something else so I would plug that bit in. I may have another bit of code that I've used elsewhere so I'll reuse that" [SE8-2].

Knowing what and when to re-use is regarded as an important design skill, comparable to generalisation, **abstraction** and de-bugging [SE1-22]. **Good design** can be re-used whereas **bad design** is "ad-hoc, you can't repeat or re-use or stand over it"[SE3-12].

A pragmatic approach to design is illustrated in a reluctance to commit the design to paper but also in an impatience to "just get on with it" to "get things done"[SE2-7]. **Pragmatism** is partly motivated by a suspicion of theory or at least recognition of its limitations - "I'm not into theory as such only with what works … you don't go in there with it [theory] in your head"[GD2-17]; "basically you have to get down and do the work" [GD3-8].

Each strategy produces its own outcomes. **Abstraction** is used to "identify key elements", "separate these out" and "decide which to tackle first". The intended outcome of abstraction is a high level conceptual model of the design. **Separation** separates functionality from presentation; **de-composition** breaks the problem down into smaller, more manageable pieces; **refinement** is a process of "trial and error", "gestation" and "elimination" that proceeds incrementally towards a solution, most notably through the use of **prototypes**.

The outcome of any strategy may not be what was intended, or an intended outcome may be offset by some other (intended or unintended) consequence of that action or another action. The need to **balance** the constraints of time and cost is a good example of this. Given specific temporal and financial targets, the designer must inevitably **compromise.** In practice, the level of **abstraction**, the degree of **separation** or the period of **refinement** is limited by the need to deliver the design on time and within budget. Compromise too has its consequences, as the quality of the design artefact is calibrated to the amount of time and money available. There is ample evidence in their own process descriptions and in comments made by software engineers to suggest that graphic designers often find such compromise difficult [GD1-10; GD2-7; SE1-21; SE3-10].

Sometimes the outcome of a strategy is unexpected but favourable nevertheless. As already noted, a number of designers spoke of not really understanding how they designed or of not really knowing how or why a particular outcome was achieved [GD2-20; SE4-4]. This makes prediction and estimation of time and cost difficult [SE4-9]. Where the outcome is positive, the strategy is most likely to be repeated, often but not always, accompanied by some attempt to rationalise the process. Acting

on intuition or faith - found in the descriptions of novice and experienced designers, within and between disciplines - has its own set of consequences. Significant changes in design context may render a previously successful strategy redundant and precisely because the strategy is not understood in the first place it is much less likely that any mismatch between problem environment and proposed solution will be noticed and acted upon.

Also of interest, aside from the consequences of individual strategies, is the sequence of action/interaction strategies and their outcomes wherein the output of one strategy becomes the input for another. Thus **abstraction**, **decomposition** and **refinement** result in a conceptual model that is used to write a specification that in turn is used to design the final artefact. In practice, as noted earlier, there is considerable iteration within and between activities. Action processes take place in a sequence as illustrated by the following protocol

> You design using notations like JSP, flowcharts, and graphs. You break it up [decompose and modularise]; then you would flowchart that (in terms of functions and content). JSP is used along with the flowcharting and that is then taken directly to code. [SE8-2]

The consequences of discrete and sequential strategies, intended and unintended, affect future design strategies. What works is likely to be repeated and what doesn't work rejected or modified. A failed strategy may discourage experimentation or it may not, depending on, inter alia, the personality of the designer, organisational culture, and resources available. An innovative and successful strategy may be developed as an in-house method or it may not, depending on the same conditions, and others. What is clear is the potential complexity of even the simplest causal relationship due to multiple conditions in operation at any point in time and the

certainty of process or change. (Chapter Six develops four predictive design scenarios and considers broader conditions that impact upon design strategies).

5.3 Interaction

Two types of interaction were identified. *Interaction with the materials of design* is broadly equivalent to Schon's (1983) "reflection in action". Here the designer interacts with the design materials - the notations, prototypes and other artefacts identified at the action level. As a result of this interaction both the design and the designer's perception of the design change. Schon refers to this dialectic relationship as 'reflection on action' but the idea of changing self as a result of interaction with work can be traced back directly to the writings of Hegel and Marx. The causes of such self-reflection were not specifically studied in this research but other researchers (Winograd and Flores, 1987, for example) have suggested that it occurs when there is a breakdown of some sort in the normal activity of design. For example when a particular process, or method, or tool no longer works, or the designer's technical knowledge proves inadequate. This notion of breakdown also underpins Schon's observations, informed, as were those of Winograd and Flores , by the Heideggerian idea of "being in the world" and, in this thesis, is further considered in light of additional evidence and analysis in Chapter Six.

Interaction with others relates to Strauss and Corbin's (1990:158) original definition of the term "people doing things together or with respect to one another in regards to the phenomenon". Such interaction does not preclude action. Indeed interaction with others would be impossible without related action processes such as thinking, talking, and writing. Moreover the use of **notations** as communication tools has already been noted. But in this category these action processes are directly related to the interaction

and are not executed independently of it. For example, **iteration** is an important action strategy but iteration is impossible without some degree of interaction (at minimum some interaction with the materials of the design must occur). In practice design iterations involve frequent and significant *interaction with others.* This is what Strauss and Corbin (1990) mean when they described action and interaction as *related processes.*

Interaction with others takes two principle forms and each is interrelated. A designer must **communicate** with users or clients or customers in order to define the problem and to build and test the design solution. This is evident in the frequency and duration of design **iterations** and in the emphasis on **prototyping**. The designer must communicate with other designers, or programmers or other members of the design team in order to complete the design and to verify its quality. Code reviews or formal 'crits' in which other designers evaluate the design are just two examples of this [SE6-1; SE6-2]. Finally, the designer must also communicate with managers or other persons in authority in order to secure the resources necessary to do the design. Hence, designers are concerned to be involved in the design process as early possible, to "make sure everyone understand what needs to be done" [SE3-11].

Communication difficulties between designers and users are implicit in much of what was said about requirements. Clients may not be able to communicate their needs to the designer, the designer may misunderstand or misinterpret what is said, they may fail to adequately convey the design to the client. Designers from each discipline described design processes that were front ended by extensive attempts to communicate with, and get agreement from, the client. This is evident not only in early and frequent meetings with clients but in the frequent iterations in design back

to the client to seek approval and in the emphasis to get the client to sign off on an agreed design (normally in the form of a prototype) as soon as possible in the process [GD7-4; SE6-6; GD2-23; SE3-4]. These attempts appear to have been only partly successful, at least they do not appear to have adequately addressed the problems associated with changing user requirements.

Communication difficulties between designers also exist and have an equally important influence in determining design context. Much of these difficulties come down to the different education, training and experiences of individual designers. In particular, as highlighted by Gallagher and Webb (1997) problems arise when the same term is used to mean different things. One example of this is the different interpretation of the word '**structure**' given by each discipline. When software engineers talk about the structure of the system they are invariably referring to data structures and algorithms, the 'back end' of the functionality. In contrast, when graphic designers talk about structure they are usually referring to the content, the information structure or the structure of the interface (the 'front end'). One designer noted "the trouble with multimedia is that we don't have the language and we are crashing along at a fast pace with little in the way of common language - we need this"[GD7-2]. There is no conclusive evidence from this data that these problems have been overcome but this and cognate research (Webb and Booth, 1995, Webb, 1996, Gallagher and Webb, 1997) has identified the issue.

There is ample evidence that the design itself, primarily but not exclusively in the form of **prototypes** and **storyboards**, is vital to **communication**. Storyboards in particular act as a point of reference within and between disciplines in multimedia

software design. This role is illustrated in the following comment by a software engineer

> "It's like a common denominator - the software engineer will develop it in one way and the graphic designer develops it in another way. [for example the software engineer may use it to develop a process flowchart, the graphic designer to plan and structure content] It's a common reference for both. They can both say that "Well at this stage back here we agreed this, this, and this" and that can be signed off by both." [SE6-9]

The **storyboard** is also vitally important in communicating with clients who "like to see some pictures" and "understand and appreciate prototypes"[GD7-1]. For these reasons the storyboard occupies a unique position in multimedia development ("the storyboard is number one" [GD7-1]) and it is unsurprising that they almost always evolve incrementally into the final product. Of course, as with all prototypes this has its own dangers (these are discussed in Chapter Eight).

Communication is a strategy - one means of managing or coping with design – but it is also a causal condition, a pre-requisite to any design effort. It is also an intervening condition, coming between a design strategy and its consequences[21]. The quality of communication between designer and designer and between designer and user may determine whether the design is good [SE5-2] or bad [GD2-4]. At least one software engineer bemoaned the lack of training given in this important interaction strategy [SE6-2]. Communication is both a cause and an effect of **collaboration.** Communication can lead to collaboration and collaboration depends upon and may increase communication, Thus communication and collaboration are related interactions.

---

[21] This is consistent with the coding procedure used since discussion under one aspect of the paradigm does not rule out discussion under another. On the contrary it adds variety and richness to the category.

Collaboration ranges from informal contacts between individual designers to more formal arrangements within or between design teams. However, where the organisational unit is small and the culture is informal (as was the case in the vast majority of companies in Data-set A), it is often difficult to discriminate between different levels of collaboration, and its quality. A "Monday meeting" constitutes structured collaboration between managers and designers but it is possible that other reported 'collaborations' - in the corridor, over coffee, after work - are equally, if not more, significant. Moreover, what constitutes formal collaboration to one designer may be an informal or casual get-together to another. Despite this semantic difficulty it is safe to say that the software design process involves frequent and extensive collaboration [GD2-2; SE2-8; GD2-11].

It is much more difficult to discern from interview transcripts the recipe for a successful collaborative effort. Again, however, some pointers may be offered based on a summarising of salient comments. Teams, which in multimedia development are necessarily inter-disciplinary, should be kept small, one software engineer suggested between five and seven members, one graphic designer suggested no more than four [GD5-4]. It is advisable to have clear reporting structures. A graphic designer lamented the earlier imposition of matrix structure within the organisation wherein software engineers and graphic designers were contracted out to projects as and when the need arose. This resulted in a lack of coherence within teams and a lack of commitment by individual designers. The alternative organisation into project groups worked better because the teams were kept small and informal, with each designer performing a number of roles, and the leadership is rotated between team members according to project (even junior designer are allowed to lead non critical tasks).

Where a software engineer is leading a graphic designer or vice versa (the latter is less common) there is a need to ensure that ignorance of the subtleties of the job does not result in the repeated allocation of unrewarding or boring tasks. This applies also to specialist project managers and to general managers [GD1-9].

Whatever the particular structural arrangements, it is clear that successful collaboration depends crucially on less tangible factors. The personality of individual designers is important because contact is often prolonged and intimate. So, “you need to get on with people who are involved with the project” [SE6-7], "I see more of [software engineer] than I do of my girlfriend"[SE6-2] and "if you get a real jerk its a disaster for all concerned"[GD6-1]. Where inter-disciplinary collaboration is concerned, the attitudes of one discipline towards the other can make or break the design effort. This transcends any organisational or structural arrangement. Thus successful collaboration was reported where two disciplinary teams were separated along strict functional lines and a black or grey box approach to design operated whereby software engineers did the functionality and then "threw it over the wall" to the graphic designers who did the interface. [SE4-5] On the other hand, an apparently identical arrangement elsewhere resulted in "gridlock between the disciplines". The software engineers developed functionality, passed it over to the graphic designers for the addition of presentation, the graphic designer's changed the design, the software engineers had to re-code it and so on until the project spun out of control and eventually aborted [SE4-5].

Design teams frequently involve clients, particularly in the early stages of the process where requirements are gathered and analysed. One designer referred to “brainstorming” sessions involving designers and clients used to generate a

specification or prototype [SE5-3]. Although the direct involvement of clients in the design process appeared to diminish as the process evolved, in the majority of processes outline approval was sought from the clients at significant milestones and before major commitments were made. In part this is a reflection of perceived good practice, in part a measure of the concern most designers had about changing user requirements, budget excesses and time overruns.

5.4 Action / Interaction as related sequences

Action and Interaction are purposeful, normally done for a reason but sometimes are unplanned as when the need for interaction arises out of a sequence of action processes. For example the designer in developing a solution may realise that further data are needed and will interact with the client to obtain this. Similarly, during an interaction, the designer may break off to perform some action process considered necessary before the design can proceed. Therefore action and interaction are related sequences (related through sets of action and interaction processes) but the need for action can interrupt interaction and vice versa.

Prototyping is a good example of this process. Based on an initial explanation of user requirements [interaction] the designer will build a prototype [action]. This prototype will be shown to the users who will comment upon its features [interaction] and the designer will return to modify the prototype [action] in line with these comments. This sequence of action and interaction continues until the prototype is acceptable to the client. The process is further illustrated in the following protocol

> "What we will try to do is develop a proposal, which we will have discussed with the client - we try to nail down as much as possible. From there we try to move towards a more formal, tighter functional specification. The coding team will then go on to code based on that functional specification. If the customer won't sign the functional specification, then we will not code it. [GD1-9]

Here discussions with the client [interaction] lead to the development of a proposal [action] that is discussed with the client -'we try to nail down as much as possible' - [interaction]. As a result of this a functional specification is written [action] which is discussed with the client [interaction]. If the client agrees with this specification by signing it off [action] then coding begins [action].

Because action / interaction is a related sequence it is possible to observe them over time and to account for process or change. Thus, we can comment not only on current action/interaction but on how previous action/interaction influenced this and how current action/interaction may influence action/interaction in the future. The action strategy **re-use** encapsulates this process. Faced with a design problem, the designer may decide that this problem is sufficiently similar to a previous problem that existing expertise, experience, methods, tools or techniques can be applied to its resolution. Similarly, after the completion of this design task (successfully or otherwise) the expertise, experience, methods, tools or techniques developed may be brought to bear on a future design problem. This is not to argue that all design projects are the same but to point out that re-use is an important action strategy that links past, present and future design efforts. Certainly what works is likely to be repeated.

Action / Interaction results in consequences but are mitigated, to a greater or lesser extent, by a set of conditions. These conditions include those causal, contextual and intervening conditions first introduced in Chapter Three.

5.5 Conditions

Each designer approaches a design problem with a unique blend of personality, education and experience. These critically determine the designer's initial **motivation**

for doing a design and the quality of the resultant effort and outcome. For example some designer's find compromise difficult, some lack training in important communication skills, others lack the experience to manage large and complex design projects. These factors in isolation or in combination intervene between a chosen action / interaction strategy and its consequences. Thus, one strategy may work for one designer but not another, or broadly similar results may be obtained from very different strategies.

The designer brings to the design his or her own prejudices and experiences. These too will influence the problem and the means of its resolution. Professional experience, education and training can act as facilitators or constraints in the development of a design solution (as such they operate as intervening conditions) but they also influence the designer's perception of the problem and his or her approach to solving it. Therefore these are also important causal conditions that lead to "the occurrence or development of a phenomenon" (Strauss and Corbin, 1990:96).

It is impossible to identify from interview transcripts all or even the most salient psychological factors that **influence** an individual designer. Nevertheless a number of broad observations are possible based on personal accounts of the design process. Designers approach a problem with an initial idea or vision "you've got to have the vision to start with and then you refine that"[SE3-2]. Clearly designers do not "start with nothing" as one interviewee put it [SE5-1]. More experienced designers have a number of different approaches or methods to call upon, which they use and when, is determined by context. Each approach is based on a combination of formal knowledge and tacit knowledge or heuristics. Experienced designers appear to be able to switch

between approaches according to the nature of the problem environment and of the emerging solution.

As in other design fields, the key competence appears to lie not in the possession of technical knowledge but in its application. Knowing what to do and when is a significant point of departure between experienced and inexperienced designers. So whereas inexperienced designers may not be aware how, or how well, they have applied this knowledge, " after a certain period of time a designer (experienced) will know that it is right" [GD2-20].

It is axiomatic that design will not happen unless a designer agrees to do the design. Evidence of a designer's **motivation** to do design can be found in direct comments, in descriptions of good design and of design goals but also detected in asides and observations apparently unrelated to the subject. Graphic designer's may be initially motivated by an interest in fine or visual art but this will be tempered by a need to address the more practical aspects of design, such as the need to communicate effectively [GD8-1]. Software Engineers are motivated by, among other things, coming up with a solution to the problem that is efficient, elegant but reliable. [SE8-2] Designers of both professions share common motivations - solving a problem, managing complexity, overcoming constraints, satisfying users (even making them happy), making an impact (in part to attract future work), being original or innovative, [SE4-7; GD4-7; SE2-3; GD1-8; SE1-5; GD5-7; GD2-22].

Whatever the **motivation**, and it is clear that a number of motivations will be in operation and will change over time, the designer must want to design. As one graphic designer put it "The desire has got to be there. Design is deeper than just a set of

drawings"[GD6-1]. Yet it is also clear that the motivation is not always positive or at least a positive motivation may be combined with, or even over-shadowed by, a negative motivation. Again, although none of the interviewees were questioned on this issue directly, indications of external pressures to do design can be found in the interviews - mostly in discussion of design constraints [GD4-8]. The imperative to take a commission because the company simply needed the work, or because the work either resulted from previous (perhaps more interesting) work or might lead to such work in the future was a factor in many cases. Also included in this category is re-work and maintenance. Equally clear was the fact that work taken on mostly or substantially for these reasons is less likely to be done to a high standard. One designer referring to the constraints under which he worked, illustrated the negative consequences of an accumulation of design work that is of little or no interest to the designer.

> "I feel like a wee battery hen sometimes dishing out screens. I don't really take enough pride in my work"[GD5-1].

Yet a designer may come to design through a less obvious causal route. Here, the motivation to design is less tangible, in some instances almost imperceptible. This motivation was recorded as the **pull of complexity, the pull of technology** and **the pull of aesthetics.** It affects both design professions, though unequally. Software designers are more prone to be pulled by complexity and technology and graphic designers by aesthetics. Nevertheless the essence of the process observed in each case is the same - the designer appears to be pulled into the design by some integral element of the design process itself.

The danger of designing Multimedia was identified - it can seduce the designer into "making arbitrarily complex things"[SE1-4]. This is to be resisted and in part this

comes with experience. Coping with complexity is a major challenge for software engineers. As noted earlier, they believe that they face levels of design complexity that graphic designers have yet to experience. Avoiding complexity altogether is one strategy, minimising it through hiding is another. "One of the beauties of software engineering is that you can produce a nicely structured, simple, consistent package that can be very complicated"[SE2-2] and “simplicity and elegance are the thing for me” [SE4-11]. At the very least complexity must be respected [SE1-14] and coped with [SE1-21].

There was much less direct evidence that Graphic designers were being seduced by complexity, or that they derived the same satisfaction from dealing with it. This does not mean necessarily that they were, as alleged by some software engineers, unable or unwilling to confront complexity, but simply that on the basis of these interviews it was less of an issue for this discipline. Moreover there was some evidence that as graphic designers become “ increasingly technically competent” [GD1-11] that this situation may change.

Designers of both disciplines may also be seduced by the **technology**. This can be evident in the use of “non relevant technology”, as for example when a designer wants to 'show off' technical knowledge by including the ASCII codes for characters [SE1-4], or use flash technology [GD1-4] or a lens flare [GD3-2] for no real reason. Such 'indulgence' acts as a barrier to readability and is to be avoided. The consequences of being seduced by the technology are evident – “the designer may get a buzz out of it , while others will think that it is awful” [GD2-10]. It can and does lead to poor design - design that is “cluttered”, “overly complicated”, “non – intuitive”, “unnatural” [GD6-13; SE2-2].

Graphic Design is more closely associated with **aesthetics** and this was reflected in the interviews. Although one software engineer spoke at length about the aesthetics of designing algorithms [SE1-9], aesthetics is not a major issue for software engineers. One graphic designer complained "Graphic design very often falls into the trap of people thinking that it is just about aesthetics because it appears to give what many refer to as a surface treatment"[GD2-9]. Here too, there is a danger that individual designers will become absorbed in one aspect of the design to the detriment of the whole. This view holds that such "indulgence" by the designer is intolerable. The multimedia designer is not an artist with freedom to pursue his own interests, [GD2-7] nor even does he have the latitude that traditional graphic designers enjoy. Multimedia design must be accountable to the user and too much concern with aesthetics will result in usability problems [GD4-10]. This said, every designer brings his or her own interpretation to each design.

Of course, designers may *enjoy* grappling with complexity or playing with the technology or creating aesthetics; and this may be one of the positive motivating factors that lead them to design. In this scenario the designer appears not to be consciously making the decision to engage in design but is being drawn in to an engagement, at least partially, by some aspect of the design process itself. It may be that the level and type of such engagement is more significant after a decision has been made to commence design and that the influence of these factors in initiating a design project is peripheral. The fact remains that these factors have been identified as causal conditions of the phenomenon and therefore influence both the nature of the problem, the context within which it is addressed and the strategies used to resolve it.

Of interest is the impact such causal motivations have on the quality of the design. The dangers of being seduced by complexity, technology and aesthetics have been mentioned but what of the advantages? Clearly "enjoying it" is a positive emotion; being "indulgent" is not necessarily a bad thing. A psychoanalytic analysis of such activity may conclude that it is these very moments that constitute the 'white heat' of design, that it is here creativity springs from the unconscious and the designer operates unfettered by any material constraints[22] It is impossible to probe such depths on the basis of interview transcripts but preliminary evidence (induced and deduced) suggests that such a study may be significant in these cases. Of particular interest is the process through which a designer, initially pulled into a design, takes control, or assumes to do so, and what was involuntary becomes voluntary.

So a designer may experience two types of motivation to do a design. **Intrinsic motivations** are intrinsic to the design or intrinsic to the designer. Those intrinsic to the design include the pulls of complexity, technology and aesthetics. Those intrinsic to the designer include both positive and negative motivating factors –for example, having fun or the fear of failure. **Extrinsic motivations** are those external to the design, and to the designer, and include external pressures such as "having to do it". In this thesis we are concerned with all types because all types help explain how and why a designer designs.

The influence on individual design strategies of cultural factors was noted. Graphic designers more easily identified these influences - previous genres of art and design, cinema, books, television, travel, fashion. One remarked " The further that you cast

---

[22] See Csikszentmihalyi (1988) and Simon (1988) for a discussion of creativity and design.

your net the better - you can't learn enough" [GD5-7] and another concluded "so, for me, the educational experience is probably the least influential" [GD2-1]. Software engineers tended to emphasise experience and technical knowledge over 'aesthetic' issues. However some did admit to being influenced by tales of specific design projects (in addition to the ubiquitous Brooks (1975), Pascal Zachery's (1994) account of the development of Windows NT and Fred Moody's (1995) insight into Microsoft's approach to multimedia development were mentioned). It is impossible to be precise on the exact nature (direction and weight) of such cultural influences on the basis of this data. Other studies may be able to identify specific influences in specific designs, and to account for their relative importance in decision making. Here cultural influences are flagged as one of the varied inputs to individual design strategies.

Beyond the individual level lies the group and organisational level wherein an additional set of conditions bear upon action and interaction. Each design strategy is influenced by the design culture and environment of the organisation, or organisational unit, within which it is enacted. The organisation's policy may insist on the use of a particular methodology, or language; designers may lack sufficient, or sufficiently relevant, task or domain knowledge to sustain the approach taken; or management may simply lack the skills to resource and manage a large and complex project. When one or more of these, or similar, conditions exist the design strategy is, to a greater or lesser extent, prescribed. The designer is not free to pursue the design solution he or she prefers (even though this strategy itself is already moderated by conditions pertaining at the individual level [GD2-7]) but must further modify the approach or abandon it altogether.

Designers from both disciplines acknowledge the impact of **constraints** upon design. The most significant constraint identified (measured by frequency of occurrence across both disciplines) is *lack of time*[23]. The following comments are representative of the problem faced. "There will always be a problem with deadlines" [GD2-14]. "I think that there is a problem of allocating enough time for the design process" [SE4-9]. "Depending on the budget and time that whole process can be telescoped into a very short period of time"[GD1-3]. "I only have a certain amount of time to convince the customer to buy this service so I need to get something that is concrete as quickly as possible" [SE3-7].

There is a feeling that users and managers do not appreciate how long it takes to do design. " That designers just sit and doodle and that they can come up with something overnight" [SE4-9] whereas "trying to work up a coherent set of designs for something like an icon family will take a significant length of time"[SE4-9]. In addition because design is a process of refinement and trial and error - it is often difficult to estimate the amount of time that a design may take. "I know that it will take a long time, but it is difficult to gauge this in terms of hours, days, or weeks"[SE4-9].

Technical constraints receive greater emphasis from software engineers. These include space on the disk, memory on the system, processing power but also the target platform. For example, where a system is developed on a Macintosh but implemented on a PC this "can be a shambles" unless thought about in advance [SE1-15]. One software engineer made an unfavourable comparison between PC and Mac

---

[23] As Oshagbemi (1998) points out, a failure to manage time is the actual constraint.

windowing environments and concluded "that's where multimedia falls down - it's flaky"[SE2-1]. Sometimes technical constraints are overcome, not by a conscious strategy on behalf of the designers, but by serendipity and patience. For example, when a performance bottle neck is resolved by the arrival of greater processing power, or more disk access, or the development of a more efficient programming language [SE1-15].

Some software engineers believe that their concern with technical constraints is a significant point of departure with the graphic design approach. Whereas software engineers are aware of these issues "very often the graphic designers won't be as fully aware" [SE1-15] so for example software engineers can see the limitations of authoring tools, graphic designers cannot. In addition software engineers claim they can come up with novel ways to overcome or by-pass technical constraints whereas multimedia designers who know about authoring systems "wouldn't have been able to start" [SE1-21]. It may also be the case that software engineers draw on their technical knowledge to actually encourage graphic designers to design something novel [SE1-1]. In this case technical knowledge is used to predict chances of success.

One software engineer described his frustration when working with a graphic designer on a multimedia kiosk system. The graphic designer had designed an interface that, although aesthetically impressive, did not conform to the hardware features of the system, in particular it did not allow the software engineer to implement the interface in an efficient manner. The issue centred on the graphic designer moving some text fractionally to allow the software engineer to better segment the screen, to implement a grid structure of images and hot spots on an HTML page.

> "Now no matter what way we tried to cut the grid it would never fit the designer's layout. The solution required a level of precision which you just can't achieve - it just wasn't feasible" [SE3-10]

The graphic designer was asked to amend his design, so that the screen could be more efficiently programmed but he refused, preferring instead to re-design the entire interface around the (now obvious) technical constraints. This according to the software engineer was an indication of the graphic designer's lack of technical knowledge in the first instance followed by inflexibility [SE3-10].

Yet, this case is not representative of the sample. Graphic designers may not have the same level of technical knowledge as software engineers, and therefore cannot be expected to anticipate and manage technical problems to the same extent, but there is evidence that they are increasingly aware of these and other constraints. Indeed one graphic designer defined design as "one big constraint" [GD2-10] and listed size of the job, complexity, the users/target audience, expertise within the group, the facilities in the company, budget and time as factors which determined his approach. Further evidence *of constraint awareness* amongst graphic designers may be found in comments such as "I just hate the way that everything is so rigid. To me this is like accountancy" [GD5-1] and, more specifically on user constraints,

> "I think it's...the client is very important because s/he at the end of the day controls what it is that you are doing. If the client doesn't like something then you are not going to use it. That would be a major constraint"[GD5-3].

Design is a **balancing** act whereby the constraints of time, cost, functionality, structure and aesthetics are managed. Design "is always trying to maintain the balance" and the objective is to produce "a quality product within resources" [GD1-4]. This is a difficult task, "there is nothing which is key - there are just so many parts

that you have to pay attention to and make sure that they are all done correctly"[GD2-13], and involves multiple design iterations. For example

> "you can produce beautifully readable code that is written in a simple way that makes it readable, but it is clumsy during execution, on the other hand you can produce something that is really efficient yet incomprehensible. The importance of balancing the constraints is what drives the elegance for me" [SE1-5].

Balancing is also a strategy – a way of coping with or managing design. Here it has been discussed as an action applied to **constraints**. These constraints may be technical or user imposed [GD5-5; GD3-10]. Interestingly two graphic designers spoke of balancing different media as a *marriage*, [GD2-11; GD3-19] and the notion of compromise or trade off underlines almost all comments on this category.

The Web was given as an example of the need to balance constraints, in this instance the balance between aesthetics and functionality, between having a visually appealing interface and the need to factor in download time [GD1-4]. The ability to balance constraints is what sets out a good designer for it is the challenge of balancing constraints that fosters creativity and inventiveness. A good designer recognises that the need to balance constraints will mean that compromise will be necessary and that sometimes, "what can start out as a great idea can become so diluted at the end of it to make it beyond recognition" [GD1-10]. This said, both disciplines recognised the value of constraints in bounding the problem – "if you didn't have design constraints you would be lost" [GD2-10].

The use of **methods** is a good example of how something can simultaneously facilitate and constrain design. On the one hand a method provides a defined approach to problem resolution, on the other it imposes restrictions on the creative process. How a method is seen, whether it is embraced as an aid to the design process, whether

it is rejected as an intrusion upon creativity, or whether- as is more likely - it is viewed with ambivalence, depends upon the specific details of its implementation. These include the 'fit' of the method to the design environment shaped by the specifics of the design problem but comprising its users (designers, users, clients, programmers and significant others involved in delivering the design) and the wider design culture within the organisation.

Organisational culture - here defined as widely shared beliefs and values about design - is in turn shaped by structural factors such as the age and size of the organisation and by collective experiences. In Data-set A, the nature of the observed phenomenon (multimedia design) and the geographic location in which that took place (Northern Ireland) combined to determine organisational cultures that were - in the main - at least ambivalent towards methods. Evidence for this is found not only in comments made by the interviewees but in observations of the design processes operating within the organisations and in particular in the lack of formalisation and documentation of activities [eg. SE2-8/9/10].

Unsurprisingly, the smaller and younger the organisation, the less likely was it to have adopted formal approaches to design. Comments such as "haphazard" [SE1-18] and "chaotic" [SE5-6] were used to describe the processes in some of these organisations. However there was also recognition that "this was a bad thing" and that greater formality was needed [SE5-6]. Exactly what form that formality should take was not clear. A few software engineers talked about software engineering approaches such as JSD and OMT; one mentioned the formal methods approaches of Z and BNF but none elaborated on how such approaches were, or could be, applied to their particular environment. Others described an eclectic in-house approach where the problem

determined the approach to its solution. One company had adapted an off the shelf method (Yourdon's object oriented approach) for its own purposes by concentrating on the essential stages of the development process [SE2-9]. In other cases the requirements of a particular organisational wide quality programme or of the client (specifying that a particular design method be used) determined the approach. Also unsurprising was the finding that on the whole, software engineers had made more efforts to apply methods or were more desirous of such methods being applied. Graphic designers, with a few notable exceptions, were wary of methods seeing them as primarily constraints.

5.6 An inductive model of software design

It is now possible to summarise the previous discussion in a model of software design. This summary also draws upon data first presented in Chapter Two (2.6) and relates to research questions one and two. Figure 7 presents the summary.

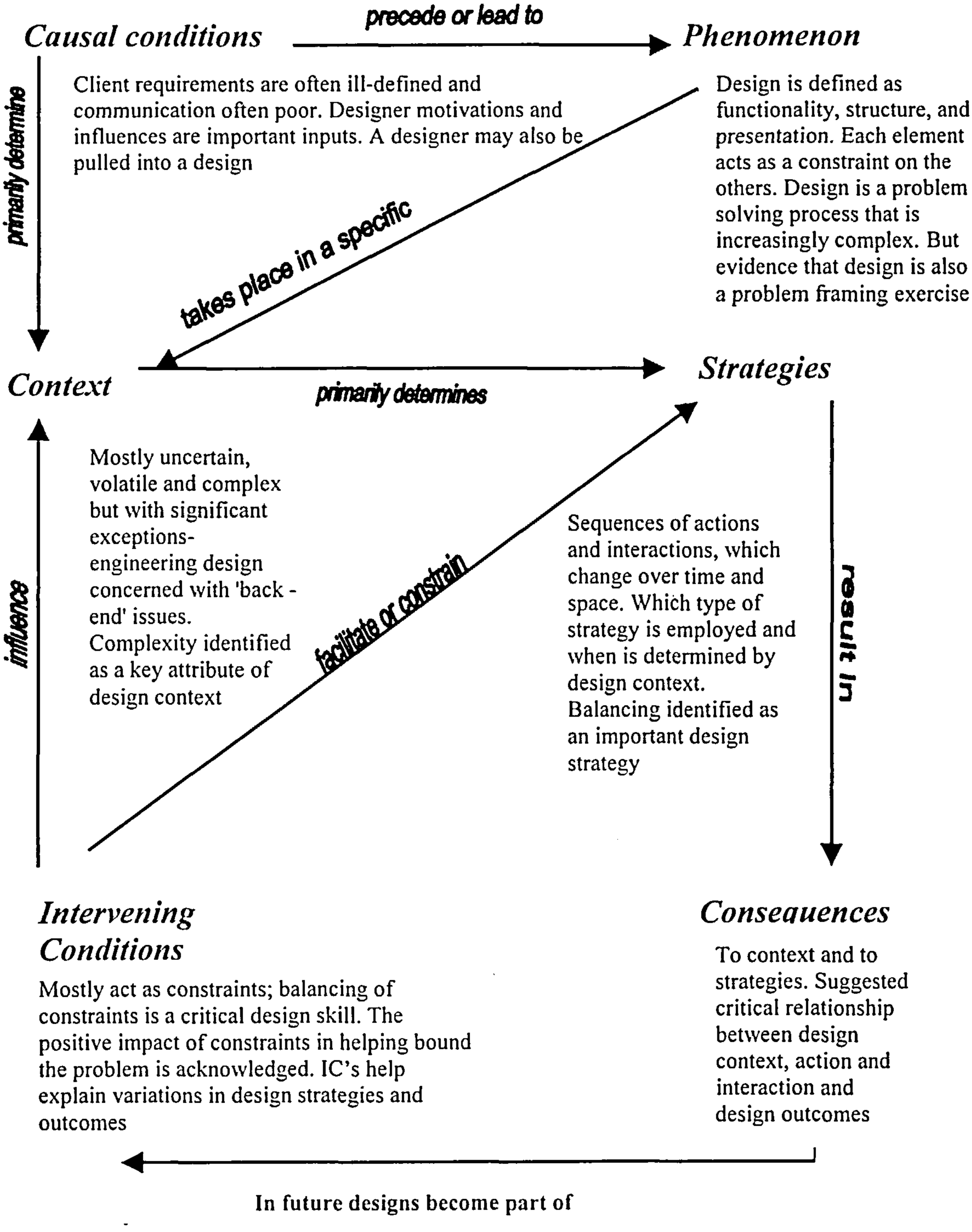


Figure 7: An inductive model of Software Design

5.6.1 The phenomenon

Software design consists of three essential and interrelated elements - functionality, structure and presentation. Broadly speaking, functionality is the 'what', structure is the 'how' and presentation is the 'look and feel'. All three elements define design and all three are present in every design. Emphasis on a particular element will vary within and between disciplines but over emphasis on one element to the exclusion of the others will result in a poor design. Each element is dependent, to a greater or lesser extent, on the others. So for example, form follows function, and presentation is determined by structure and, in turn, by functionality. The existence of these elements and their relationships, act as a constraint upon design and the designer must be able to manage this. There is some evidence that there is an increasing awareness of the importance of all three elements and of the need to balance these (and other) constraints. Finally, it is suggested, that although the relative importance of each element and the relationships between them will vary within and between disciplines, differences are becoming less pronounced due to the "entropic" effect of designing in a shared environment[24].

Each of these elements – Function [F], Structure [S], and Presentation [P] is a category, defined by its properties and their dimensional values. Each discipline placed different emphasis on each of these categories. Thus the property *Emphasis in design* with an ordinal value of *High or Low,* for each occurrence in each category,

---

[24] This metaphor works on two levels. Firstly, as a general observation (originating in experiments with mechanical and thermal agencies in thermodynamics) of the process whereby one phenomenon takes on the characteristics of another as the result of interaction with it. Secondly, in the application of this observation to information theory and Shannon's (1959) concept of mutual information - a measure of the information contained in one process about another process

across all possible relationships between the categories ~ (F <-> S; F <-> P; S <->P) produces eight theoretical design scenarios,[25] as follows

Function	H	H	H	L	L	H	H	L
Structure	H	H	L	H	L	L	L	L
Presentation	H	L	H	H	H	L	H	L

Figure 8: Some attributes of software design

Where Function [is H] and or Structure [is H] and Presentation [is L] the design may meet functional and technical requirements, it may be efficient and robust but may lack the usability, interactivity, and engagement vital to a Multimedia design. On the other hand where Presentation [is H], and Function [is L] and / or Structure [is L] the design may be superficial, possessed of a good user interface but lacking the necessary functionality and structure to provide a satisfactory user experience. Examples of such scenarios were found in the data in opinions expressed by both disciplines, reported in Chapter One and elsewhere dichotomised as" inside out" versus "outside in design" (Gallagher and Webb, 1997). Examples may also be found in specific domains or genres, such as in Multimedia games design (Mallon and Webb, 2000).

Each element is a constraint upon the others and the designer must balance these constraints. But what may such compromise look like and how may it be achieved? Again evidence can be found in the data. A "good" design will balance functionality / structure with presentation. The nature of this balance is determined by the nature of the application and by the designer or designers involved. Equal attention paid to all

---

[25] (1(2)x1(2)x1(2)=8).

three elements is unlikely and may even be undesirable in certain circumstances. An individual designer may struggle with such compromise due to the operation of a host of intervening conditions including his or her education, training, job experience, personality and motivation. In design teams responsibility for each element will be divided amongst individuals and groups. It then becomes the task of the chief designer or project leader to manage such compromises. These processes are directly linked (as one form of interaction) to the design strategies explored earlier in this chapter.

Software design was also described as a "problem solving" process. The problem solving process is not easy. There is never one correct answer, rarely a quick fix and seldom a clear, unambiguous path to follow. Sometimes the designer does not know that the problem has been solved, or how it was solved, or even how a particular point in the problem resolution process has been reached. The problem definition is seldom fixed but subject to frequent change and much of what the designer does in solving the problem can be interpreted as a reaction to this. A solution evolves through a problem solving process characterised by continuous refinement, incremental improvement and trial and error. It is difficult to delineate design's scope and conflicting views were presented on this.

There is some evidence that an already complex process is becoming more so. This relates primarily to the increasing functionality and richness of the interface making abstraction and separation more difficult. Evidence also can be found in the broader problems of defining and fixing user requirements, delivering a design that satisfies users on time and within budget and the general management of the process within resource constraints. Yet, software design is also (at least in part) a problem *framing* process whereby the designer manipulates constraints (including the essential

elements of function, structure and presentation) to frame or bound the problem. An apparent anomaly between definitions of design as a problem solving process and descriptions of that process suggesting problem framing is noted (and further discussed in Chapter Eight, 8.2).

5.6.2 Causal conditions

Design begins with the problem. The problem forms the basis of design and it is the task of the designer to identify the problem, to understand it and to solve it. Users play an important part in this process and in a user centred or participatory design approach users too may be problem-solvers. However the focus of this study was on what (professional) designers do. Here, the design problem (expressed as what the design must do) is distinguished from other, attendant, problems of the design process such as technical, human and managerial problems.

The problem originates with the user or customer or client. It was observed that these terms were used without definition but that based on this and previous research the terms client or customer were more appropriate. Unfortunately clients often don't know what they want, even when they think they do they don't, or even when they do know it may not be what they need, or they can't communicate their requirements, or change their minds. The result is a problem that is ill defined, volatile, "fuzzy", "woolly" and "complex".

A designer is motivated to undertake a design by intrinsic and extrinsic factors. Intrinsic factors are generally positive and include such things as a personal interest in the problem area, the challenge of solving a complex problem, the desire to satisfy users, the opportunity to be innovative. A designer may engage in design not through

conscious action but through unconscious or semiconscious action based on some integral element of the design process itself. This was observed and noted as the pull of complexity, the pull of technology and the pull of aesthetics. Extrinsic factors tend to be less positive and include the need for work (either this design or a future design that depends on its successful completion) a follow on from previous work, re-design or maintenance.

The designer brings to the design his or her own prejudices and experiences. These too will influence the problem and the means of its resolution. Professional experience, education and training can act as facilitators or constraints in the development of a design solution (and thus are also intervening conditions) but they also influence the designer's perception of the problem and his or her method of solving it. As such they are important causal conditions that lead to "the occurrence or development of the problem". It is clear that designers do not 'start with nothing' as one interviewee put it.

5.6.3 Context

It is the properties (and their dimensions) of causal conditions that are most important in determining design context. Client requirements are often ill defined and subject to frequent change, designers bring to the design a range of personal and professional motivations and prejudices, communication between designers and clients and between designers and designers can be problematic. This creates a design context that is uncertain, volatile and complex. It is difficult to scope a design, there is little agreement as to when a design begins and when it ends. The outcome of the design process is controversial and evaluations of design remain highly subjective. Technical

complexity is increasing due, in part, to the increased functionality (richness and interaction) at the interface.

However a significant exception to this design context was observed. There are occasions when the design context is well defined, stable and the technology is understood or at least under control. User requirements are fixed, or at least fixed earlier, changes are less frequent, and communication is better. This context is most reflective of engineering type approaches to software design, for example 'back -end' server design. The "contra" context is accounted for by a different set of values attached to the (same) properties pertaining to the (identified) causal conditions. This dichotomy of contexts (and design approaches) suggested that complexity was a key attribute in determining contexts, strategies and outcomes. This is further explored in the next chapter.

5.6.4 Strategies

Strategies were classified as either action strategies or interaction strategies. Action strategies are individual strategies taken as a response to managing design and are manifested as a series of action processes. Interaction strategies are defined as "people doing things together or with respect to one another in regards to a phenomenon and the action, talk and thought processes that accompany the doing of those things". Interaction strategies in the context of software design include communication, negotiation, compromise and collaboration. Also noted were the sequences of actions, of interactions, and of actions and interactions and the impact of context in determining which type of strategy was employed and when. This is further explored in the next chapter. Finally, balancing was identified as a powerful (and explanatory) design strategy.

5.6.5 Intervening conditions

Intervening conditions act to facilitate or constrain design strategies taken in a specific context. They come between a strategy and its consequences and between context and strategy. Intervening conditions can explain why faced with a given context two designers take very different strategies or why a given strategy is successful or unsuccessful. For example a designer may be encouraged or dissuaded by factors operating at the individual level (personality, education, experience, motivation, influences) and at the organisational level (methods, management style, culture etc). Only a few (the most salient) of the many intervening conditions that act and interact upon design were discussed in this study. Intervening conditions were identified primarily as constraints (technical, financial, temporal, user, individual (including personality, education, training, experience) and organisational (including in-house culture and methodology)) but also as facilitators. In fact, constraints can also facilitate design by bounding the problem. Nevertheless design has been defined as constraint driven and balancing constraints is a key design skill.

5.6.6 Consequences

A number of cause and effect relationships have been identified directly in the transcripts. Many however are difficult to unravel, not least due to the operation of intervening conditions. In the next chapter the relationship between context, strategy and outcomes is examined under specific scenarios and a further analytical tool is used to trace conditional paths within these scenarios.

5.7 Discussion

Strategies (action and interaction) produce certain outcomes (or consequences). However the relationship is not simply linear but is interrupted by a set of conditions (causal, contextual and intervening) that uniquely shape both the strategy and its outcomes. In turn, strategies and their outcomes become part of the set of conditions that interrupt future strategies. Thus software design may be seen as a process of continuous adaptation to the environment. Designs are adapted to the context within which they are enacted (they are largely constraint driven) and are, by varying degrees, adaptations of previous designs

Design begins with a problem and the problem originates with the user but the design comes from the designer. The designer generates a design from previous experience and adapts that design in line with the constraints of the existing problem environment. These constraints originate in the problem 'space' but are also imposed by the designer (or others) to frame or bound a problem. Whilst the detail of the cognitive problem solving process was not obvious from taped interviews it is clear that designers evaluate constraints along with design alternatives and that this activity itself is subject to constraints such as missing or erroneous information about the nature of a constraint. In addition constraints, like design goals, and user requirements, are often open, ambiguous and only partially satisfied. Thus the designer must balance or trade off constraints in order to achieve a satisfactory outcome.

Design adaptation of a previous design to meet a new context or of an existing design, on the fly, to meet changes in context, is regarded as a key design skill since knowing what to adapt and when is often a problem. The key appears to be to have a strategy

and method for adaptation that can be applied in different situations. "A successful designer is going to be one who has a long and creative life who can diversify" [GD4-1]. Overall, experienced designers do this better than less experienced ones but the dangers of designers becoming too reliant on existing methods and strategies was recognised. Finally adapting a design has consequences, direct and indirect, intended and unintended and these too must be factored into decision making process.

Software design may be seen as a process of refinement in which detail is added incrementally by generating alternatives and making commitments. However the process does not proceed in an even step manner but rather is quite erratic with frequent stops and starts, lurches forward and backwards, advances and retreats. This trial and error activity is in fact usually the result of a deliberate commitment strategy (although truly ad-hoc behaviour was suggested by a small minority of descriptions). Examples of early commit, late commit, top down and bottom up strategies were observed. Previous experience may encourage an early commitment or may foreshadow a more cautious approach - by the same designer, within and across design projects. However it was also observed that individual designers had preferred approaches and that sometimes the approach was incompatible with the context. The next chapter explores the role of context in more depth, in particular through an examination of the property of complexity.

5.8 Chapter conclusion

In this chapter an attempt has been made to present the outcomes analyses of two phases of the coding procedure (open and axial) in a manner that was both transparent and critical. Inevitably a compromise has been necessary and here it has been between

description and analysis. In this chapter greater emphasis has been given to laying out the data as a foundation for further analysis in Chapters Six and Seven.

Chapter Six: Context in action – towards a theory of software design

"I don't rely on theory as such, instead I rely on what works" [software engineer]

6.1 Introduction

A paradigm model of software design has been presented. But does this model work? Can it identify and account for the critical categories and relationships that determine the nature and outcomes of any design effort? Can it be used to account for design strategies in specific contexts and to predict future strategies based upon afforded understandings of such contexts? (questions of internal validity, relevance or fit). The chapter is structured following the selective coding procedures discussed in Chapter Four (4.3.4). The story-line is set out, the core category is identified and further developed in terms of its properties or sub-categories and its key relationships. A theoretical framework is then presented and further developed using the Conditional Matrix. Finally an emergent theory is laid out as a series of propositions.

6.2 The Story-line

The analysis of Data-set A resulted in the following "story" structured around Research Questions One and Two first posed in Chapter One (1.2).

> A software design is more than the sum of its parts. **Functionality, Structure** and **Presentation** define a design but it is the *relationships* between these categories and the relative importance placed on each by the two disciplines that is of greater interest. There is some evidence that, as a result of working together in design teams, software engineers and graphic designers are re-evaluating the significance of each of these categories and the relationships between them. For example, through a process described as design **entropy,** each discipline is becoming more aware of the strengths and contributions of the other and each is learning skills normally associated with the other discipline. These categories also act as **constraints** on the design and need to be managed or **balanced.**
>
> Software Design was described as a **problem solving** process, one that is becoming increasingly **complex.** However an apparent anomaly exist here as many in vivo descriptions of the process of designing software suggest problem *framing* rather than problem solving. This anomaly is found also in discussions of the design process – as described it is often linear and progressive, as observed it is often non-linear and regressive. Discrepancies between what software designers say they do and what they actually have important implications for product and process quality. (This is discussed in Chapter Eight, 8.2.2)

When designing software designers employ a range of *strategies*. These strategies can be categorised as **action** strategies and / or **interaction** strategies. Again however it is not the individual strategies that are of most interest but the *relationships* – between strategies within a category and between strategies across categories. Thus, software design can be seen as a sequence of actions (eg. **abstraction →de-composition → refinement**) of interactions (eg. **communication → collaboration**) and of actions and interactions (eg. **prototyping**). Such relationships are not static but change within and between design projects and over time, significantly adding to the **complexity** of the *phenomenon*. Action and interaction must be balanced and **balancing** is itself an important design strategy. What works is likely to be repeated and what doesn't will be discarded, or at least revised. The action strategy **re-use** was given as an example of this.

To explain why a particular strategy is adopted by an individual designer or why once adopted it succeeds or fails, is discarded or re-used, we must understand the influence of *conditions* attendant upon any design effort. In fact it is possible to characterise software design entirely as a logical and practical response to such conditions. So, uncertain and volatile **user requirements** lead to a **complex** design context, within which a high level of **interaction** occurs (identified as the predominant design context). On the other hand, clear and stable **user requirements** lead to a less complex design context within which a high level of **action** (or a low, or lower level of interaction) occurs (identified as the secondary or contra context). However these relationships are not linear, nor simply causal, because of the impact of *intervening conditions*. In these design scenarios, the **motivation** and **personality** of designer and a host of other factors operating at the individual and organisational level (including **influences** and **methods**) interfere to facilitate or **constrain** the strategies and their outcomes. They also influence context and, in turn, future design efforts.

Yet it would be wrong to view software design simply as a logical and rational response to the phenomenon (or more precisely to the prevailing context of that phenomenon). Sometimes software designers act irrationally, *or at least appear to do so* because their actions are contrary to what one may expect in the circumstances. Such activity was identified in incidents coded as the **pulls** of **complexity, technology** and **aesthetics** and may also be viewed where response does not match design context. However researchers need to be careful in the interpretation of such behaviour (Lee, 1999).

6.3 Choosing and defining the core category

At this juncture of the analysis, Strauss and Corbin (1990:121) recommend the identification of a core category. The "storyline" set out above has identified the major categories plus significant sub-categories. Any one of these major categories could be developed as a core category. Software design could be described as a **constraint-driven** process wherein strategies are devised and executed in a **constraint-bounded** context. **Balancing** is a significant design strategy because designers must balance **functionality, structure and presentation** and other **constraints**.

A strong case can be made for **complexity** as the core category. **User needs** are complex (uncertain and volatile), **designer needs** also, including design-intrinsic **pulls** – and complexity itself is a pull. Each set of needs contribute to context complexity, and in combination increase context complexity again. The design context is complex and strategies can be said to be managing or **balancing** this complexity. Finally there is some evidence that complexity is increasing and that it remains a major problem facing software designers.

Yet none of these categories –alone - is sufficient to describe and explain what is going on when software designers design, or at least to account for the data on this gathered and analysed in this research study. **Balancing** is powerful concept in explaining the process of design but itself tells only half the story for it is insufficient to address the response to the phenomenon one must also address the phenomenon itself, and the relationship between the two. Similarly, **complexity**, though highly descriptive and explanatory of the phenomenon studied (and capable of subsuming the other major categories as sub-categories), does not tell the whole story. Rather it is likely that only some combination of two or more categories will be sufficient. This is a problem for the researcher as Strauss and Corbin (1990:121) caution that it is unwise to seek to develop more than one core category in a single study.

The solution lay in the data. The storyline outlined above has pointed to the importance of relationships between categories. Again any one of a number of relationships presented themselves as worthy of further analysis (**user/designer needs – context; context – constraints; constraints – balancing**). However one key relationship stood out **context-complexity – action/ interaction.** This relationship overcomes a major shortcoming of having only one thematic category (even

complexity itself) as the core category. With the **context complexity – action / interaction** relationship the phenomenon and response to the phenomenon is considered, the context which (in part) determines design strategies (and within which such strategies take place) and the strategies themselves. This key relationship can be illustrated using Strauss and Corbin's analogy of pain relief

> Pain varies in intensity. When pain is at its most intense some pain relief is needed. However in order for the pain relief to be most effective at the point of greatest intensity it needs to be administered some time before the pain reaches its most intense. If this is done, then there will be some relief from pain. If it is not then there will be none. There are a number of reasons why pain relief may not occur. The patient may not be aware of his condition. The patient may be aware of his condition but fail to take action in time. The patient may take an inappropriate action or choose not to take any action. There are consequences for the patient in each of these strategies. By not taking pain relief or not taking it in time (therefore failing to control the pain) the patient may experience much greater pain (intensity) the next time pain occurs. (Strauss and Corbin: 1990:110)

The causal transactions identified here are applied to the activity of software design as follows

1. Context-complexity varies in intensity.
2. Response to context-complexity must be timely to be effective.
3. Ineffective response may be due to (a) lack of awareness of the phenomenon (b) failure to take make a response on time (c) making the wrong response (d) making no response.
4. There are consequences to each of these responses, including impact upon future responses.
5. To these we may add, responses may be explained by a set of conditions that operate upon the phenomenon and the response to the phenomenon.

Thus the key relationship **context complexity –action /interaction** became the core category, or rather a constructed core category comprised of two related sub-categories. This approach is consistent with Strauss and Corbin's earlier

recommendation for singularity since the focus of analysis is the relationship between two existing categories rather than the categories themselves. Therefore only the relationship is further developed and the properties of each category as they may describe and explain that relationship, rather than both categories per se.

That said, before examining the relationship in greater detail, it is necessary to further examine each sub-category. In particular the essential differences between context complexity and software complexity must be set out and, although much has been said about action and interaction in Chapter Five, this sub-category too needs to be developed, specifically as a response to the phenomenon of context-complexity.

6.3.1 Context-complexity (the phenomenon)

> Complex {L. complexus, p.p. of complectere, COMPLECT}, a. Composed of several parts; composite; complicated. n. A complicated whole; a collection; a complicated system.... Complexity, complexus {COMPLEX} (Oxford English Dictionary)

The word complex refers to an entity or phenomenon. This entity or phenomenon may be a product - a physical artifact such as an aeroplane (which is composed of many thousands of parts), it may be a problem ("a question proposed for solution; a question involving doubt or difficulty; a matter difficult to understand," wherein a solution is "the resolution or act or process of solving a problem"), or it may be a process ("a course or method of proceeding or doing, a natural series of continuous actions, changes etc").

By this definition, an entity that is complex is not necessarily difficult, at least not cognitively difficult. An artefact may be both complex (composite, intricate, consisting of many parts) and simple (not hard to understand). Model ship or aircraft building is one (trivial) example. Yet, it is clear that when designers talk about

complexity they mean (cognitive) difficulty. One example is the common juxtaposition of complexity and simplicity as design goals. Thus a good design is "functional" yet "efficient", "complex" yet "elegant", "complex on the inside" yet "simple on the outside".

Brooks (1986) sets out a causal relationship between complexity and difficulty (and its consequences). Software complexity leads to difficulty of communication among team members which results in product flaws, cost overruns and schedule delays. It leads to the difficulty in enumerating, much less understanding, all the possible states of the program and this results in program unreliability. Software complexity leads to difficulty of invoking function which makes the program hard to use; difficulty of extending programs to new functions without creating side effects and to unvisualised states that constitute security trap doors. He goes on

> "Not only technical problems, but management problems as well come from complexity. It makes overview hard, thus impeding conceptual integrity. It makes it hard to find and control all those loose ends. It creates the tremendous learning and understanding burden that makes personnel turnover a disaster". (Brooks, 1986:1070)

In this thesis it is argued that software design is complex because (a) the software product is composed of many parts (b) the problems or questions posed in the design of software are difficult and (c) the process of solving these problems involves continuous change. It is not necessary for all of these conditions to be true in any situation. Any one renders software design complex. However, the combination of conditions tends to increase complexity. Each of these conditions will now be considered.

(a) The software product is composed of many parts.

Brooks (1986) argues that complexity is inherent in the nature of software because

> "The essence of a software entity is a construct of interlocking concepts [data sets, relationships among data items, algorithms and invocations of functions]. The essence is abstract in that such a conceptual construct is the same under many different representations. It is nevertheless highly precise and richly detailed." (1986:1069)

The number of concepts or components in a software entity is caused by the fact that, above the statement level, "no two parts are alike" (otherwise similar parts are incorporated into a sub routine). Moreover, as the size of the software increases so too does its complexity, but non linearly. Elsewhere, Belady observes that software complexity is caused by "a staggering number of components" (Randall, Ringland and Wulf, 1994:419).

(b) The problems or questions posed in the design of software are difficult

Brooks was concerned with the early stages of software design

> "The part of software building I called the essence is the mental crafting of the conceptual construct; the part I called accident is the implementation process ". (1995:209)

He is in no doubt which transformation is the most difficult

> "I believe the hard part of building software to be the specification, design and testing of this conceptual construct, not the labour of representing it and testing the fidelity of the representation ..The hardest part of software is arriving at a complete and consistent specification, and much of the essence of building a program is in fact the debugging of a specification". (1986:1069)

Blum (1992) suggests why this is so. Abstraction is a difficult mental process due the limitations of the human information processor. There are few proven tools and techniques to support abstraction (ie to aid the formulation and communication of an informal model of a real world need). The majority of software engineering tools and techniques apply to the second phase of the transformation (the transformation of the specification into software code), where the underlying theory of computer science is much stronger.

(c) The software design process involves continuous change

Brooks observes that

> "The software product is embedded in a cultural matrix of applications, users, lans and machine vehicles. These all change continuously, and their changes inexorably force change upon the software product." (1986:1070)

Lehman's laws of programming state that (1) A program undergoes continual change, until it is replaced (2) An evolving program becomes increasingly more complex, unless action is taken (Lehman, 1980:1067-1068). Changeability according to Brooks is an inherent property of the software artefact, an "irreducible essence" along with lack of conformity, invisibility and complexity itself. All these essential properties contribute to software design's complexity. Lack of conformity introduces arbitrariness - "complexity forced without rhyme or reason by the many human institutions and systems to which interfaces must conform". Changeability encourages volatility - "software can be changed easily - it is pure thought stuff, infinitely malleable" and invisibility makes representation difficult - software " remains inherently unvisualisable" and does not "permit the mind to use some of its most powerful conceptual tools" (Brooks, 1986:1070).

In this thesis, it is argued that complexity in software design is caused by a specific set of conditions (contextual, causal and intervening) that pertain at a given point in time but which change over time. These conditions determine design strategies and their outcomes. To be sure, many of these conditions emanate from the nature of the software artefact itself but many do not. The role and impact of the designer's personality, education or professional experience in the design process for example is not primarily determined by the nature of the software artefact but by multiple influences operating at the individual and organisational levels. Thus the concept of context complexity is much broader than Brook's notion of software complexity. The impact of the problem in shaping the solution environment is recognised but so too

are other factors[26]. The important differences between the two perspectives are summarised in Table 11

	Software complexity (Brooks)	**Context-complexity**
Research design	Aristotlean notion of *essence* and *accidents* used to categorize major difficulties associated with software design. Concentration on four inherent properties that form the "irreducible essence of modern software systems" - complexity, changeability, invisibility and conformity.	Phenomenonological approach used to generate concepts and categories which are then ordered using Strauss and Corbin's (1990) framework. Context is identified as key variable determining strategies and consequences. Complexity is identified as a major category explaining the phenomenon.
Unit of analysis	Software programs, more specifically those essential elements of software listed above. The software design process is evaluated only to the extent that these elements affect it.	The software design process is evaluated as a transactional system comprised of a series of cause and effect relationships. Software programs are evaluated only as one element of this.
Unit of measurement	Not clear. Suggests it can be measured when he expresses the essence as a fraction of the software design task (1995:209-210) but does not indicate how individual elements may be measured. Refers to some general work on measuring the intellectual component of early design task.[27]	The dimensional values of the properties of the categories. Each property is located on a dimensional scale consisting of nominal or ordinal values. The dimensional value of the properties is then used to organise the data in terms of similarities and differences.
Outcomes (of observed phenomenon)	Technical and managerial problems caused by *software* complexity.	Technical and managerial problems caused by *context* complexity.

Table 11: A comparison of context-complexity and Brook's (1986) software complexity

Because of this distinction significant epistemological and methodological difficulties in identifying and measuring software complexity are avoided. Rather the definition of **context-complexity** is derived from the paradigm model and consistent with earlier

---

[26] In fairness, Brooks does address some contextual factors - most notably he later cites (albeit indirectly) Herzberg's work on motivation (1995: 210) but these are not the focus of his study.
[27] He cites Fjelstadt and Hamlen's (1979) study of software maintenance and Glass and Conger's (1992) study of requirements specification both indicating a *80/20 intellectual / clerical* split. Specific measures of software complexity are available (see Blum, 1986) but these apply to the later stages of the design process.

analyses using that model. This permits identification and measurement of the key variables using the dimensional values of the properties of each category. Thus

Phenomenon: context complexity

Property	Dimensional Range	Dimensional Value
Number of parts	many - few	many
Level of difficulty	difficult - not difficult	difficult
Frequency of change	frequent - infrequent	frequent

These properties (and others) of **context-complexity** determine the response and as the value of these properties change so too does the response.

The definition and development of the concept of context-complexity, and in particular the attempt to distinguish between it and software complexity, and including the discussion of a complexity threshold (6.7), should itself be placed in the broader context of the general literature on complexity and complexity theory. Software complexity is one aspect of computational complexity, which according to Hartmanis (1989:102) is "one of the central and most active research areas of computer science". The approach to context complexity in this thesis can be seen as one instance of the progressive development and application of computational complexity theory to other fields, including management (see for example, Elliot (1991) or Stacey (2000)).

6.3.2 Action and/ or Interaction (as response to the phenomenon)

Complexity may be best understood by reference to the phenomenon *and* the response taken to the phenomenon. Simon and Newell's (1972) laws of qualitative structure state that the structure of the task environment determines the possible structures of the problem space and the structure of the problem space determines the possible strategies for problem solving. Moreover Simon (1973) notes that problem spaces are

subject to constant change due to changes in the task environment and changes in the problem-solver's representation of the task environment. As problem spaces change so too do strategies. Therefore the design problem will (to a greater or lesser extent) determine the response.

But what determines the individual designer's response to context-complexity? Why do some designers make an appropriate response and others not? Why is the response sometimes too late? Why do some designers take no action? Why is **context-complexity** more of a problem in some projects than in others? Why do two designers take radically different approaches to the same problem? Why does the same strategy applied in the same or similar environment produce radically different results? The answers are again found in the context.

The nature of the problem will have a significant influence on the nature of the solution and of the solution process (Simon and Newell, 1972) but individual designers will view a problem through "their own set of ontological glasses" (Wernick and Winder, 1994), they each will have their own education and experiences to call upon. Each will be further facilitated or constrained by structural conditions operating at the sub-organisational or organisational level such as methodology, budget and time. All these factors, and others, determine a designer's response to **context-complexity**. Moreover an individual's response will change over time as structural or personal conditions change. It is observed for example that response is modified with experience. Whereas an inexperienced designer will rely more on intuition and cognitive ability, a mature designer, as cognitive abilities wane, will rely more upon knowledge of what works and what doesn't.

For Marr (1982) perception consists of vision (what we see) plus representation (how we describe formally, what we see). In this combined sense then, complexity is a unique creation of the designer - individuals will have different perceptions of complexity and no two designers will experience complexity in exactly the same way. A designer's experience of complexity is determined by his or her perception of the product, problem or process and will change over time. Brooks (1995:241) for example, observes that "both the actual need and the user's perception of that need will change as programs are built, tested and used", and Lehman (1980) also makes a distinction between actual and perceived complexity.

Where the designer is unaware that complexity exists, he cannot properly respond. Where the designer does not fully understand the complexity, he cannot properly respond. Where the designer perceives complexity where none exists, or perceives more complexity than actually exists, the response will be flawed[28]. Where the designer fails to identify or understand complexity –or overestimates his or her ability to cope with complexity the response will be flawed[29]. A designer may over define the approach at the beginning and fail to retain necessary flexibility as the design proceeds. Alternatively if a designer commits too early, he may be unable to deal with complexity as it unfurls. Sometimes the designer will be unaware of the complexity but do OK, intuitively.

---

[28] Pure phenomenonologists may argue that since everything is perception, there is no such thing as *actual* complexity. To avoid this epistemological conundrum, where a designer reported complexity it was recorded. Invariably, a distinction between perceived and actual complexity was made, implicitly or explicitly. This position is closer to Gordon (1989) the physical world is assumed to have an existence independent of perception.

[29] According to software engineers this was a problem for Graphic Designers, although this study found limited evidence to substantiate this allegation

The strategies used to manage complexity are –by and large - those used to design. That is, there appears to be no qualitative differences in responses to complexity and in general design strategies. Responses to complexity include **decomposition, refinement, iteration, trail and error, re-use**. Generic responses include **early commit, late commit, postponement and avoidance strategies**. A specific response is to **hide it**. Overall there appeared to be no silver bullet, rather a resolve to work harder and therefore a quantitative rather than a qualitative difference in approach. This analysis is corroborated by Data-set B (Chapter Seven).

Different responses will result in different consequences. For example we would expect the response to high intensity complexity to differ from that to complexity of lower intensity. Strategies for dealing with complexity may differ depending upon how long the complexity lasts (duration). The consequences of strategies will differ depending upon the degree of success of each strategy (full or partial) and the length of time (duration) the strategy works. Complexity also makes prediction of outcomes much more difficult. One consequence of strategies to manage complexity can be increased complexity (since the outcome of one strategy becomes part of the conditions impacting future strategies).

Intervening conditions come between a design strategy and its outcomes. They also influence the design context. This influence may be benign, the context is made easier (less complex) and design strategies are facilitated. An individual designer faced with vague, uncertain or volatile user requirements may employ education, training or experience (or a particular combination thereof) to minimise or reduce process change and problem difficulty. On the other hand, the influence may be malign, the context is made more complex and design strategies are constrained. Thus the designer who has

inadequate or inappropriate education/skills/experience may exacerbate context complexity by adding to problem difficulty or increasing process change.

For each sub-category – **context complexity** and **action/interaction** – it is possible to set out the properties and dimensional range of each, and to speculate on the interaction as phenomenon and response to phenomenon respectively. A selection of properties and dimensional values is set out in Figure 8 below

Context complexity (phenomenon)--action / interaction (response to the phenomenon)

Property	**Dimensional value (and of response)**
Amount or level	High – low (enough / not enough)
Intensity	High – low (high enough / not high enough)
Duration	Long – short (Long enough / not long enough)
Rate (of increase)	High – low (Fast enough / not fast enough)

With the possible values for each combination of relationships represented as

High / Low	**High / High**
Low / Low	**Low / High**

Figure 8: Some Properties and Values of the core category

Complexity always elicits a response. A decision to do nothing is still a response. The response may be intended (a conscious strategy to tackle the complexity) or unintended (such as when another strategy pursued for some other purpose indirectly addresses the complexity). Responses too can be measured in terms of their *amount, intensity, duration* and *rate* and variations in strategies may be explained by variations in the values of these attributes. The dimensional values of the properties of the response may not correspond to the dimensional values of the properties of the event

causing the response. For example, complexity that is intense may be met with a low intensity response, complexity of long duration may be met with a response of short duration (a possible quick fix).

This approach does not suggest that a given value of a property of one category must trigger a given value of a property in another category but rather that non correspondence can, and does, lead to design problems. Even then however, consistent with earlier analysis, we must be wary of value judging real or apparent mismatches. It may well be for example that a low intensity response to a high intensity phenomenon is entirely reasonable and appropriate to the circumstances (for example it may be resource constrained).

The software design context is not uniformly complex. Context-complexity is not experienced consistently within or across disciplines or environments. Rather context complexity will vary depending on the particular set of causal factors and intervening conditions in operation at a particular time. Therefore the study of context-complexity is the study of the properties and values of the phenomenon and of the response to the phenomenon, singularly, collectively and comparatively. In fact, most may be gained by concentrating on those occasions where there is a clear mismatch between the nature of complexity and the nature of the response (as measured in the dimensional values of the properties of each).

6.4 The theoretical framework

Each design strategy consists of action and interaction. In every strategy there is some element of interaction (at minimum, self-reflection) and some element of action (for example, talking or thinking). A strategy is therefore defined by a set of actions and

interactions directed at solving the design problem. But each strategy consists of action and interaction in unequal measure. Some strategies will be primarily action based whereas others will be primarily interaction based. For example, the action strategies of drawing, flowcharting and prototyping also serve important interaction (communication) functions. The relative importance of action and interaction in a given strategy is determined by context. In practice a designer will use both action and interaction strategies in each design effort - and move quickly and seamlessly between them as context changes. Therefore software design can be seen as a series of actions and interactions with frequent but subtle iterations between the two. Figure 9 identifies the four theoretical design scenarios created through the juxtaposition of **context-complexity** (level of complexity) and action/interaction (level of interaction).

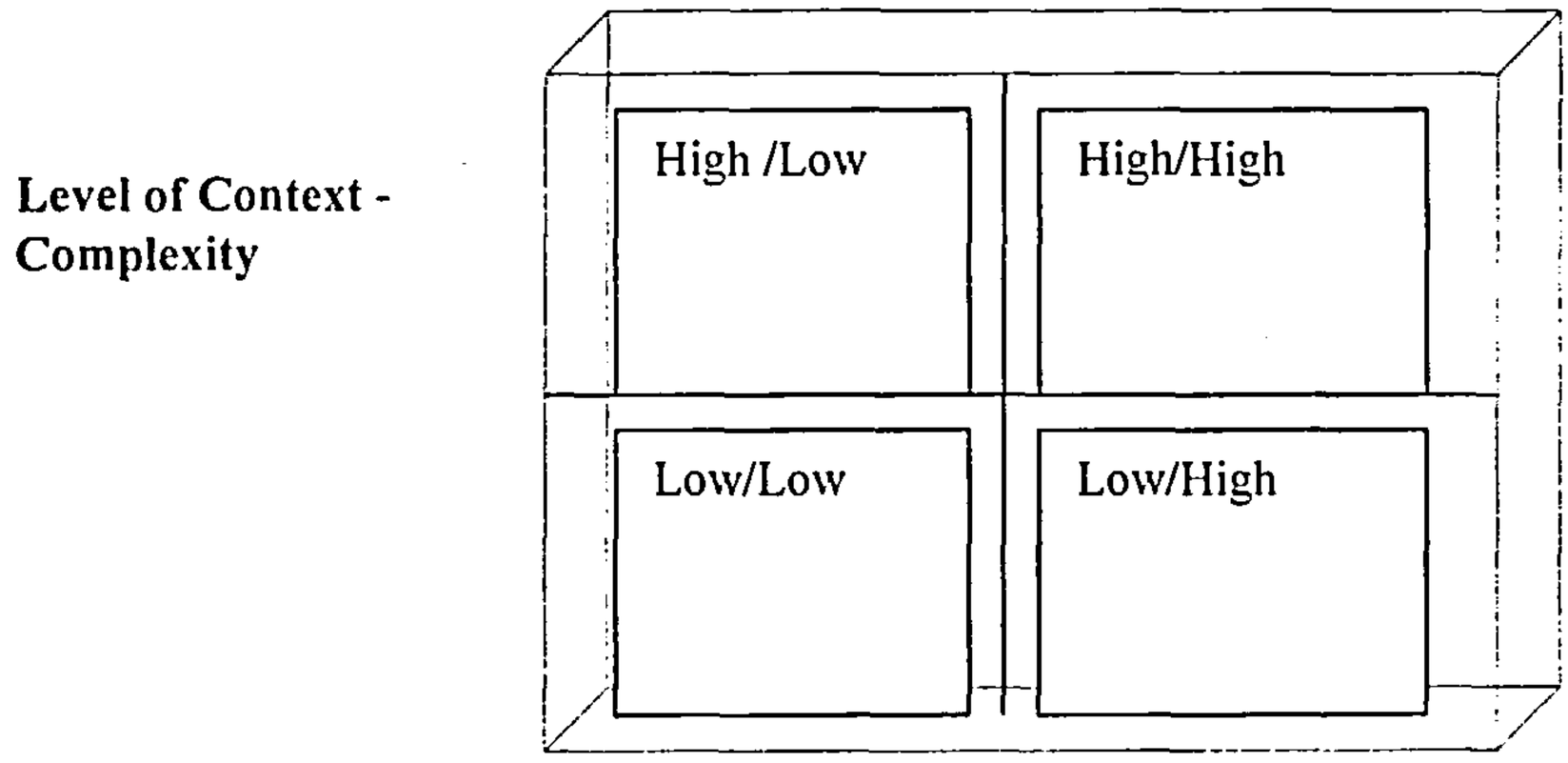


Figure 9: The Theoretical Framework

6.4.1 The low context-complexity –low interaction context

Where the context is less complex, strategies will require less interaction. One extreme example of this is in puzzle solving where the problem is clearly defined and the puzzle-solver can solve the puzzle with a minimum amount of interaction. Some

interaction in the form of self-reflection may occur but there is no recourse to interaction beyond the limited problem domain[30]. In fact in puzzle solving the problem provides considerable guidance as to the solution. Another example, from the field of Management, is the calculation of Economic Order Quantities (EOQs) in inventory control. Here, the problem is to balance inventory against cost using a mathematical formula in which all the variables are known and their values assigned or deduced.[31]

In software design such a context can be observed in traditional software engineering problems, normally associated with the 'back-end' of the application where problems of algorithms, coding and testing are usually well defined. In 'black box' design the designer seeks to specify a set of pre conditions, a set of post conditions and the transformations necessary to turn one into the other.

> "I have a black box model of design; if I have a requirement for a piece of design then I will put in the elements that I know will enable a designer to make the design. Then they do what they need to do to produce the design and they hand it back to me in a suitable format. That is all I need to know, and it is all that I want to know" [SE4-5]
>
> " If its the backend - the server constraint - I would hand you a set of preconditions and post-conditions; a set of white box tests and black box tests. That would be given to the coders and they would be told to go and code a solution in JAVA that meets this functional requirement that passes those sets of tests. I or someone else will produce that design and you or someone else will go off and deliver that design."[SE3-6]

At the front end of the application - the user interface- this context is much less likely due to the necessary involvement of users (and therefore reliance on extensive and intensive interaction). Converting a product catalogue to a Web site, where the structure and content of the printed catalogue is to be maintained is one example.

> "I mean a lot of designers are simply "OK lets turn the handle and we've got the design"....if you have got 800 pages of material you wanted to put on the Web – that's the spec" [SE2-4]

---

[30] Note that in this scenario the problem solver is deemed to be the sole problem solver and no interaction with other problem solvers takes place.

[31] The appropriate formula is SQRT of 2 ZC1 / cC where Z is total annual usage, C1 is cost of placing an order, c is unit cost of the item and C is carrying cost rate per year.

The first design context can be stated

Level of context complexity (low) --> level of interaction (low)

It is important to recognise however that even where the context is stable (less complex) iteration and refinement still goes on

> "sometimes the server design will be fixed beforehand, sometimes the front-end design will be fixed beforehand (but) usually there will be several iterations of refinement" [SE3-3]

6.4.2 The high context-complexity – high interaction context

Where the context is complex a high degree of interaction is necessary to frame or bound the problem. This interaction is with clients and other designers and involves **communication** and **collaboration** - the primary purpose of which is to clarify ill-defined and shifting requirements and to deliver a design that is sufficiently satisfying to the client. Interaction between designers is also important, for example, to define and agree tasks, negotiate roles and responsibilities or obtain resources. **Prototypes** are important to this interaction. Interaction with the materials of design also increases with increases in complexity as the designer tries to understand, and reflects upon, the design context. This context was the most frequently observed design context and can be stated as

Level of context complexity (high) --> Level of interaction (high)

6.4.3 The high context-complexity – low interaction context and the low context-complexity – high interaction context

The previous strategies are compatible with context. That is given a complex context we may expect to find strategies that are primarily based on interaction and where the context is less complex we may expect to find strategies that are primarily action

based. However, the juxtaposition of context and strategies throws up two other scenarios that are less expected and more difficult to account for. These are

Level of context-complexity (high) -- level of interaction (low)

Level of context-complexity (low) -- level of interaction (high)

Here the prevailing context is not met with an appropriate, or expected, response. Strategies that are predominantly action based are pursued in contexts that are complex. Strategies that are predominantly interaction based are pursued in contexts that are less complex. What causes these anomalies? The answer can be again found in the context, this time in the form of intervening conditions.

In both scenarios the designer is using an inappropriate strategy to deal with the design context. Possibly because they have failed sufficiently to understand the problem or having recognised it, are unable to implement an appropriate strategy problem - due perhaps to inadequate education, training and experience, possibly because the designer is hide bound by the in-house design culture, environment or methodology. Another possibility, based on observations made in Chapter Five, is that the designer is "locked in" to an approach or way of thinking about the design by the design itself – through the pulls of complexity, technology or aesthetics.

Users may force the response. Many designers praised prototypes and storyboards but others used them reluctantly or avoided them altogether. It may be that in a given design effort users' enthusiasm or client expectation for such devices forces a level of interaction inappropriate to the level of context-complexity. A high level of interaction is not always a good thing. Where it is inappropriate to the design context and peripheral or unnecessary to the design outcomes, it is wasteful of limited resources. Indeed it may be that too much interaction may result in an inferior design,

"It's bad communication if every time you do something more and show it to the client they tell you it is not really what they had in mind" [SE5-2].

One explanation may be found in process. The design context changes – becomes more or less complex – and the response has yet to catch up. Such delays can be fatal. If the response to a changing context is not made in good time – even to the extent of anticipating further increases or decreases in **context complexity** – the outcomes may be unsuccessful. Moreover this may impact future responses to continuing or intermittent changes in **context-complexity**. Of course, even where the response is timely and successful, context-complexity will be modified and future strategies changed accordingly.

Breakdowns in communication may be identified between designers and the materials of design, between designers and users and between designers and designers. Such events are also noted in the literature (see for example, Guindon, Krasner and Curtis, 1990). Another possibility is an exponential increase in context-complexity such that existing strategies do not work, and can't be calibrated to meet the new demands. Both these possibilities are further discussed later in the chapter. It is conditions that account for this mismatch between context and strategy and potentially any intervening condition or combination thereof may be culpable. The nature of these breakdown contexts is further explored, through the literature, in Chapter Eight (8.3.3).

Table 12 summarises the key categories and relationships for each scenario

Scenario/context	Context	Causes	Strategies	Consequences
1 low/low	Non complex	Stable and simple user requirements	Predominantly action based	"Fit" between context and strategy
2 high/high	complex	Uncertain, volatile and complex user requirements	Predominantly interaction based	"Fit" between context and strategy
3 high/low	Complex	As in Scenario 2	Predominantly (but erroneously) action based	"Lack of fit" between context and strategy
4 low/high	Non complex	As in Scenario 1	Predominantly (but erroneously) interaction based	"Lack of fit" between context and strategy

Table 12: A summary of the design contexts

In such analysis we are also concerned with global process or change. That is change within each scenarios is of interest – due to prevailing conditions - but also movement across scenarios. For example the process through which a Low/ Low scenario becomes a High / Low scenario, examining all conditions (causal, intervening and contextual) that occasion a change in state of the phenomenon and / in the response to the phenomenon. This may be particularly valuable in directing interventions for improvement. Finally it may be fruitful to examine values and relationships in combination since it may not be individual mismatches that are crucial but a pattern or profile of such mismatches over time and space.

6.5 Developing the theoretical framework using the Conditional Matrix

Strauss and Corbin (1990:158-159) describe the conditional matrix as "a framework that summarises and integrates" [the previous analysis], as an "explanatory framework" that represents "the highest level of analysis that is possible with the [grounded theory] method". It is appropriate then that this analytical tool – first introduced in Chapter Three and operationalized in Chapter Four – is used here to give depth and specifity to the theoretical framework. This discussion includes a

consideration of conditions at the outer levels of the matrix, and of process or change, and is developed primarily through use of a specific analytical technique – the tracing of conditional paths.

As well as its integrative and summarising roles, the conditional matrix performs two other important functions. Firstly it facilitates the consideration of a wider range of conditions, those at the outer levels or periphery of the main study that nevertheless impact the analysis and its conclusions. In this thesis the focus of inquiry has been at the inner levels of the matrix – on action and interaction – and conditions have been considered primarily as they impact directly upon these levels. That is, although some conditions at the individual and organisational levels have been discussed they have not been considered in relation to conditions found beyond the organisational level. This is important since as Strauss and Corbin (1990:161-162) point out "conditions at all levels have relevance to any study" and "regardless of the level within which a phenomenon is located, it will stand in conditional relationship to levels above and below it, as well as within the level itself".

Secondly, the conditional matrix facilitates the more explicit consideration of *process* or change or movement in the data. Again some such considerations have already been given - in discussions of action and interaction, and of conditions, and of the relationships within and between categories. This is a significant challenge for the researcher given the volatility of many conditions that attend a study of any size.

Figure 10, on the next page, sets out a Conditional Matrix for Software Design.

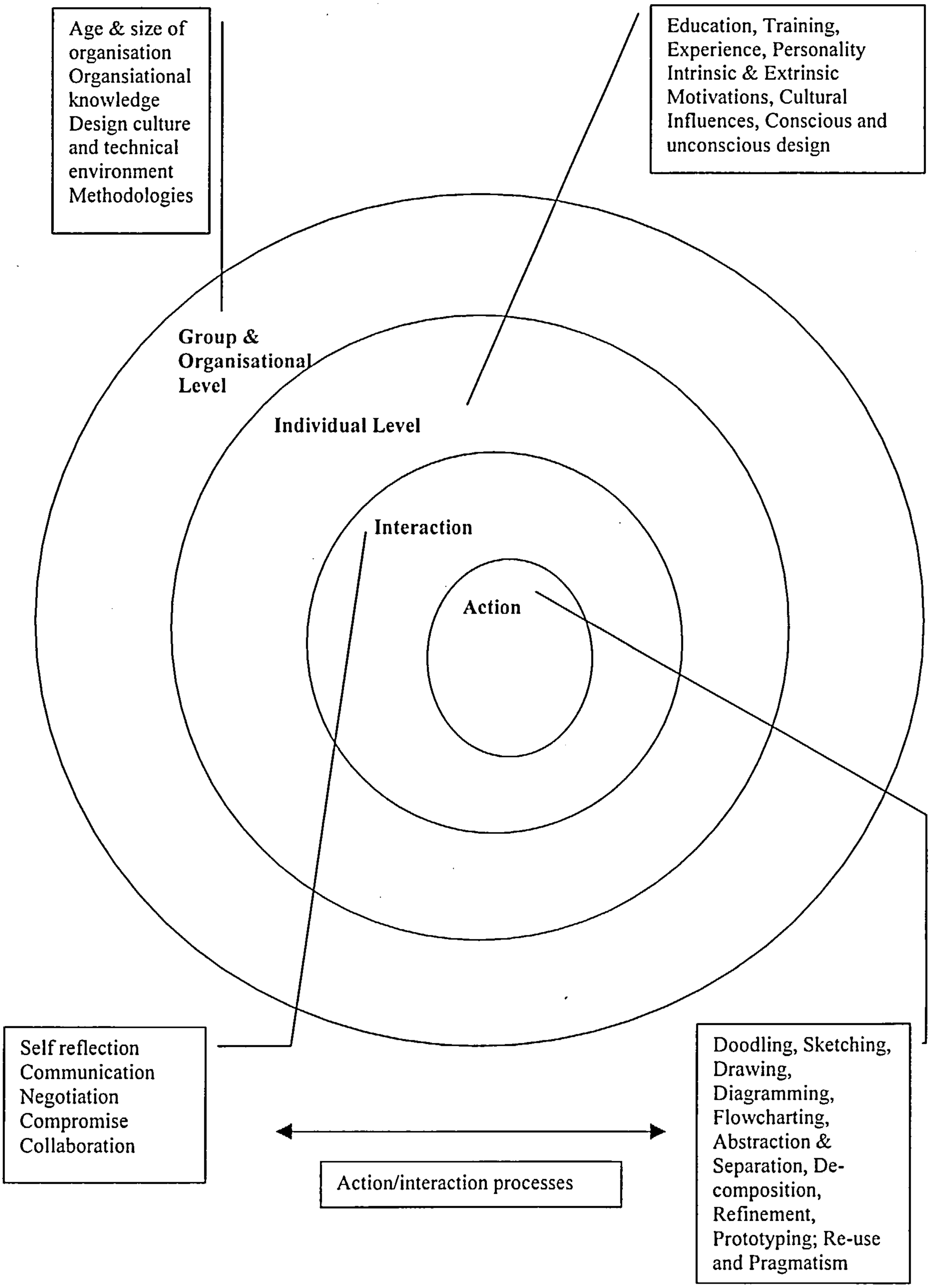


Figure 10: A Conditional Matrix for Software Design

As pointed out in Chapter Three (3.6), it is feasible to trace only a few conditional paths. These can be immediately limited to those within each of the four design scenarios. But there are multiple paths in each and some further qualification is necessary. Since any conditional path should be both relevant and interesting (Strauss and Corbin, 1990:167), it is necessary to examine the conditional paths in one scenario only – the **high complexity – low interaction** context (scenario 3). Here there is a clear mismatch between context and strategy or more precisely between the prevailing dimensional values of the properties of context complexity and the prevailing dimensional value of the properties of action and interaction in response to that complexity.[32]

For this scenario, the conditional path will be traced using a detailed description of a Multimedia development project given by two software engineers. This is presented first as a narrative then the project is analysed using the conditional matrix at the following levels– action/interaction; individual, sub-organisational, organisational, supra-organisational. The objective is to trace a conditional path from a specific incident or set of incidents, examining those conditions that shape it, and it them, directly linking conditions and consequences with action/interaction (Strauss and Corbin, 1990:166).

6.5.1 Mini Case study of a multimedia development project

This case study was constructed from three separate interviews with two key participants in the development project. One was the principal or lead software developer, the other the project manager. Unfortunately it was not possible to

---

[32] Scenario 4, the Low context complexity – high interaction context was weakly supported in the data (See Chapter Four, 4.3.3) and was not developed through the tracing of a conditional path.

interview any of the graphic designers involved in the project. However the project manager, himself from a software engineering background, purported to speak on behalf of all participants and in doing so frequently and strongly sets out what he believes to be this position.

6.5.1.1 Case study narrative

> Project X was a Multimedia Kiosk designed by a team of up to twenty, including content and audio-visual, a core development team of five designers, three of whom were software engineers. The graphics and programming elements were split between two organisations, with development carried out at separate sites. One of these organisations was also the customer. The project consisted of twelve man months of effort of which approximately fifty percent was programming and fifty percent was graphic design and content production. An in-house methodology was developed and applied during the project. The main development tools were C++ and Macromedia Director.
>
> The project was driven by software engineers in the customer organisation. The Project Manager (a software engineer) initially proposed that the work be divided along strictly functional lines, with one organisation (his) doing all the coding and the other organisation doing all the graphics. This was rejected by the graphic designers. What happened was that the software engineers in organisation X developed a prototype then passed it over to the graphic designers in organisation Y who developed the graphics and passed it back for further coding. So in practice there was some cross over in programming and graphics.
>
> The geographical separation within the design team inevitably brought problems in communication. Software engineers and graphic designers met "about once a week", for a "couple of hours" but
>
> > "Having the graphic designers at [Company Y] and the rest of us at [Company X] was a nightmare. Things were being passed back and forth and we didn't even have a network to aid communication. At least being in the same room would have allowed us to talk and argue about things there and then. As it was we had to postpone the argument or try to talk about it over the phone, which is very hard to do and so the phone was used very rarely" [software engineer]
> >
> > "the prototype would come back totally messed up after the graphic designer had done his bit. This then involved re-doing a lot of work" [project manager]
>
> Despite, but also because of, this communications barrier, a number of steps were taken to manage the interface between the two disciplines. After approximately six weeks "where we were running into consistent disagreements and difficulties" a modus operandi was established through a combination of formal and informal procedures, to "minimise" the communications issue. For example, following an initial brainstorming session a specification (prototype) was drawn up which also clarified the respective roles, this tolerated some changes to modules designed by software engineers but only within agreed limits. An object oriented type approach to design by the software engineers, encapsulated in references to "a grey-box approach" facilitated communication and control. Thus software engineers were able to develop prototypes at a higher level of abstraction " the interface and content [were] just grey boxes, rectangles with text on them to tell you what function they are to perform" and the graphic designers were able to "drop in the real graphics" later.
>
> There is plenty of evidence that such steps and others were successful. The collaboration produced an end product, on time that was generally well received. Evidence of successful

collaborations permeates the transcripts. Compromises were reached on the use of fonts (postscript and bit mapped rather than true type), on anti-aliasing text (on the main menu but not on subsequent menus), and on the use of thumbnail graphics (avoided with menu buttons because it restricted modifications). The project manager (also a software engineer) down played any problems – perhaps not surprisingly

> "Bob had some excellent ideas which were worked upon grudgingly by the graphics designers. Likewise the graphic designers had some excellent ideas which won the respect of the software engineers on the project" [project manager]

Yet there is also plenty of evidence of problems unresolved and of broader, underlying issues between the two disciplines. Different values and priorities held by each discipline were reflected in different objectives that continued to undermine unity of purpose. The software engineers wanted to build a robust system that was easily modified. The graphic designers were motivated by a desire to build an up market but essentially one off kiosk. Each had very different views on the relative contribution of each discipline (earlier dichotomised as graphics led versus functionality led, outside in versus inside out) and on quality "software engineering wants to define quality (metrics, definitions, quality assurance), whereas to a graphic designer quality is more intuitive""[software engineer].

6.6.1.2 Case Study Analysis

It is clear that communication (or the lack or it) was a major problem in this project.

How and why did this breakdown occur? The narrative highlights some major

concerns and suggests others, sometimes implicitly, that require further analysis. At

this juncture the Conditional Matrix is employed to develop the identified causal

relationships in terms of those conditions operating at progressively more distant

levels to the incident but which uniquely shape it.

Action/Interaction level

The failure to co-locate the design team is obviously a major issue and needs to be

explained. Whilst neither interviewee accounted for this decision directly we may

nevertheless assemble a plausible case from other data obtained across all three

transcripts. It is clear that both organisations were not equal partners in the venture.

Rather one organisation (Company X) was also the customer, commissioning and

developing the product but in effect sub-contracting out the graphics elements to the

other organisation (Company Y). Other parties to the design team were also

geographically dispersed – audio visual and content - but with much less impact on

process and product outcomes. None of the key developers were involved with the project full time and the project was quasi commercial in that it was not being developed for a third party customer and had no budget as such. The project did however have a tight time-scale (6 months) with "limited time to plan and organise in advance" [project manager]. It is also clear that key players, including the project manager had limited experience of such projects. These factors may explain the absence of co-location but was the absence of co-location the critical cause of the design breakdown?

Firstly it is instructive to examine again the interventions made to address this situation. Up until around six weeks into the project no effective communication between the organisations and between software engineers and graphic designers existed. There were "consistent disagreement and difficulties". Then the principal software engineer came up with "a set of guidelines by which the programmers and graphic designers could operate". This was "a real positive input" and "imposed a degree of logic". In practice the implementation of these guidelines established a communications "pipeline" between the two disciplines. This was subsequently exploited to deliver the "grey box" design approach that further regulated the roles and responsibilities of each discipline but particularly constrained the scope of graphic designers. These steps, as pointed out in the narrative, met with some success, a working relationship was established, compromises were reached and the product was produced on time. But intervening conditions can be seen to come between these intervention strategies and their consequences, sometimes facilitating, mostly subverting intended action and interaction. Since here we are interested in the cause or causes of the design breakdown, we will concentrate on those conditions that

constrained outcomes. These are discussed here at the individual, sub-organisational, organisational and supra-organisation levels of the Conditional Matrix.

Individual level

At this level we begin to examine the operation of some of the myriad influences that determine design context and therefore strategy. Earlier an overall lack of experience was commented upon. At the individual level this was significant. One programmer had never used a key development tool (Macromind Director) before, others had only been using it for about six months. The narrative points out that each discipline had its own objectives, beliefs and values and that these were often incompatible. These can be traced back to very different education and career experiences. The principal software developer identified this influence when he said

> "If you do engineering or programming then someone will teach you design and they will use the changing a wheel of a car, which isn't a very creative example to illustrate this. It's a very functional example. Whereas if someone goes to art college then their example will be the design of some sort of painting, which is a creative example but not very functional".

Therefore whilst the absence of co-location of the design team may have precipitated a design breakdown, and whilst this was reasonably managed when it did occur, it is clear that deeper, structural psychological and sociological influences were at work at the level of the individual designer, causing and perpetuating the communication problem between the disciplines and transcending intervention strategies.

Sub –organisational level (group)

Aside from the geographical dispersal of the design team, it lacked appropriate leadership, systems and methods. The project manager was a software engineer with no previous experience of Multimedia. "The fact that this was a new team that had never worked together before meant that we had a learning curve to cope with" and "the main tool that we lacked was the facility for networking between the two groups

because at any one time there could have been three versions of a section" [project manager]. Communication difficulties arising at the individual level, manifested themselves at this level, and in turn often reinforced individual prejudices. Whilst there is some evidence of mutual understanding and appreciation between the disciplines, the fact that the entire project design methodology was software engineering and functionality led relegated graphic design to a secondary role. The graphic designer did not for example, become involved until after the initial brainstorming session which determined the specification and thereafter were limited to building a front end to an engineering back end.

Organisational level

Discussion at this level is limited both by the lack of direct data and by the need to conceal the identity of the two organisations. Sufficient to point out however that the age and size of the organisations reflect an unequal alliance and that neither organisation had had significant experience of multimedia product development. Moreover the double role of organisation X as developer and user created a complex and uncertain design environment since it was never clear who the *end* users were and what they expected.

Supra-organisational

At this level we consider those external factors which are most remote to action and interaction but which nevertheless influence them through successive layers of the design matrix. These factors include for example the systems of education and professional training that are in place for each discipline and that crucially impact individual and collective effort in design. Whilst references to this level of influence were few, they do agree on the need for greater common purpose in this area – for

example on the need for a shared language, and a methodology that harnesses the best of both disciplines.

6.5.1.3 A re-evaluation of the theoretical framework

The analysis of this mini case project using the conditional matrix has also identified a number of issues that require a return to the theoretical framework. In particular the nature of the relationship between context-complexity and action and interaction must be re-examined

1. It is not absolute levels of **context-complexity** or interaction that are important but the level of context-complexity *relative* to the level of interaction. In this case study the project could not be described as significantly complex (it was at most a medium sized project, there was no third party user, the specification was determined early on by the software engineers and rigidly adhered to, with a few exceptions, throughout). However the level of interaction was low relative to the context-complexity (there was no co-location of the design team and even after steps were taken to facilitate communication the level of interaction remained poor). Thus the level of interaction was incommensurate with the level of context-complexity and design breakdowns occurred.
2. A consequence of taking steps to improve interaction that are successful is to reduce the level of context-complexity. The relationship is symmetrical. A change in state in one sub-category will lead to a change in state in the other sub-category. However in practice these changes are often asymmetrical, not immediate and may be hard to quantify. In the mini case project the introduction of an agreed specification and the establishment of a communication pipeline reduced **context-complexity** which in turn required a different set of management strategies (for

example to maintain exiting channels of communication rather than to develop new ones). This observation reflects the transactional nature of the phenomenon under study.

6.6 The emergent theory as a series of propositions

1. Complexity in software design can be defined and measured.
2. Complexity in software design can be measured as the dimensional values of the properties of the design context pertaining at any point in time (and these will change over time).
3. Software design is complex because the software product is composed of many parts, the problems or questions posed in the design of software are difficult and the process of solving these problems involves continuous change.
4. Complexity in software design is caused by the nature of software itself and by other contextual factors (analogous to but not synonymous with Brooks' (1986) essence and accidents).
5. In theory there is a symmetrical relationship between level of complexity and level of interaction - the greater the level of complexity, the greater the level of interaction.
6. In practice sometimes this relationship does not hold. As the level of complexity increases the level of interaction is static or declines (the balance of action and interaction present in any design strategy shifts towards action or interaction does not keep pace with rises in complexity).
7. The causes of such design 'breakdowns' are again found in the context, this time in the form of intervening conditions. Any intervening condition may be culpable and may be found at each level of the design matrix.

6.7 Discussion

But what causes increases in level of context-complexity? One hypothesis is that software designers experience a complexity barrier or threshold, sooner and more often than engineers in other engineering design disciplines. This causes software designers to extrapolate beyond their technical knowledge, to work beyond their level of expertise and results in poor quality systems - systems that are over budget / late / don't meet user requirements. (Randell, Ringland, and Wulf, 1994). Randell argues that in software engineering there is a cumulative increase in innovation and complexity across a series of successive projects to an extent not experienced in other engineering disciplines, although this is not the case in the packaged industry. He asks do more software projects fail because of this than other engineering applications? (Randell et al, 1994:420) Comparisons are most often made with civil engineering design and bridge building continues to be cited as both a metaphor and an exemplar of software design practice (see for example Van Vilet, 1993).

According to this view, level of context complexity increases with size of project in a linear direction until at some point (labelled the complexity barrier or threshold) increases in complexity become exponential to increases in size. At this point the level of integration (number of components), or the difficulty of problems experienced or the volatility of the environment (or some combination thereof) become such that existing design strategies prove inadequate or inappropriate. The quality of the design deteriorates, or the schedule slips, or the budget is exceeded, or all three. Moreover, the intensity of the complexity experienced and the rate of its increase make current explanations and future predictions more difficult.

Intervening conditions are also at work. These include education “needs to be better, longer, more settled"; process "need for people to work systematically design methodologies "poor"; analysis "analytical aspects of SE not well understood"; standards "lack of real standards even in well understood areas". There are also commercial / business pressures pushing designers into complexity "over ambitious goals" most notably feature and functionality battles between rival software producers / products (Randell el al, 1994:413-419)

Evidence of a complexity barrier or threshold can be found in the general literature.

Brooks (1986) refers explicitly to such an arrangement

> "a scaling up of a software entity is not merely a repetition of the same elements in a larger size; it is necessarily an increase in the number of different elements. In most cases, the elements interact with each other in some non-linear fashion, and the complexity of the whole increases much more than linearly." (1986:183)

He states that "many of the classical problems of developing software products derive from this essential complexity and its nonlinear increases with size" (1986:183). As noted earlier, these problems include technical problems (including what he refers to as conceptual integrity problems or problems with the coherence and consistency of the design itself and managerial or systems problems (which can indirectly impair conceptual complexity). Earlier, drawing upon the work of Lehman and Belady (1971) he alludes to a similar effect: although the total number of modules increases linearly with the release number of a large operating system (OS/360) the number of modules affected increases exponentially (Brooks, 1975:122).

Evidence for or against the existence of a **context-complexity** barrier increases our understanding of complexity in software design and of the role and impact of context in determining design strategies and consequences. It also informs interventions to

improve the software design process. If a complexity barrier is found to exist, if it can be defined and measured in terms of its properties and their dimensional values, then it is possible to observe these over a period of time and to better predict behaviour before, during and after the occurrence of the phenomenon. Alternatively if no evidence of a complexity barrier or threshold can be found and the relationship between level of context-complexity and level of interaction remains uniformly linear then, by definition, it should be easier to manage complexity and to predict its consequences.

By definition the complexity barrier or threshold occurs sooner and more often in larger projects. These are the design projects referred to by Randal, Belady and others where there is "integration of a staggering number of components", "over-ambition of application" and "extrapolation beyond the technical knowledge base" (Randell et al, 1994:419-421). In small and medium sized projects a complexity barrier may still be encountered but less frequently and with less severity. A lack of evidence of a complexity threshold in Data-set A is unsurprising. The age and size of organisations, the size of the projects, and the experience levels of individual designers suggests that such a barrier is unlikely to be encountered. Moreover whilst there is support in the literature for the concept of a complexity threshold this is invariably referring to software complexity which is here only one aspect of context complexity. Further evidence of a complexity threshold was found in Data-set B and this is reported in the next chapter.

6.8 Chapter conclusion

In this chapter the model of software design presented in Chapter Five has been further developed through the setting out of a story-line, the identification and development of a core category, the development of a theoretical framework and the specification of one aspect of this using the Conditional matrix. In response to the questions posed at the beginning of the Chapter it is contended that (a) a means to identify the occurrence of design breakdowns has been presented (the theoretical framework) and (b) a means to examine how and why such breakdowns occur (further analysis of design scenarios using the conditional matrix).

Chapter Seven: Further validation of the model and theory

"Learn from small experiments rather than large ones" [software engineer]

7.1 Introduction

In Chapter Four the case for internal and external validation of research findings was set out (4.4). Although the grounded theory method is inherently self-regulatory - through for example, the constant comparison of data and the grounding of hypotheses – further tests on the accuracy and reliability of the analysis are appropriate. This chapter introduces three such tests. The first test seeks to "ground" the categories generated from Data-set A in the technical literature using a secondary analysis of eighteen textbooks. The second test is a quantitative analysis of the same categories and is therefore one validation of the inductive model. The third test uses a second empirical data set – Data-set B, introduced in Chapter Four (4.2.3), to both validate the original categories and to extend the model and theory to another software design domain. It is therefore both a test of internal validity and of external validity or reliability. The tests employ a mixture of parametric and non-parametric statistical techniques alongside further qualitative analysis. Together they may be seen as further triangulation of the original data (Cresswell, 1994).

7.2 Test One: Using the technical literature

The choice of literature for analysis at this juncture in the research project was guided by theoretical sensitivity and sampling based on relevance and purpose. It was also influenced by the expedients of availability, access and cost. The author was already familiar with two theses in the areas of software engineering and graphic design. On re-reading these it became clear that a direct comparison could be made between the

outcomes of this research and the results reported in these theses and that such a comparison would be valuable.

The two sources selected for comparison jointly provide access to the analysis of eighteen textbooks pertaining to the field of software design. Wernick (1995) and Gallagher (1998) set out to identify paradigms or communities of software engineering (Wernick and Gallagher) and graphic design (Gallagher); Wernick in the field he termed Computer Based Systems Development (CBSD) and Gallagher in the field of Digital Interactive Multimedia (DIMM). Both employed Thomas Kuhn's philosophy of science as a method for identifying the paradigms (indeed Gallagher's work is very much an extension of Wernick's, which itself has parallels in other areas of information systems research, see for example van Gigsch and Pipino (1986), Farhoomand (1987) and Banville and Landry, (1989).

It is neither appropriate nor necessary to detail the research reported in these theses. (The interested reader is referred to the unpublished theses or to the published papers which emanated from the same, or cognate, research - Wernick and Winder, 1994; Winder and Wernick, 1994; Wernick and Winder, 1996; Gallagher and Webb, 1997; Gallagher and Webb, 2000). However, for the purposes of this chapter, it is necessary to provide some background to Wernick's and Gallagher's data.

Wernick (1995) selected eight software engineering textbooks (without stating the basis on which they were selected - though one may infer that he considered them to be representative of the field). These were

- Birrell and Ould: A Practical Handbook for Software Development (1985)
- Sommerville: Software Engineering (1992) (Fourth Edition)
- Downs et al. SSADM- Design and Context (1992)
- Licker: Fundamentals of Systems Analysis with Application Design (1987)
- Van Vilet: Software Engineering: Principles and Practice (1993)
- Jackson: System Development (1993)
- Schach: Software Engineering (1993)
- Carmichael: Object Development Methods (1994)

Gallagher selected four software engineering texts (since he claimed Wernick had already produced evidence that a community of software engineering did in fact exist) and eight graphic design texts. Two of the software engineering texts had also been selected by Wernick. Gallagher was more specific on the basis of his selection (Gallagher, 1999:99). His selected texts were

Schach (1993) 'Software Engineering', 2nd edition
Sommerville (1995) 'Software Engineering', 5th edition
Pressman (1994) 'Software Engineering: A Practitioner's Approach', 3rd edition
Budgen (1994) 'Software Design'
White (1988) 'Graphic Design for the Electronic Age'.
Muller-Brockmann (1964) 'The Graphic Artist and his Design Problems'.
Hamilton (1970) 'Graphic Design for the Computer Age'.
Labuz (1991) 'Contemporary Graphic Design'.
Cheatham *et al* (1983) 'Design Concepts and Applications'.
Marcus (1992) 'Graphic Design for Electronic Documents and User Interfaces'
Rand (1985) 'A Designer's Art'
Swann (1991) 'Graphic Design School'

In the interests of parsimony and efficiency it was decided to match the elements found in the literature sources to the categories of the inductive model rather than vice versa. (The alternative would have meant listing all elements of Wernick's and Gallagher's paradigms and then for each element determining the degree of support in the inductive model. An initial attempt at this showed that this approach would have resulted in a listing of many elements that were unsupported or only weakly supported). Besides, the objective of this chapter is to verify the inductive model not Wernick's and Gallagher's paradigms[33].

[33] Although the term 'paradigm' is shared between this study and Wernick's and Gallagher's research, it means very different things in each case. Wernick and Gallagher were concerned to identify Kuhnian paradigms or communities in the disciplines of software engineering and graphic design. They refer to this paradigm as a Disciplinary Matrix. This research seeks to produce a descriptive model of software design using a research tool developed by Strauss and Corbin that they so happened to call a paradigm model.

Each output table listing the elements of the respective Disciplinary Matrices (or paradigms) was read through and tagged with the relevant category code (or codes). A statement may be tagged with more than one category code as it may support more than one element of the inductive model. Where a statement was found that directly refuted a category of the inductive model (or a sub-category or property of this) this was included and indicated by placing a negative sign (-) in front of the refuted category.

In some cases the match is obvious. The statement is at a sufficiently high level to indicate an immediate match with one category of the model. In other cases the match is less obvious. The statement is at a lower, more detailed level, and the match has been made on the basis of a correspondence to some sub-category or property of the category indicated. This is not evident from the table but may be traced through the category and concepts (which are summarised in the appendices).

The "weight" attached to each statement refers to the number of texts that included support for this statement. It is therefore a measure of the support for the statement in the original sources and not a measure of the level of support for the matching of the statement to a category of the inductive model. It is included here merely to give some indication of the importance of the original statement and, by inference, the significance of correspondence in the model.

Category Code(s)	Supporting DM Statement	Weight (Max 4)
Structure; Functionality	Design describes how a product is to do what it is supposed to do	2.0
Approach; Method	The software design process requires thorough planning	2.0
User-requirements	Software design serves to satisfy user's needs	3.0
Problem Solving/Framing	Design involves more than the application of technical skills- sound management is also desirable	4.0
Iteration; Refinement	Design is an iterative process which adds greater formality and detail as the design develops	4.0
Notations; Communication	Notations/Models (flowcharts) aid the designer in visualising, organising and communicating design concepts	3.0
Communication; Collaboration	A compromise must be made between conflicting priorities during system design- trade-offs are inevitable	2.0
Abstraction; Good Design	Abstraction is an essential feature of good software engineering design	4.0
Refinement; Iteration	Testing is an integral aspect of design and as such it should be performed continually	2.0
Decomposition; Abstraction	Design usually progresses from higher levels (architectural design) towards lower levels (detailed design)	4.0
Decomposition	Decomposition and stepwise refinement are valuable design techniques	4.0
Re-use; Influences	Domain knowledge and/or prior experience enhances a designer's ability to design a solution	4.0
Context-complexity	Choice of design method /strategy depends on the nature of the product application as well as the designer	4.0
Problem Solving/ Framing	Design is a creative problem-solving activity	2.0
Good Design / Bad Design	There is no generally accepted notion of what exactly constitutes good software design	2.0
Refinement; Complexity	By its very nature the design process is difficult to formalise and refine	2.0
Designer Influences	Design depends on the knowledge, intuition and skill of the designer	3.0
Abstraction; Separation	It is generally good design practice to separate (as far as is possible) the user interface from the data processing functions	2.0
Iteration; Users; Prototypes	User interface design should be an iterative user centred process involving users and prototyping	2.0
Designer-influences; Motivations	Design is open to interpretation with different designers developing different solutions to the same problem	2.0
Design-Constraints	Awareness of practical constraints	3.0
Prototyping	Prototyping is a valuable design tool that permits the designer to evaluate proposed solutions	3.0
Complexity	Modularity reduces complexity	3.0
Complexity	Designers should try and hide as much info as possible	3.0
Motivation; Pulls	Elegance of code	3.0
Re-use	Re-use of code and design plans	4.0

Table 13: Evidence found in Software Engineering – textbook trawls (Gallagher, 1999)

Category Code(s)	DM Statement	Weight (Max 8)
Communication; Collaboration	Compromise (with other designers, managers and clients) is inevitable	1.0
Re-use	No hard and fast rules just the advise of experience	5.0
User-requirements	Adopt the users perspective when designing	3.0
Influences	Influence of a designers personality affects the outcome of design	3.0
Influences	Influences of fashion	3.0
Influences; Motivation; Constraints	Design is a question of interpretation based on preference, understanding, objectives and materials	2.0
Problem Solving/Framing	Design is a creative problem solving activity	5.0
Presentation; Functionality	Graphic design is not merely a matter of aesthetics - designers must be aware of the practicality and functionality of the proposed design	5.0
Decomposition; Complexity	Decomposition provides a useful mechanism by which designers can manage large complex problems	6.0
Functionality	Fitness for purpose	2.0
Functionality	Everything which is designed must be justified or have a reason for being	3.0
Functionality	Designers must have a clear understanding of purpose	2.0
Problem Solving / Framing	Design requires the designer to generate alternatives and to choose the one that makes the best sense	3.0
Communication; Collaboration	There is greater power in a solution reached by common effort	3.0
Problem Solving / Framing	Designing is a planning activity, design requires a clear plan	4.0
Abstraction; Decomposition	Use of abstraction - design moves from the abstract (logical) to the detailed (physical)	3.0
Constraints	Designers must be aware of existing technology	3.0
Iteration; Prototyping; Storyboarding	Visual/verbal exploration of ideas is a useful technique to help clarify and develop ideas	2.0
Prototyping	Prototyping is a useful design tool that helps explore and evaluate design ideas	3.0
Constraints	Awareness of practical constraints - context and environment	4.0
User-requirements	Some level of user/client involvement in the design process is required	2.0
Refinement; Iteration	Testing is an integral aspect of designing	1.0
Functionality	A designed artefact must fulfil the intended function	1.0
Context-complexity	Choice of design depends on the nature of the problem/proposed product	1.0
Functionality	Fitness for purpose	2.0
Functionality	Functionalism - a design must work	3.0
Pragmatism	Pragmatism	2.0
Re-use	Reuse	1.0

Table 14: Evidence found in Graphic Design– textbook trawls (Gallagher, 1999)

Category Code(s)	DM Statement	Weight (Max 8)
Communication; Compromise	A compromise must be made between conflicting priorities	6.0
Abstraction	Abstraction as a feature of design is a good thing	6.0
Re-use	Design for re-use	4.0
Decomposition	Benefits of de-composition/composition outweigh disadvantages	6.0
Users	Users cannot be treated as objects or reduced to roles	5.0
Influences; Pulls	Following fashion (a specific example is given)	5.0
Decomposition	Solutions can be better generated by a process of breaking the complete process into smaller bits	7.0
Presentation; Influences	Consideration of aesthetics	2.0
Notations	There are good examples for the number of elements in a diagram	2.0
Balancing	Controlling the software process produces a better product	6.0
Balancing; Constraints	Controlling the software process produces benefits that outweigh the costs	5.0
Good Design	Efficiency	3.0
Good Design, Motivation	Elegance	5.0
(-) Context-complexity	The software development process is capable of being managed	6.0
(-) Context-complexity	The cost and timescale of a computer system development can be estimated in advance with a reasonable degree of accuracy	4.0

Table 15: Evidence found in Software Engineering– textbook trawls (Wernick, 1995)

That less support for the model was obtained from Wernick's thesis may be explained by the extent and direction of his research. Firstly he conducted less textbook trawls than Gallagher (eight rather than twelve). Secondly Wernick's focus was on quality whilst Gallagher's focus was on design. Although there is a fair degree of overlap between these two concepts clearly there is greater commonality between this study and Gallagher's. Thirdly, Wernick studied only software engineering, Gallagher studied both software engineering and graphic design. Finally, statements found in support of the model were derived from Wernick's listing of common elements (elements which united the discipline in a single Disciplinary Matrix) only. No attempt was made to search for supporting statements in the (much longer) list of elements that, he claimed, divided the discipline into competing schools. Therefore the set of data searched for supporting evidence was smaller in Wernick's case.

The final two paradigmatic elements identified by Wernick are at odds with the predominant design context identified in this study. "Estimation with a reasonable degree of accuracy" and "management of the process" are statements that do not sit well with a design context described as complex, volatile and uncertain. However they do comply with the description of the "Contra context" which is less complex, more stable and certain. In such a context estimation, planning and monitoring is easier, and in terms of this analysis, closer to textbook theory.

Overall, greatest support was found for strategies - particularly cognitive and interactive strategies, context - particularly the importance of **constraints** on design, and definition and description - through the identification of **functionality, structure** and **presentation** elements. However there was little or no support for the relationships between these elements or for those unconscious or semi-conscious factors that '**pull**' a designer into doing a design. This is unsurprising. The paradigmatic elements identified by Wernick and Gallagher are statements of beliefs and values and not of observed action and process. There is therefore little scope for these to include transactional cause and effect relationships. It is also unsurprising that textbooks on software engineering or graphic design do not include references to subtle interactions between context and designer. These themselves are controversial and not usually considered in discipline specific textbooks. Indeed, the identification and development of these relationships may be considered a contribution of this work. Table 16 summarises the findings of the comparison.

Category	SE (1999)	GD (1999)	SE (1995)	Totals
Functionality	1	7		8
Structure	1	1		2
Presentation			1	1
Entropy		4		4
Problem Solving /Framing	2	2		4
User-requirements	2	1	1	4
Designer-Motivations	2	3	1	6
Designer-Influences	3		2	5
Design-Pulls			2	2
Context-complexity	5	2	2 (-)	9
Notations	1		1	2
Prototypes	2	2		5
Abstraction	3	1	1	5
Separation	1			1
Decomposition	2	2	2	6
Refinement	4	3		7
Re-use	1	2	1	4
Storyboards		1		1
Iteration	3	3		6
Pragmatism		1		1
Communication	1	2	1	4
Collaboration	1	2	1	4
Balancing			1	1
Good Design	2		1	3
Bad Design	1		1	2
Design-constraints	1	3	2	6
Methods	1			
Total (categories)	40 (21)	42 (18)	21 (16)	113
Percentage Agreement	77.8%	66.7%	59.25%	

Table 16: Summary of comparison with two sources of technical literature

Percentage agreement is a measure of consistency between two or more coders. Here it is used to measure the consistency between the findings of this study and the findings of two cognate studies conducted by two different researchers. Boyatzis (1998:154) states that percentage agreement is most appropriate when the unit of coding and the unit of analysis are the same and when the themes being coded call for yes/no or presence/absence judgements by the coder. In this case, the percentage agreement is the number of categories in common between the original study and the comparison studies expressed as a function of the total number of categories in the original study. Thus

Percentage agreement = no. of categories of Data-set A identified in the Wernick or in the Gallagher study (excluding duplicates) / total number of categories in Data-set A (X 100)

The levels of agreement between Data-set A and the comparison studies are high and this is encouraging, suggesting support in the technical literature for the outcomes of this study.

7.3 Test Two: Using Data-set A

It is possible to use the same data to validate the outcomes of the study. Here the final set of thematic categories is analysed using some simple statistical techniques. The purpose of this test is to establish further support for individual categories and to investigate the level of support within each discipline. An early and significant assumption made in this study was that both software engineers and graphic designers could be regarded as software designers. If this is so then we would expect to see a common set of core categories across the disciplines and a low level of variance between the disciplines. Yet some variance is to be expected otherwise there would not be distinct disciplines. What is common and what is distinct among the disciplines? What does this tell us about software design and each discipline's approach to it? What are the most explanatory categories and do these match the qualitative analysis? Table 17 compares Software Engineering and Graphic Design within Data-set A.

	SE	GD	Total	Mean	Var	X2
Functionality	6	10	16	8.00	2.83	1.60
Structure	7	6	13	6.50	0.71	0.17
Presentation	5	16	21	10.50	7.78	7.56
Entropy	2	5	7	3.50	2.12	1.80
Problem Solving	8	14	22	11.00	4.24	2.57
User-requirements	9	11	20	10.00	1.41	0.36
Designer-motivation	2	11	13	6.50	6.36	7.36
Designer-influences	4	14	18	9.00	7.07	7.14
Designer-pulls	7	9	16	8.00	1.41	0.44
Context-complexity	10	3	13	6.50	4.95	16.33
Notations	11	6	17	8.50	3.54	4.17
Prototypes	7	4	11	5.50	2.12	2.25
Abstraction	5	1	6	3.00	2.83	16.00
Separation	5	1	6	3.00	2.83	16.00
Decomposition	0	3	3	1.50	2.12	3.00
Refinement	10	5	15	7.50	3.54	5.00
Re-use	8	3	11	5.50	3.54	8.33
Storyboards	3	2	5	2.50	0.71	0.50
Iteration	4	3	7	3.50	0.71	0.33
Pragmatism	2	2	4	2.00	0.00	0.00
Communication	6	13	19	9.50	4.95	3.77
Collaboration	3	5	8	4.00	1.41	0.80
Balancing	5	9	14	7.00	2.83	1.78
Good design	17	13	30	15.00	2.83	1.23
Bad design	7	8	15	7.50	0.71	0.13
Design-constraints	17	11	28	14.00	4.24	3.27
Methods	10	8	18	9.00	1.41	0.50
Total	180	196	376			
Average	6.67	7.26	13.93			
Variance	4.10	4.49	6.97			112.41

Table 17: A quantitative analysis of Data-set A

The low level of variance between the disciplines (as indicated by the parametric tests) appears to support the hypothesis that there is a common core of software design (concepts are similarly distributed across categories within each discipline). This is important to both the outcomes of this research study and to an early assumption that shaped its execution – that made in Chapter Four (4.2.2) where both software engineers and graphic designers were equally held to be software designers.

What of the differences between disciplines? These have already been discussed in the main analysis but require further comment here. A significant assumption upon which parametric measures (such as mean and standard deviation) are based is that the distribution is normal. However this is *not* the case when considering the loading of concepts onto categories in Data-set A. Since both the identification of concepts from transcript sources and the aggregation of these selected concepts into thematic categories was driven by theoretical sampling (Chapter Four, 4.3.2.1 – 4.3.2.2) the distribution is in fact closer to the binomial or the Chi-squared distribution.

Relaxing this assumption (the distribution is not normal or, at the very least, is unknown) through the introduction of the non parametric Chi Square test produces a very different picture (X2). Here the variance between disciplines is much greater (X2= 112.41) and the value of X2 is sufficiently outside its range of critical values[34] to suggest accepting the null hypothesis that there is not a common core of software design (the categories are not similarly distributed between the two disciplines). What are we to make of these two apparently contradictory outcomes?

One possibility is error in the calculation of X2. This test should not be used if more than twenty percent of the expected frequencies have a value less than five. The values for the expected frequencies in this case (Graphic Design) clearly shows this to be the case – nine out of twenty seven (33%) have a value less than five. Moreover common sense suggests that the two disciplines are not significantly different. The maximum difference between observed (Software Engineering) and expected (Graphic Design) frequencies is eleven but

[34] At n-1 degree of freedom (26) critical values are 38.9 at 5% and 45.6 at 1%

nineteen (70.3%) of the recorded differences between the disciplines have values of five or less[35].

One solution to this problem is to combine those categories that (a) are conceptually close to each other and (b) currently have expected frequency values (individually) of less than five. This produced a truncated Table 17 wherein the number of categories is reduced from twenty seven to twenty-two through the following amalgamations – **Abstraction + Separation + Decomposition; Prototypes + Storyboards; Refinement + Iteration; Re-use + Pragmatism.** This produces one rather than nine, expected frequency values of less than five (or 4.5%) and a Chi-squared value of X2 = 73.8. This is much more reflective of a simple reading of the table and of the parametric test scores but is still well outside its acceptable range of critical values[36]. Again, on the basis of this data we are inclined to accept the null hypothesis that there is not a common core of software design (the categories are not similarly distributed between the two disciplines).

In fact we want to accept *both* hypotheses. The qualitative analysis of data shows that software design can be conceived as consisting of a common core of categories, shared by the two disciplines, but that it also consists of categories that are unique to each discipline, or at least that have greater emphasis in one discipline than in the other. This is what a simple reading of Table 17 suggests and is confirmed by parametric tests of variance. It is also consistent with Gallagher's (1999) analysis of paradigms in the field. Yet non parametric tests (Chi-squared) suggest that there is no such common core, or at least that such a common core cannot be proved statistically.

---

[35] It is important to point out that not all concepts were assigned to these thematic categories and that one concept could be assigned to more than one category – in fact the average is two.
[36] At df=21, these are 33.9 for 5% and 48.3 at 1%

However statistically, we cannot accept *both* hypotheses. Pragmatically we must neither accept nor reject the hypotheses on the basis of this data but reserve judgement pending further and more detailed statistical analysis[37]. This analysis is not included here but is further discussed in Section 7.5 of this Chapter and in Chapter Eight, under further research (8.6.2).

7.4 Test Three: Using Data-set B

In Chapter Four a second data set was introduced derived from interviews with software designers published by Lammers in 1989. This data set (Data-set B) will now be used to validate the original analysis. Would the same analysis produce consistent results in another dataset (internal validation)? Would the outcomes of the main study translate to another design domain, from design-in-the-small to design-in-the-large (external validation, reliability or generalisability)?

Appendix 6 lists the concepts generated from the open coding of Data-set B. As explained in Chapter Four these were the outcome of a coding process driven by theoretical sampling – at this juncture in the study further evidence was sought on context-complexity. Nevertheless a range of concepts was identified. Although the coding process at this point did not extend to axial coding and the development of categories it has been possible, subsequently, to read through each concept record (and associated memo records) and to match these to the set of thematic categories derived from Data-set A. Table 18 shows the outcomes of this process.

---

[37] At least this "sitting on the fence" avoids the possibility of Type 1 or Type II errors – rejecting a hypothesis when it should be accepted or accepting a hypothesis when it should be rejected.

	A	B	Mean	Var	X2
Functionality	16	0	8.00	11.31	16
Structure	13	0	6.50	9.19	13.00
Presentation	21	0	10.50	14.85	21.00
Entropy	7	0	3.50	4.95	7.00
Problem Solving	22	11	16.50	7.78	5.50
User-requirements	20	0	10.00	14.14	20.00
Designer-motivation	13	9	11.00	2.83	1.23
Designer-influences	18	19	18.50	0.71	0.06
Designer-pulls	16	14	15.00	1.41	0.25
Context-complexity	13	19	16.00	4.24	2.77
Notations	17	5	11.00	8.49	8.47
Prototypes	11	4	7.50	4.95	4.45
Abstraction	6	11	8.50	3.54	4.17
Separation	5	0	2.50	3.54	5.00
Decomposition	3	11	7.00	5.66	21.33
Refinement	15	10	12.50	3.54	1.67
Re-use	11	20	15.50	6.36	7.36
Storyboards	5	0	2.50	3.54	5.00
Iteration	7	3	5.00	2.83	2.29
Pragmatism	4	1	2.50	2.12	2.25
Communication	19	0	9.50	13.44	19.00
Collaboration	8	0	4.00	5.66	8.00
Balancing	14	0	7.00	9.90	14.00
Good design	30	1	15.50	20.51	28.03
Bad design	15	2	8.50	9.19	11.27
Design-constraints	28	1	14.50	19.09	26.04
Methods	18	1	9.50	12.02	16.06
Total	375	142			
Mean	13.89	5.26			
Variance	7.01	6.72			271.188

Table 18: Statistical comparison of Data-set A and Data-set B

In this analysis both the results of parametric and non parametric tests for variance are greater than those reported for Data-set A in Table 17. In fact X2 (at 271.188) is almost two and one half times greater than X2 for Data-set A. This is as one would expect. A glance at the Table shows significant and repeated differences between the two domains.

This is highlighted in Table 19 below

Concepts in range	1-5	6-10	11-15	16-20	21-25	26-30
Data-set A	4	4	8	7	2	2
Data-set B	18	2	4	3	0	0

Table 19: Distribution of concepts by discipline

Many of these differences can be quickly discounted as the consequence of the nature of the data collection and analysis conducted for Data-set B. The focus at this stage of the research was on complexity and data was sampled and analysed based on this construct. No attempt was made for example to look for evidence of good and bad design or to determine a definition of software design. Where such evidence emerged it was recorded but this was not the focus of the analysis.

In Data-set B there was little evidence in support for interaction strategies (save for a few isolated references to teams). This is explained by the nature of the designers and the projects on which they worked. Lammers (1989:3) chose her subjects because they were "pioneers who shaped the software industry" and like most pioneers they are (or were) essentially loners. Almost all worked on their own, ran their own company or both. Many have unconventional backgrounds, few have ever had to work routinely as part of a design team. The contrast with Data-set A is marked.

Data-set B was analysed to gain further insight into the core category of **context-complexity – action/interaction**. Since we may expect complexity to be greater in bigger projects we may expect a calibration of responses identified in Data-set A. This was the case. The strategies of **abstraction, decomposition, refinement** and **re-use** were much in evidence. However an additional level of response at a higher level of abstraction, not identified in Data-set A (Chapter Five, 5.5), emerged.

In this data set responses to complexity were identified as either **avoidance strategies** or **engagement strategies**. Faced with complexity the designer may be reduced to a 'fight or flight' instinct.

> "Then you have two choices, either back off to some other problem you do understand, or think harder"[SE2-3]

Avoidance strategies can be further subdivided into those that seek **permanent avoidance** and those that seek only **temporary avoidance**. Permanent avoidance strategies include "backing off" to another problem one that is easier or otherwise more amenable to solution by the designer. Temporary avoidance strategies include doing something else first, (for example doing a simpler task or taking a holiday, holding off, not committing to a solution too early, tackling the simple stuff first). Each strategy has the same objective - to avoid complexity - but whereas the former seeks to do this indefinitely, the latter seeks only to delay the engagement. By definition temporary avoidance will result in engagement but temporary avoidance may become permanent avoidance when a designer, having initially postponed tackling complexity, eventually confronts it and finds it too difficult. On the other hand, a strategy to walk away from complexity (permanent avoidance) may be revoked after experience and confidence has been gained on other tasks.

Holding off as long as possible avoids a "corpus of code building up" and makes it easier to change direction as the design evolves but this implies additional cognitive effort and the designer cannot hold a design in his head forever. [SE12-2] However, as the program is developed, this becomes difficult "At some points the code gets explosive and I have everything inside my brain at the one time" [SE4-5]. Holding on to a mental model can be very efficient "once your in the grove" but loose it and "you've got to work on it quite a

while to get back in." [SE6-12] Moreover "any time there's a flaw in this great mental simulation, it turns into a bug in the program"[GD2-1]. When this occurs the programmer can "feel pretty bad" because "once your mental simulation is imperfect, there might be thousands of bugs in the program" [SE5-6]

Some designers will then **postpone confronting complexity**, choosing to tackle the easiest units first (here, an avoidance strategy is nested within an engagement strategy).

Some however prefer to **confront complexity early**. This is a conscious decision by designers to confront the most complex aspect of the design first, in the belief that unless this part of the design can be completed, or at least assurance obtained that it can be completed, then there is little point in proceeding with the rest of the design. "You start at the point where you think it's too hard to solve, and then you break it down into smaller pieces."[SE4-1]. This strategy comes with experience and confidence and less experienced designers prefer a more cautious approach.

Sometimes a lack of experience can lead a designer to fall into a **"do something/anything approach"**. Here the pressure to get "something done" panics the designer to rush in, to commit to something too early. [SE6-13] Thus, engagement may mask superficial or surface approaches to problem solving. Combined with the lack of experience that initiates such action, this strategy is nearly always doomed to failure. A related problem is when a designer does not understand a design but attempts to modify it, causing multiple additional problems [SE5-3].

As in Data-set A, **decomposition** is often accompanied by, or closely followed by **refinement**. This often involves **prototypes** - either build one to throw away [SE3-4] or incremental development [SE4-2]. Get something working very quickly, learn form small

experiments rather than large ones [SE3-4] write small pieces of code and " improve and monitor it along the way" [SE4-2] " get the program up so that it just begins to work, and then add features to it" [SE9-6]. This process is often, but not necessarily, sequential.

> " When I'm trying to solve a problem that has a series of steps, I take them in order, one at a time - step A, step B, step C. I've tried but I just can't work on C until B has been completed."[SE4-5]
>
> "I try to do the minimum that will get me one step further. Within a goal, within a step, I take the minimum subset." [SE7-1]

Refinement involves "building a skeleton", starting out with a basic framework and adding features to it [SE9-6] trying out different approaches, and re-working them if need be, growing the program [SE8-4], design by "successive approximation" [SE10-2]. One programmer made an analogy between writing code and sculpting a clay figure. You start with a lump of clay, shape it, add more clay, shape it and so on. Sometimes a chunk of clay is torn off and discarded. There is a high degree of interaction between artist and material, between programmer and code [SE7-3].

Re-use, an important design strategy identified in Data-set A, is also a means of engaging complexity. Designers readily acknowledged the extent to which they drew upon their own experience and the experience of others in writing code. In part, this practice is under-pinned by a belief that design is design. For example, all programming is essentially the same, involving a few basic algorithms, loops and conditions [SE9-5] and all programmers are very alike [SE5-9]. In part it is a pragmatic recognition that success comes from doing the same thing over and over again, improving with each attempt [SE9-5], that one keeps running into the same kinds of problems [SE5-9] and that most products are designed by successive approximation and refinement by a number of people [SE10-2].

The many types of re-use are again highlighted. Re-use is made of models [SE2-4], algorithms [SE3-3], language structures [SE3-1], tools [SE4-4] code [SE9-3; SE12-3] processes and procedures, products [SE6-10] other people's problem solving approaches [SE4-4], ideas [SE9-3, SE5-3] and tricks [SE9-7]. Outside personal experience, sources of re-use include project reviews [SE5-3], conversations with other programmers, examining program listing and reading the relevant literature [SE9-7]. Some programmers do not like to use tools or programs written by others [SE12-7]. The potential for component, modular, off-the-shelf software is recognised [SE12-7]

> "One sign of very good programs is that even internally they follow that philosophy of simplicity. If they want to do something complex, they call the code with simple operations internally, rather than doing the complex operation from scratch". [SE5-8]

Another response, or strategy for dealing with complexity, is to **hide it**. This is regarded as a real challenge for software engineers - designing something that is complex on the inside but simple on the outside. End users appreciate this (though the industry may not as products that flaunt complexity often get favourable reviews) because

> "as they become more aware of what the computer can do for them, their demands increase but their desire for complexity doesn't. They want the new programs to do more but stay simple". [SE6-4].

The same designer claimed writing a complicated program is easier to write than straightforward program, in the same way automatic transmissions are easier to design than manual transmissions because "you reflect the complexity back to the user"[SE6-1]. In order to achieve simplicity you 'have to master complexity' [SE2-3] but the 'cult of simplicity' is 'highly suspect' and it is only through the understanding of complexity (such as is occurring in Mathematics) that real advances can be made [SE1-2].

In Data-set B we can also identify a number of **design pulls**, primarily but not exclusively as the **pull of complexity**. Hiding complexity is a 'fascinating challenge' [SE6-3] and a

source of enjoyment. "Programmers love to tame complexity" [SE6-11], get "a kick out of solving something that looks hard, and making it look easy"[SE3-4] and "feel great" when they figure out how to make a complicated process simpler [SE5-4]

> "If I can solve it, I can do something everyone thought was impossible. That gets my adrenalin going and my heart rate up. I just love it. I get like a dog with a bone - *I will not put it down.* I think about it driving around, swimming up and down and in the shower. I just tease the problem to death until I find some way of solving it using some technique that nobody's thought of." [SE6-13]

A well written program, just like a well built car, a well built bridge, or a well built building, from an engineering point of view, is "very elegant; it sings"[SE7-2]. Programming is the ultimate field for someone who likes to tinker [SE10-5] and "one especially neat feature of programming is that its very clear when you do it well"[SE11-1]. "When you program, you fall in love with the fact that you can handle all of these complex abstract elements. You think that this has to be that way" [GD3-3].

Therein lies the danger. A programmer may be **seduced by complexity** and quickly get out of his or her depth [SE8-4]. When a programmer does not understand complexity and does not benefit from the experience of others, "then they too will get burned" [SE2-3]. Modern computing environments increase rather than diminish the danger.

> "The computer is a very alluring machine, it always tempts you to do one more thing. If you're word processing, you want to get every last typo correct; if you are programming, you want to put every last feature in. It's good to know when to stop".[GD1-3]

The tools are better, the programming environment is better and the languages are more expressive but this creates more opportunities for more mistakes [SE3-2]. There is a danger that programmers think they can do anything, therefore they overreach themselves and fail [SE2-5].

Design is also an intuitive exercise. "The best software comes from the realm of intuition" [SE12-6] and requires some curiosity [SE8-5]. "Once you get to certain level of

experience, you go from the idea to the program without ever thinking about all the intermediate steps; the process becomes automatic" [SE9-3].

> "The actual coding process has always been a little scary for me, because I don't know if I am writing the right code, nor do I know what I'll write next. It just seems to come out. Sometimes I realise the code is not exactly right but I also realise intuitively that it will relate to something else - it will factor out and become right even if I don't know exactly how at the time I am writing it." [SE4-3].

This is a reminder that it would be wrong to make too much of conscious or explicit strategies and that experienced designers as well as inexperienced ones have difficulty in accounting for their actions. The designer may apply an overt strategy for dealing with complexity but may also have covert or hidden strategies. For example he may change his work routine or simply complain to someone else. These too are valid strategies for coping with complexity.

Finally, suggestions of a (technical) **complexity threshold** can be found in references to abstraction and cognitive overload.

> "At some points the code gets explosive and I have everything inside my brain at the one time; all the variable names and how they relate to one another, where the pointers start and where they end, disk access, et cetera. All sorts of things go on in my brain that I can't put on paper simply because I'm always changing them." [SE4-5].

Here the designer is describing a complexity threshold. There is a perceptible difference between the situation before this point (when the code explodes) and the situation afterwards.

> " when the code explodes it becomes tough because I'm working under pressure to get the code back together. When you've got code all ripped apart, it's like a car that's all disassembled. You've got all the parts lying all over the garage and you have to replace the broken part of the car or the car will never run. It's not fun until the code gets back to the baseline again" [SE4-6]

How does this changed context affect strategy? There is no evidence that new strategies are adopted. Rather existing strategies are calibrated to take account of the new conditions. The designer tries harder, the cognitive effort becomes more intense, iterations and interactions increase in amount and frequency. The stakes are higher because failure

at this level is more costly and visible but the general problem solving approach remains the same. Managing complexity often comes down to hard work "you always get an answer if you work hard enough at it" [SE7-5] and learning from experience

> "A lot of people don't understand what the consequences of complexity are, and without that understanding they are likely to get burned. If they are not willing to take the word of someone who has gotten burned, then the only way they are going to find out is to try it and get burned themselves" [SE2-3].

Sometimes however, the designers have to accept that what they are trying to do is just too hard [SE2-5].

7.5 Discussion

Strauss and Corbin (1990:191) caution that, whilst the combination of quantitative and qualitative methods is perfectly sound,

> "unless the quantified findings are integrated into the theory, made part of the theory itself through the paradigm, they will be merely an aside".

In this chapter, further evidence in support of the theory, from the same and a different data set, has been presented at the levels of open coding (identified common concepts), axial coding (causal relationships) and selective coding (the core category of context-complexity). As a result of this analysis a number of observations can be made on the applicability of the model and theory presented in Chapters Five and Six.

1. Design strategies remain the same despite an increase in size and complexity of project. The same basic strategies were identified in Data-set B as in Data-set A. What seems to be different is the way in which those strategies are employed (strategies are calibrated to the prevailing context).
2. Yet a further categorisation of (the same) strategies, at a higher level of abstraction, was identified. These are "avoidance" and "engagement" strategies, "holding off", "confronting it early", "do something / anything approach".

3. Of all the existing strategies identified in Data-set A, re-use was the most prevalent in Data-set B and the range and depth of the category is considerably extended as a result. Of particular interest is the high level of personal and design re-use identified in Data-set B.
4. Finally, the category of design-pulls was corroborated and extended in the second data set, primarily because of the increased influence of complexity at this level.

All these finding have implications for the development of methods and tools. In addition, the higher order categories (2) may prove valuable in any further organisation and understanding of design practice.

7.6 Chapter Conclusion

In this chapter three tests of the analysis presented in the previous chapters have been applied. These involved a mixture of quantitative and qualitative analysis, parametric and non parametric statistical techniques. From these we may conclude that (a) there is support in the technical literature for the categories of the inductive model (b) a common core of software has been identified along with significant differences between disciplines, however this has not been proven satisfactorily in the statistics presented, (c) significant differences between Data-set A and Data-set B, design-in-the –small and design-in-the-large have been identified but a common approach is also evident. This has been identified in generic design strategies such as abstraction, decomposition, and prototyping. In addition in Data-set B a number of higher order design strategies have been identified, together with a wider range of re-use and greater depth to the pull of complexity.

Chapter Eight: Conclusions and recommendations for further research.

> "that's the trouble with design everyone has got an opinion, everyone is a bloody critic" [graphic designer]

8.1 Introduction

The research questions first posed in Chapter One that have inspired and guided this research are returned to in the final chapter to structure its conclusions. The third research question –how may such knowledge [of software design] inform interventions to improve design practice? – is a challenge of relevance and precedes a discussion of thesis contributions. Firstly however, thesis outcomes are "grounded" in existing theories and models of design through a review of related literature. Following a discussion of thesis contributions the research is evaluated using criteria set out by Strauss and Corbin (1990:252-257). Arising out of earlier discussions and this evaluation are some recommendations for further research. The chapter (and the thesis) concludes on a philosophical note – and a final contribution is suggested.

8.2 Thesis conclusions

The three research questions first posed in Chapter One are used to structure the thesis conclusions. Each question is addressed using empirical data and the general literature. Where possible the literature informs further development of the research outcomes. At the end of this section, the conclusions are summarised in advance of a discussion of thesis contributions.

8.2.1 [RQ1] What is software design?

A practical difficulty with the definition of software design developed in this thesis is that its distinctiveness criteria are non-deterministic. That is, it is not possible to delineate its boundaries, to say what is specific and what is not specific to itself. Such criteria as

problem solving, presentation, structure and function are not specific to design but may be commonly found in definitions and descriptions of other phenomena. Therefore defining software design by the existence or absence of such criteria is not a powerful definition. There is no *differentia specifica* of the phenomenon presented in this thesis. Nevertheless, a contribution to the definition of software design may be claimed. Nachmias and Nachmias (1976:17) point out

> "definitions that describe concepts using other concepts are conceptual definitions..the significance of this observation is that a conceptual definition is neither true nor false"

They argue that conceptual definitions need to be turned into operational definitions, that is to give conceptual definitions specificity through dimensions of time and space. This bridges the gap between the theoretical – conceptual level and the empirical – observational level[38]. Nachmias and Nachimas define an operational definition as

> "a series of instructions describing the operations that the researcher must carry out in order to demonstrate the existence, or the degree of existence, of an empirical occurrence represented by a concept" (1976:17)

In this thesis, such an operational definition has been given. Although each category is a "concept" described using other concepts, each is given specificity through its properties and dimensions. Thus software design is defined by the dimensional values of its properties – structure, function and presentation. Moreover the way in which the analysis and each code is set out (described in Ch 4, listed in Appendix 5, according to Boyatzis' criteria for good code) goes some way to meet the requirements set out by Nachmias and Nachmias.

---

[38] I am indebted to Bride Mallon for this insight

8.2.2 [RQ2] How do software designer's design? (What do software designer's do when they design and why?)

An apparent paradox in this study was the fact that whilst designers defined design as a problem solving process, their descriptions of that process suggested something more. Evidence that designers believe design to be a problem solving process can be found in comments such as "in practice design comes down to solving problems and this is what designers should be taught (along with communication skills)", "failure to solve problems is a failure to design"and "the designer has been likened to a technician, following a series of logical steps to solve a problem, someone who knows that doing x will work under certain circumstances"[39]. Evidence that design is something other than simple problem solving can be found in the frequency and significance of design iterations, the nature of the design context (uncertain, volatile and complex) and in the difficulty in scoping design. Much of what designers do, in particular their efforts to define and specify the problem, could be described as problem framing rather than problem solving.

If software designers actually do believe design to be a rational problem solving process, whether or not their descriptions of their actions when designing suggest something much more complex, this would, according to Winograd and Flores (1987), be an "impoverished view" leading to "impoverished design". The implications for the use of methods, language and tools to support the process would also be disturbing. If a traditional problem solving approach to software design is being followed (even though rational decision making is bounded by the constraints associated with any design), with attendant use of methods tools and languages to support that approach, there is at least a

---

[39] In the literature there are many references to software engineering as a problem solving. For example, Glass (1996) gives a spirited if somewhat overstated case for the software life cycle as a general problem solving model.

suggestion that process and problem may be incompatible. Butterfield (1998) in an exploratory study, suggests that analysts do not conceptualize complex systems projects using the traditional problem solving model (the IPO model on which, he argues, many methods such as JAD, RAD and OOD are based) but instead use a range, of much less formal, approaches. In Chapter Six (6.4.2, 6.4.3) it was suggested that (such) mismatches between design context and design strategies constitute a design breakdown. This is further explored later in this chapter (8.3.3).

Design is by its very nature processural (Budgen, 1994). The process of design is commonly described in the literature as a series of steps Requirements – Build – Test – Implement (see for example, Sommerville, 1992) and in this study by designers. However evidence of the activity of design (how designers actually design) suggest that design is not linear and progressive but rather a pattern of responses to prevailing conditions. It is purposeful and non purposeful, characterised by flexibility, expediency and opportunistic behaviour.

Strauss and Corbin (1990:152) refer to such a phenomenon as "non-progressive movement" and "purposeful altercations or changes in action / interaction in responses in conditions, but movement that does not necessarily occur in stages or phases". They give as an example, chronic illness. An individual does not necessarily or ordinarily move between phases of the illness but will seek to manage it by keeping it as stable as possible, or even reversing it. So too for example, with the concept of **balancing** identified in this study. The objective of the designer is not necessarily or ordinarily to move through the stages of design but to manage the complexity of the design by balancing competing constraints. At various times this balancing may mean stopping progression, or accelerating it or simply treading water until a more favourable set of conditions pertain.

Such observed behaviour is at odds with descriptions of design as a problem solving process. Design consists of both progressive and non progressive movement but the significance of the latter is often overlooked.

8.2.3 [RQ3] How does such knowledge inform interventions to improve software design practice?

Some implications arising out of research questions one and two have been highlighted. A number of others are now discussed. This is by no means exhaustive but includes those with (a) clear relevance to the phenomenon under study and (b) strong support in the data.

8.2.3.1 In software design there is no clear distinction between definition and development

According to Pressman (1982:35) definition is the 'what', development is the 'how'.

Development includes software design which "translates the requirements for the software into a set of representations that describe data structure, architecture, interface etc". A specification says what is to be done and a design says how to do it and serves as a precise medium of communication between members of a development team working on a large system. Sommerville (1992:180) rejects such a distinction as ambiguous and meaningless.

In this study a distinction is made between the separation of definition and development as a design *goal*, and the situation as it prevails in practice. Most, if not all, software engineers advocate the separation of functionality and implementation. A text editing system was used to illustrate this separation of concerns. This will require the facility to delete text (a functional requirement) and this requirement may be implemented using a keyboard, a mouse or a combination of both, but the implementation should not be

physically rendered to the functionality. Software engineers fear that graphic designers too often do not seem to appreciate the significance of this separation. Yet they also recognised that the separation of functionality and implementation is becoming harder to preserve in practice. The interface is becoming much richer and more complex and the traditional boundary between presentation and functionality (well defined in conventional software) is becoming increasingly blurred. Of course, the more complex the system the more important is the separation.

This has implications for the support environment. If the separation of functionality and implementation is important to communication (as some software engineers maintain) then any blurring of the boundaries may lead to communication difficulties. Also since the purpose of language is to describe the specification, where the specification is not clear or becomes merged with implementation issues, a language may prove inadequate. One consequence of the merging of design stages has already been noted in this study – it is difficult to scope a design and estimations are difficult.

8.2.3.2 Software design is a process of refinement

Programming has long since been described as a process of refinement (Wirth, 1971, Dahl, Dijkstra and Hoare, 1972) and refinement continues to be the paradigm underpinning today's most popular programming methods. Here the original specification described in abstract concepts is transformed into a working programme through a sequence of successively more refined and concrete steps. This process (what Agresti (1986) calls transformational implementation) is represented by the following notation

S(pecification) --> P(rogram)

In this study software design has been described as a process of refinement. This includes programming but also those activities that take place *pre* specification - primarily requirements gathering and analysis. This expanded process is represented thus

A(bstraction) --> S(pecification) --> P(rogram)

Here refinement describes those steps prior to the development of the specification (what Agresti (1986) calls the operational specification) to those steps involved in the development of the specification and to those steps that follow it. In practice it is difficult to distinguish each phase precisely because of the number of iterations involved. Significantly, refinement in software design does not end with the program or shipped product but continues to describe the relationship of designer and artefact in use. Thus the designer is involved in enhancements and maintenance long after the design has been officially designed. This is illustrated in comments such as "design is ongoing" and - in relation to Web design - "design is re-design".

This process description has implications for the wider software development environment. Methods must include guidance on those crucial activities pre specification - requirements gathering and analysis, languages must describe functionality in a way that both designers and users understand, and tools must support the communication between designer and user. Moreover, methods must support those activities post shipment that cover the design in use. In truth we are well aware of these criteria for they have repeatedly appeared in evaluations of software development methods. Of interest here is the application of these criteria to multimedia development methods too many of which are based on engineering type processes (see Chapter Two, 2.4)

8.2.3.3 Software Design relies on the extensive use of prototypes

Agresti (1986:99) defines a prototype as "a simulation or model generating functional behaviour early in the design process but not efficiently, expressed in a language or form that allows it to be evaluated or interpreted to show systems behaviour". It acts as a medium for users and designers to discuss the intended behaviour of the system. Two issues identified in this study have wider significance in this respect. Firstly prototyping has been defined to include story-boardying and the importance of the storyboarding process to user-designer and designer-designer communication has been recognised. Although originating within and used extensively throughout the graphic design community, storyboardying is increasingly recognised as a valuable approach by software engineers. In particular its role in determining user requirements has been stressed.

Secondly, prototypes are rarely thrown away but incorporated into the design cycle beginning at the requirements phase (there was no evidence that prototypes were being developed separately by end users). According to Carey and Mason, (1985) this has three consequences for the wider design environment. Firstly the underlying paradigm is assumed to be the traditional life cycle - and it is intended that, at some point, users will agree to the prototype and that it will become an input to the design phase. This is consistent with the widely held view, recorded in this analysis, that design is a problem solving process. Secondly, tools should exist to facilitate the development of the prototype into the final product, for example to permit screen display components to be transferred to later stages without re-coding. Thirdly, the prototype becomes part of the systems documentation and therefore must be maintained through explicit version control.

8.2.3.4 Re-use is an important design strategy

Brooks (1995:223-224) identifies re-use as an important "attack on the essence of building software." He draws an example from mathematics to suggest that the amount of re-use will be determined by the operation of simple cost/benefit logic - if a piece of software costs more to build than to re-use it will be re-used, if not it won't. However poor descriptions of existing modules and a lack of a standard nomenclature (unlike in mathematics) raises the cost of re-use as greater effort must be made to discover what has been done and how, than to start afresh. He goes on to refer to a number of studies on re-use but concludes that there "is not nearly so much of it as we had expected by now." In contrast to the pessimism of Brooks (and of DeMarco 1990) in this respect, Poulin (1999:100) argues that re-use is widespread in practice and that there are many success stories. Re-use has made considerable progress in recent years and many of the major obstacles have been overcome (he cites library re-use, domain analysis, metrics and organisational re-use). He concludes that re-use "will significantly contribute to our ability to meet our society's voracious appetite for software".

None of these contributions are concerned specifically with *design* re-use. Yeh (1990:13) identifies three types of re-use in software development. There is the re-use of program parts and "a piece of software is re-usable when it is interpretable, incorporable and portable". There is re-use of systems and there is re-use of design. Software and systems are components that can be re-used during run time but the re-use of design involves the application of rules at a high level of abstraction and is therefore the most difficult. Whilst abstraction entails flexibility, flexibility entails room for error. The inductive model presented in this thesis emphasises re-use as an adaptation to the environment. Existing designs are invariably re-used in new situations by adapting or modifying constraints or

design commitments. In addition the importance of personal re-use (ideas, plans, influences) has been highlighted – particularly in Data-set B - and the need for productivity tools to support such re-use must now be noted.

8.2.3.5 Software design is a social activity

The importance of interaction in the form of **communication** and **collaboration**, between designers and users and between designers and designers, has been identified in this study. This has significant process implications, well documented in the literature (from Brook's (1975) Tower of Babel analogy to Myers' (1999) call for early and better communication between designers and users). Winograd and Flores (1987) identified the importance of communication in software design and developed a specific tool - Co-ordinator - and method - Action Workflow -to address this deficiency. Efforts to improve communication in software development can also be found in brainstorming or argumentative approaches to design, see for example, Rittel and Weber (1973) and more recently in CSCW and Groupware studies (Greenberg, 1991). It is noted that in this research study a number of designers complained about the lack of education and training resources devoted to these critical design skills. An intervention in this area may prove valuable over the long term.

8.2.3.6 Summary

In response to the three research questions posed in Chapter One a number of conclusions (and contributions) have been identified. Notwithstanding the difficulty in giving a truly distinctive definition of software design, an operational definition has been given that makes it easier for other researchers to understand and to apply it to their own work (8.2.2). A discrepancy between *in vivo* definitions of design as a problem solving process and descriptions of design as something akin to a problem framing process was noted, as

were some implications for the support environment (8.2.3). Moreover, it was observed that software design sometimes consists of non progressive movement and that such activity needs also to be supported by methods (8.2.2). There then followed a number of unsurprising but significant observations (8.2.3.1 – 8.2.3.5). In software design there is no clear distinction between definition and development, software design is a process of refinement, it relies on the extensive use of prototypes, re-use is an important design strategy and software design is a social activity. Each of these has implications for practice and these were commented upon.

Section 8.4 sets out the major claimed contributions of this thesis. But first outcomes of the research are "grounded" in existing theories and models of design using a selection of related literature.

8.3 Grounding thesis outcomes in existing theories and models of design and problem solving

It is possible, and productive, to map existing theories and models of design directly to the theoretical framework presented in Chapter Six. The process is recursive. An outcome of the research is re-used to "ground" the empirical data in the literature and the literature is used (where appropriate) to reflect back upon the theoretical framework.

8.3.1 Scenario 1: the "problem solving" context (low context-complexity – low interaction)

This context is most easily identified with Newell and Simon's (1972) theory of general problem solving. The design context can be said to exhibit a high degree of structure. Structure is the amount and quality of information in the task environment that can be used to guide the problem-solver. This information covers the problem initial state, the final or goal state and the operations necessary to get from one to the other. In this context

(for example, in puzzle solving, EOQ's or "back end" engineering design) the structure of the task is well defined at the outset, problem solving is goal oriented and proceeds sequentially as a search through the problem space. However, since the problem space is an abstraction of the task environment it will not contain all the structure (information) available.(Newell and Simon, 1972:823-825). In addition, information may exist in the task environment (or indeed in the problem-solver) but may not be accessed in the right place at the right time. Therefore, a problem space may contain more or less structure than the environment it represents.

Hinrich (1992) provides a taxonomy of problem solving approaches within which he distinguishes between design as a task, design as a process and design as cognitive activity. Design as a task is predicated on the view that design is a routine, completely defined, hierarchically structured problem space searchable by algorithms and heuristics (he cites as one instantiation of such assumptions, McDermot, 1980). Design as a process includes synthesis, hierarchical refinement and transformation approaches and each assumes that the problem space is basically a graph and solutions can be found by searching through (and 'pruning') the graph (e.g. Williams, 1990). Cognitive models of design (e.g. Goel and Pirolli, 1989) are constructed bottom up based on the invariants of human information processing or top down based on the properties of the task and on observable behaviours of designers. The latter "is how most cognitive models of design are derived" (Henrich, 1992:6) and include the analytical tools of task analysis, complexity analysis and protocol analysis.

Support in this data for the traditional problem solving approach to design was initially and superficially high. That is almost all designers defined design in terms of problem solving and many in their descriptions of the design process used terms entirely

compatible with the rational decision making model. However, as already remarked upon, there soon appeared a crucial distinction between what designer's said they did and what they actually did based upon further analysis of (their) process descriptions. The implications of such a discrepancy have already been discussed. A net outcome of the analysis was that most designers do not occupy this context when they design.

8.3.2 Scenario 2: the "problem framing" context (high context-complexity – high interaction)

Newel and Simon's theory of problem solving is predicated on significant assumptions about the nature of the process. It is clear (from the literature and from the data analysed in this study) that these assumptions do not hold for many, or even the majority, of design problems. In fairness to Newel and Simon, they readily acknowledged that their theory, derived from a small set of studies in cryptarithmetic, logic and chess, would not apply to all problem environments and in earlier and later work addressed the question of less structured problems (Newel, 1969, Simon, 1969, Simon 1973). Moreover they admit that not all behaviour relevant to problem solving is a search through a problem space and that problem spaces can be changed and modified during the course of solving (1972:809).

Newell (1969) argued that ill defined or ill structured problems are simply problems for which no strong problem solving techniques exist. In such cases, he advocated the application of so called weak search methods, such as generate and test. Simon (1973) viewed an ill-structured problem as one yet to be formalized by the problem-solver (whereupon it became a well-structured problem). He pointed out that problems can be well structured in the small but ill-structured in the large. Earlier in another seminal work ('The Sciences of the Artificial', 1969), he suggested that the assumptions under Information Processing Theory (IPT) worked would not be rigid. Design problems did not

lend themselves to an optimal solution and the designer must be prepared to sub-optimise or satisfice in the search for a solution.

It is this modified form of Newel and Simon's problem space hypothesis that can be found to underpin cognitive models of design such as that put forward by Malhotra, Thomas, Carroll and Miller, (1980), the Task Episode Accumulation (TEA) model advanced by Ullman, Dietterich, and Stauffer (1988) and the theory of generic design put forward by Goel and Pirolli (1989). In addition, Simon's concept of satisficing has encouraged the development of a number of constraint based problem solving theories with direct application to the field of design (for example, Sussman and Steele, 1980; Stefik, 1981; Fox, 1983).

In fact, ill structured problems had been described much earlier by Reitman (1964). He defined these as problems for which the initial state, the goal state or the operators are unspecified or only partially specified. He argued that existing models of problem solving assumed that problems were well structured whereas the difficult part of problem solving was not the search but *formulating* the problem.

In this design context a high level of interaction is indicative of attempts to frame or bound the problem. This can be seen in the emphasis on prototypes/ storyboards, the frequent design iterations, the level of communications between designer and clients and between designers and other designers / developers. It cannot be seen, at least on any superficial reading, in the comments of the designers themselves. Elsewhere in the literature, can be found references to the importance of interaction in framing or bounding the problem. Runco (1994) for example, does so in a discussion of problem finding, which includes identification, definition and *posing*.

8.3.3 Scenario 3 and Scenario 4: the "breakdown" contexts (high context complexity – low interaction or low context complexity – high interaction)

One explanation of breakdowns may lie in inflexibility or an inability to adapt to changing circumstances. Elsewhere in the literature, in addition to Blum (1996), adaptive models of design are increasingly prevalent. A feature of such approaches is a desire to retain maximum flexibility in response to the prevailing environment by means of contingency (Flynn, 1992); cognitive fit (Verney and Glass, 1994); morphology (Lawrence, 1981), design rationales (Carroll et al, 1994) and Agile Software Development (Aoyana, 1998). The model of design presented in this thesis stands squarely in this tradition, predicated as it is upon the importance of design context and responses to that context. However two important and inter-related questions remain to be addressed (a) what is the nature of the breakdowns reported? (b) how can the model and theory explain such breakdowns?

As in any analysis using the Conditional Matrix, the phenomenon may be discussed according to its *level* and *type*. Firstly, the description of design breakdowns given in this thesis applies to both interactions with others and to interaction with the materials of design. Although the analysis of breakdowns presented has emphasised breakdowns in communication (with other designers and / or with users) – see for example the Mini case presented in Chapter Six (6.5.1) – breakdowns between context and strategy may equally occur with the materials of design, analogous to Schon's (1983) concept of self reflection. Thus the scenario taken from Data-set B to illustrate a complexity threshold (Chapter Seven, 7.4) can be seen also as a breakdown in the individual designer's communication with the design.

Secondly, the type of design breakdown has been, at least implicitly, negative. Whether a breakdown in communication with others or a breakdown with the design itself the imperative has been to "repair", "recover" or "get back" to the situation before the breakdown, which is considered the normal state. Yet is clear from the literature that design breakdowns can also be positive, even primarily positive. Schon and Bennet (1996:173) observe that breakdowns occasion *reflection in action* "thinking about what she is doing while doing it, in such a way as to influence further doing" or *reflection on action* "pausing to think back over what she has done in a project, exploring the understanding that she has brought to the handling of the task".

Winograd and Flores (1987) also draw upon the philosophy of Martin Heidegger and apply his concepts of "being-in-the-world", "throwness", "present-at-hand" and "readiness-at-hand" to the design of computing systems. Breakdowns are fundamental in shaping the design context and outcomes

> "Breakdowns serve an extremely important cognitive function, revealing to us the nature of our practices and equipment, making them 'present-to-hand' to us, perhaps for the first time. In this sense they function in a positive rather than a negative way" (1986:78)

> " A breakdown is not a negative situation to be avoided, but a situation of non obviousness, in which recognition that something is missing leads to unconcealing (generating through declarations) some aspect of the network of tools that we are engaged in using. A breakdown reveals the nexus of relations necessary for us to accomplish our task. This creates a clear objective for design – to anticipate the forms of breakdown and provide a space of possibilities for action when they occur." (1986:165)

Thus, the breakdowns identified in this thesis may be positive as well as negative and apply to *interaction with the materials of design* as well as to *interaction with others.*

The second question – how can the model / theory explain such breakdowns? – may be addressed through reference to the paradigm model and conditional matrix, the two research tools used in the development of the model and the theory, and inherent features

of it. The theoretical framework identifies four design scenarios or contexts two of which are labelled "breakdown" contexts. For each of these contexts it is possible to use Strauss and Corbin's (1990) paradigm model to investigate the causes, conditions and consequences of such breakdowns. Furthermore the Conditional Matrix may be used to extend this analysis to multiple levels that impact upon the breakdown including personal/individual, sub-organisational/group, organisational, supra-organisational. A conditional path can be traced for specific events or incidents, such as that outlined in Chapter Six (6.5.1.2) or suggested in Chapter Seven (7.4).

Clearly the outcomes of this research are similar in many ways to the conclusions reached by Schon (1983) and Winograd and Flores (1987). The rational decision making model has been rejected, the role of context in determining action is emphasised, action / interaction is at the centre of the design process (for Winograd and Flores these are "speech acts") and design is explained using –at least in part- the concept of "breakdowns". One point of departure is the different interpretations of the context – breakdown relationship. For Winograd and Flores (and to a lesser extent Schon) **breakdowns → context.** In the model / theory developed in this thesis **context → breakdowns.**

In this research the emphasis has been on developing, from first order concepts, a holistic picture of software design (within which breakdowns are one, albeit important, feature) rather than on developing existing theories. The "grounding of outcomes" in existing theory, although legitimate and necessary, should not obscure the inductive nature of the research reported here and the advantages of such an approach.

8.3.4 Summary

Figure 11 summarises the positions developed in the preceding discussion.

Scenario 3 **(Problem solving)**	**Scenario 2** **Problem Framing**
Scenario 1 **Problem Solving**	**Scenario 4** **(Problem framing)**

Figure 11: The Theoretical Framework re-stated

The theoretical framework provides examples of problem solving and problem framing in action. Each context determines a dominant approach. In two of the four contexts the approach is compatible with context (scenarios 1 and 2). In the other two scenarios it is not, a problem solving approach is used when the context demand problem framing (3) or vice versa, a problem framing approach is used when the context demands problem solving (4). The intention is not to create a dichotomy between the approaches. Each approach is valid in its own context and a designer will move between approaches as context changes. Where a designer fails to make such an adjustment a design breakdown will occur. The predominant approach observed in this study was more akin to problem framing than to problem solving. There was little empirical evidence in support of Scenario 4.

8.4 Thesis contributions

8.4.1 A means to identify, explain and predict design breakdowns

The Theoretical Framework is presented as a means to identify and explain design breakdowns. Here a design breakdown is defined as a mismatch or lack of fit between phenomenon and response to the phenomenon, between design context and strategies taken to manage that context (action/interaction). The framework identifies four "types" of **context – response** and for each, use of the conditional matrix and the tracing of

conditional paths provides a means to analyse causes, conditions and consequences, at various levels of observation. Through identification and explanation of existing breakdowns, and in particular through an understanding of the causal relationship **action/interaction → consequences [modified by intervening conditions]**, future design breakdowns may be better predicted.

8.4.2 An inductive model of software design

The inductive model developed in this thesis presents a rich, holistic picture of software design. It considers a wide range of factors that act and interact upon the design process and in particular factors not specifically determined by the problem task, such as personal, organisational and cultural factors that crucially influence design strategies. The model is inductive in two senses (a) it is derived from an inductive technique of data analysis (Strauss and Corbin's (1990) paradigm model) and (b) it facilitates further inductive studies of design practice. The model is descriptive rather than prescriptive, it seeks to document how design is done, rather than to advocate how it should be done. It is flexible, adaptive and robust, applicable to many different design environments. Finally the model is explicitly stated, through explication of the Meta-process.

8.4.3 An explication of the Meta-process.

A stated objective of this thesis is to set out the process through which the data was collected and analysed and through which the model and theory were developed. This has been done most obviously in Chapter Four (Research Design) but also throughout the "results" chapters (Chapters Five and Six). The purpose of such explication is that other researchers may understand how the outcomes of this grounded theory study were reached and in so doing, be better able to replicate this study, or develop a similar one,

elsewhere[40]. The success or otherwise of this feature of the thesis is evaluated in the next section.

8.5 Evaluation of the work

Strauss and Corbin (1990:252-257) lay out two sets of criteria for judging a grounded theory study. The first set (of seven criteria) relates to the research process, the second set (also of seven criteria) relates to "the empirical grounding of the study". The purpose of these criteria is to allow readers and other researchers to assess the adequacy of the research but they also permit some degree of self-evaluation and it is very much with this in mind that they are presented here. To avoid duplication, and to save space, each set of criteria is presented as a table wherein each criterion is stated as a question or a series of questions and then a response is made based upon relevant work carried out in this research. After the tables have been presented a retrospective on the overall thesis process and outcomes is given.

[40] Since completing this research I have become aware of more, and better, grounded theory studies in the field of IS (for example Pandit, 1996; and in particular the work of Cathy Urquhart (Urquhart, 1996, 1997) within which the grounded theory generation process is more explicitly set out.

8.5.1 Evaluation of the research process

Criterion 1: How was the original sample selected? What grounds?	Interviews based on purposive/judgement/ snowballing sampling [Data-set A – 4.2.2] and theoretical sampling [Data-set B – 4.2.3]
Criterion 2: What major categories emerged?	6 meta categories derived from 27 thematic categories, derived from 160 concepts [4.3.1-4.3.3, Appendix 5]
Criterion 3: What were some of the events, incidents, actions, and so on (as indicators) that pointed to some of these major categories?	Too numerous to identify here but process set out in 4.3.2, 4.3.3 and 4.3.4 and discussed in Ch 5 and Ch 6.
Criterion 4: On the basis of what categories did theoretical sampling proceed? How did theoretical formulations guide some of the data collection? After the theoretical sampling was done, how representative did these categories prove to be?	As described above for Data-set A [4.2.3] and Data-set B [4.3.2] but also theoretical sensitivity in analysis of Data-set A [4.3.2.1]
Criterion 5: What were some of the hypotheses pertaining to conceptual relations (among categories), and on what grounds were they formulated and tested?	Major relationship was context complexity (level) – action/interaction (level) [4.3.4 and Ch 6]. Formulated using literature then validated in the data, including second data set (Data-set B).
Criterion 6: Were there instances when hypotheses did not hold up against what was actually seen? How were these discrepancies accounted for? How did they effect the hypotheses?	Negative data did not automatically invalidate a hypothesis, but may have added richness and depth to it. However weak evidence in the data lead to reject of hypothesis or qualification of it. [4.3.3 and 4.3.4]
Criterion 7: How and why was the core category selected? Was this collection, sudden or gradual, difficult or easy? On what grounds were the final analytical decisions made?	See 4.3.4 and Ch 6 but also 4.3.2 and 4.3.3 and Ch 5. Thus decision was not sudden. However was difficult due to number of candidate categories. Decision made on best fit / explanatory basis.

Table 20: Evaluation of the research process

8.5.2 Evaluation of the empirical grounding of the research

Criterion 1: Are concepts generated?	Yes [Appendices 3, 6, 7, 8] from transcript sources. Mostly technical (in vivo) codes but some common sense concepts subsequently grounded in data
Criterion 2: Are the concepts systematically related?	Yes, as per procedure in Ch4, and woven into main text (Ch5 and Ch6). Categories validated in Ch7(but not relationships)
Criterion 3: Are there many conceptual linkages and are the categories well developed? Do they have conceptual density?	Yes, through use of paradigm model, theoretical framework and conditional matrix [Ch 5, Ch6]
Criterion 4: Is much variation built into the theory?	Yes, Examples given in Ch4 where categories amended to tolerate variation or process
Criterion 5: Are the broader conditions that effect the phenomenon under study built into its explanation?	Yes, but in a specific context where impact is directly observable rather than as a general discussion of background factors [Ch6]
Criterion 6: Has process been taken into account?	Yes, for example action→interaction, causes→context-complexity.
Criterion 7: Do the theoretical findings seem significant and to what extent?	Inductive model as rich description of phenomenon informing practice. Theoretical framework as means to identify and explain design breakdowns

Table 21: Evaluation of the empirical grounding of the research

8.5.3 Evaluation of the Meta-process

In relation to the last criterion in Table 21, Strauss and Corbin (1990:257) point out that it is possible to have a good process but a poor outcome, as in design itself. This was recognised early in this thesis, indeed it was one of the motivating influences that informed the research (Chapter One, 1.4.3; 1.6). No particular claim was made to have explicated the Meta-process better than other researchers (Chapter 2, 2.3) but rather a more limited objective was set. A contribution to the literature was claimed in making this an explicit goal, at the outset, and in accounting for the success or otherwise of the attempt at the conclusion of the thesis. This has been done (8.4.3) but ultimately its value must be judged by others. Returning to Strauss and Corbin's point on the balance of outcomes and process, it is argued that this thesis does not "pay lip service" to grounded theory, nor is the method used as a "bumper sticker" of convenience, nor is the presentation of the method "imprecise" (Bryman and Burgess, 1994:6). Again this is left to the judgement of others. An outstanding concern is the extent to which meeting this objective has impacted upon the range and quality of the outcomes presented.

8.6 Recommendations for further research

These are organised into three categories, each progressively more distant from the evaluation that has just been presented. Firstly those recommendations that derive directly from limitations in this research are presented. Then, some recommendations are made that would enhance such a study, were it to be repeated, or a study similar to it undertaken. The first two sets of recommendations are internal and primarily concerned with process. The third set of recommendations is externally and outcomes focused, suggesting the application of this research to other design domains, and the forward integration of the work into the domain of methods.

8.6.1 Further development of the model and theory using qualitative methods

This thesis gives a broad description of the software design process, its scope reflecting the nature of the phenomenon and of the method. (Actually it is a reflection of the interaction of phenomenon and method and a variety of other factors including the researcher's prior knowledge of software design, his experience in the use of the method and time available.) Whilst it has been argued that such an approach is valuable, neither phenomenon nor methodology (nor the interpretation of this thesis) precludes more focused studies. On the contrary, the phenomenon and the methodology invite such studies and the thesis facilitates further research by providing foundation and direction. Any of the major categories identified and discussed in this thesis admits further investigation. For example, the development of the categories of **Balancing** and **Design-Pulls** promises further insights into design practice.

One way in which the existing model and theory may be corroborated is through the use of Multidimensional Scaling. Multidimensional Scaling (MDS) was developed in the behavioural and social sciences to study the mental model people use to understand objects and people and is used in education, geography and marketing (Butterfield,1998:35). MDS identifies similarities and differences between various entities and can calculate and display the psychological distance between them. Thus it can make visible the constructs / dimensions that individuals use to understand particular entities. The method proceeds as follows

(a) a list of common elements is derived from the textbooks

(b) the list is examined and judgements of similarity and differences made (each element is compared to all the others)

(c) a set of paired comparisons of each combination of elements is generated and listed in random order.

(d) the list is then distributed to subjects to grade

(e) responses are averaged for each comparison

(f) a similarity matrix is produced

(e) the data is analysed in two, three and four dimensions (using Kruskal method)

Use of such a technique would address, in one way, a significant shortcoming of the work presented here – validation of the theory and model by practitioners. Notwithstanding the evaluation reported in section 8.5 of this chapter, the important distinction to be made between theory development and theory testing, and the practical constraints attendant upon any research effort, this remains a fruitful avenue of further research.

8.6.2 Further development of the model and theory using quantitative analysis

Chapter Four (4.4.2.1.2) suggested that the clustering of concepts and categories inherent in a grounded theory study facilitated further quantitative as well as qualitative analysis. In Chapter Seven some limited quantitative analysis of the data was presented. In the context of mixed methods approaches advocated by Cresswell (1994) Boyatzis (1998) and others, other researchers may proceed to perform more sophisticated non-parametric tests on this, or similar data, using for example factor and cluster analysis. Those wishing to carry out such tests on data arising from their own research would do well to design this in at the beginning, since the applicability and value of such tests are crucially dependent upon the range and quality of data collected.

8.6.3 Inter-disciplinary comparisons

Inter or cross-disciplinary comparisons could be made using the model developed in this thesis as a template. Comparing software design with other design disciplines should tell us something about the current state of software development and highlight areas for further research. When one discipline is held as a paradigm to another, we may expect weaknesses in the subject discipline to be exposed. This was the case, for example, when VLSI was compared with manufacturing engineering. Mechanical and electrical mechanical engineering was found to lack sufficient abstraction and formalism to permit successful technology transfer. Specific weaknesses were identified and targeted as research recommendations[41].

The choice of a comparator discipline is clearly significant to the outcomes. Boehm (1979) compared software engineering with hardware engineering and concluded that it was in "a very primitive state" and that "one can seriously question whether our current techniques deserve to be called software engineering". A comparison with other types of engineering - mechanical and electro-mechanical for example- may have been, and may still be, kinder. Engineering is not a monolith, nor is design; within each there are a number of paradigms in operation at any one time. For example, mechanical engineering deals with a very wide range of products (from paper clips to aeroplanes) and design methods, design techniques and fabrication methods vary enormously. How much more varied are the methods and techniques used across a range of engineering or design disciplines?

---

[41] see the proceedings of the workshop on New Paradigms for Manufacturing, Arlington, Virginia, May 2-4 1994, sponsored by The National Science Foundation.

In this regard some illuminating comparisons of software design with other design disciplines have been conducted, for example Vertelney, Arent, Liberian (1991) with graphic design; Lootsma and Rijken (1998) and Rijken (1999) with architecture. A small but related piece of work arising out of the research reported in this thesis, compared software design with civil engineering, specifically on the core category of **context-complexity – action/interaction**, and suggested significant differences in approach, determined by context[42].

8.6.4 Method evaluation

A number of authors have identified the value of models to facilitate method evaluation and / or integration (Jayaratna, 1994, Brinkkemper, 1996; Punter and Lemmen, 1996; Kitchenham and Jones, 1997; Saeki, 1998). A contribution to method evaluation may be made by listing important elements of the inductive model – communication, action/interaction, context influences – and then comparing a method, or number of methods against this list. By determining the level of support offered in a particular method for software design as practised, rather than as theorised, a manager may be better able to discriminate between methods and to make an appropriate selection. Moreover a method developer may be better informed of those crucial "features" to include in any proposed new method for this domain. A fuller discussion of method evaluation and integration, in the context of Multimedia design, can be found in Gallagher and Webb, (2000).

---

[42] This research was conducted early in the project lifecycle and also acted as a pilot for the main study (see Chapter Four, 4.4.1.6)

8.7 Conclusion

A difficulty with the theory, and the underpinning paradigm model, is that deterministic accounts of complex phenomenon are notoriously difficult to present. Any such account is of course a model, a simplification of what is, or may be, happening in the real world. It is difficult to unravel cause and effect and to account for this empirically. Faced with the enormity of the task in the study of a phenomenon of any size, the researcher is forced to select one task over another. The arrangement of concepts and categories in transactional relationships presented in this thesis is thus one particular interpretation of the data. The essential difficulty of reduction is common to all deterministic theory. It is not that there is no answer, rather there are too many answers and we are forced to choose between them. In retrospect a final contribution of this thesis may be as a reminder, in the manner of good historical accounts, that reality is always much more complex than any single representation of it.

Appendix 1: Glossary

Action / Interaction: Strategies devised to manage, handle, carry out, respond to a phenomenon under a specific set of perceived conditions.

Axial coding: A set of procedures whereby data are put back together in new ways after open coding, by making connections between categories. This is done by utilising a coding paradigm involving conditions, context, action/ interactional strategies and consequences.

Category: A classification of concepts. This classification is discovered when concepts are compared one against another and appear to pertain to a similar phenomenon. Thus the concepts are grounded under a higher order, more abstract concepts called a category.

Causal Conditions: Events, incidents, happenings that lead to the occurrence or development of a phenomenon.

Code Notes: Memos containing the actual products of the three types of coding, such as conceptual labels, paradigm features, and indications of process.

Coding: The process of analysing data.

Concepts: Conceptual labels placed on discrete happenings, events, and other instances of phenomena.

Conditional Matrix: An analytical aid, a diagram, useful for considering the wide range of conditions and consequences related to the phenomenon under study. The matrix enables the analyst to both distinguish and link levels of conditions and consequences.

Conditional Path: The tracking of an event, incident or happening from action / interaction through the various conditional and consequential levels, and vice versa, in order to directly link them to a phenomenon.

Consequences: Outcomes or results of action and interaction.

Context: The specific set of properties that pertain to a phenomenon; that is, the locations of events or incidents pertaining to the phenomenon along a dimensional range. Context represents the particular set of conditions within which the action / interactional strategies are taken.

Core Category: The central phenomenon around which all the other categories are integrated.

Dimensionalising: The process of breaking a property down into its dimensions.

Dimensions: Location of properties along a continuum.

Discriminate Sampling: Associated with selective coding. Its aim is to maximise opportunities for verifying the story line and relationships between categories and filling in poorly developed categories.

Interaction: People doing things together or with respect to one another – and the accompanying action, talk and thought processes.

Intervening conditions: The structural conditions bearing on action / interactional strategies that pertain to a phenomenon. They facilitate or constrain the strategies taken within a specific context.

Memos: Written records of analysis related to the formulation of theory.

Open coding: The process of breaking down, examining, comparing, conceptualising, and categorising data.

Operational notes: memos containing directions to yourself and team members regarding sampling, questions, possible comparisons, leads to follow up on, and so forth.

Phenomenon: The central idea, event, happening, incident about which a set of actions or interactions are directed at managing, handling, or to which the set of actions is related.

Process: the linking of action / interaction sequences

Properties: Attributes or characteristics pertaining to a category

.

Selective coding: The process of selecting the core category, systematically relating it to other categories, validating those relationships, and filling in categories that need further refinement and development.

Story Line: The conceptualisation of the story: This is the core category

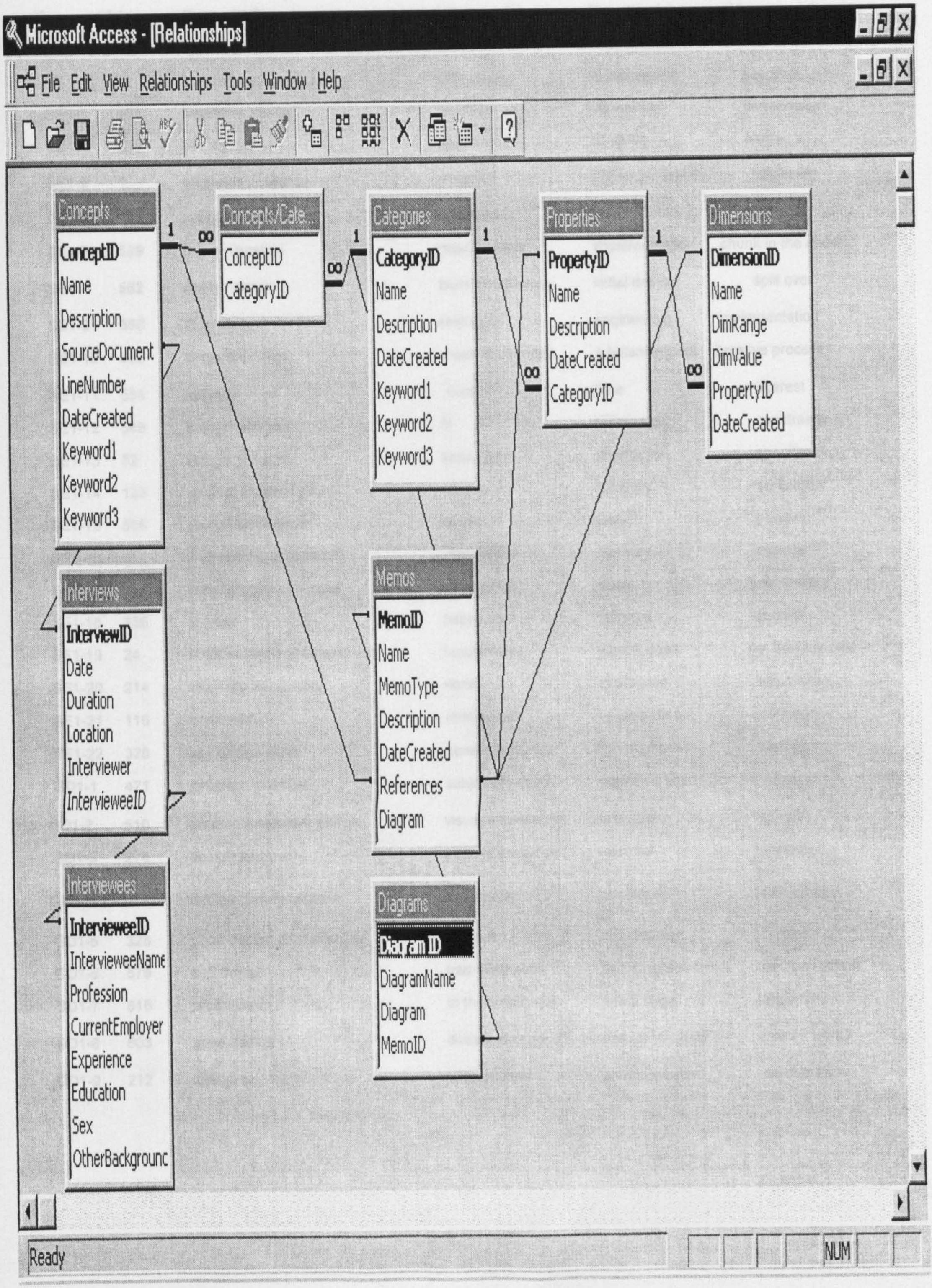
Story: A descriptive narrative about the central phenomenon of the study

Theoretical Notes: Theoretically sensitising and summarising memos. These contain the products of inductive or deductive thinking about relevant and potentially relevant categories, their properties, dimensions, relationships, variations, processes, and conditional matrix.

Theoretical Sampling: Sampling on the basis of concepts that have *proven theoretical relevance* to the evolving theory. Where *proven theoretical relevance* indicates that concepts are deemed to be significant because they are repeatedly present or notably absent when comparing incident after incident, and are of sufficient importance to be give the status of categories.

Transaction System: A system of analysis that examines action / interaction in relationship to their conditions and consequences.

Appendix 2: Data Management Structure



Appendix 3: Concepts by source (Data-set A)

Source / Line		Concept Name	Keyword 1	Keyword 2	Keyword 3
SE1-1	91	living with tech constraints	processor	user	implementation
SE1-2	92	constraint awareness	technology	management	key skills
SE1-3	682	balancing design	prioritise	separation	correctness
SE1-4	672	bad design	seduction	usability	barrier
SE1-5	?	elegance in design	elegance	understandability	balancing
SE1-6	657	design goals	updatable	maintainable	portable
SE1-7	589	design location	requirements	implementation	chunk in the middle
SE1-8	562	design deadline	blurs boundaries	initial design	spill over
SE1-9	566	algorithms	aesthetics	engineering	implementation
SE1-10	552	deign definition	meeting user req.	a balancing act	iterative process
SE1-11	554	balance	cost	time	interest
SE1-12	546	design definition	fit	technology	constraints
SE1-13	52	design process	separation	abstraction	specification
SE1-14	133	respect for complexity	seduction	arbitrary	screen-full
SE1-15	508	design constraints	space	time	power
SE1-16	511	overcoming constraints	processor	memory	chance
SE1-17	407	separation of concerns	abstraction	state	complexity
SE1-18	336	re-use	haphazard	informal	mental
SE1-19	24	function and implementation	functionality	what it does	not how it works
SE1-20	214	interface complexity	richer	interaction	separation
SE1-21	116	complexity	abstraction	appreciation	confidence
SE1-22	328	key design skills	generalisation	thoroughness	debugging
GD1-1	471	evaluating design	subjective good	objective good	function
GD1-2	510	graphic v software design	visual engineering	prejudices	stimuli
GD1-3	274	design approach	back of envelope	informal	doodling
GD1-4	417	design communication	aesthetics	publishing	balancing
GD1-5	328	good design can be subtle	obvious	non-obvious	crucial
GD1-6	579	bad design	bad aesthetics	bad functionality	bad navigation
GD1-7	616	prototypes	in the small	in the large	advanced
GD1-8	603	good design	does it look good?	does it interest?	does it work?
GD1-9	212	design approach	user centred	communicate	co-operation

Source / Line		Concept Name	Keyword 1	Keyword 2	Keyword 3
GD1-10	258	design constraints	dilution	budget	technical
GD1-11	301	design entropy	individual changes	info exchange	mutual awareness
GD1-12	534	design entropy	new entrants	computer games	graphics
SE2-1	226	physical design	platforms	problems	multi-tasking
SE2-2	695	bad design	non-intuitive	clutter	overt complexity
SE2-3	679	good design	simplicity	elegance	innovation
SE2-4	600	importance of design	innovation	simple solution	critical
SE2-5	590	design definition	shape	how	representation
SE2-6	582	design definition	development	management	marriage
SE2-7	538	design process	pragmatism	practice	what works
SE2-8	343	basic design process	specification	design	implementation
SE2-9	197	design process	Yourdon	outline design	four stages
SE2-10	355	semi-formal approach	OO diagrams	structure views	pseudocode
GD2-1	268	design influences	self-knowing	travel	interaction
GD2-2	108	role of designer	communicator	facilitator	sub-contractor
GD2-3	520	design definition	Herbert Simon	painful process	medical analog
GD2-4	581	bad design	not functional	ineffective	uncommunicative
GD2-5	573	design analogy	driving a car	answer the prob.	within resources
GD2-6	560	good design	sustainable	be self critical	be socially aware
GD2-7	209	designer as individual	artist	agent	problem solver
GD2-8	540	design definition	design for use	for users	social-functional
GD2-9	158	definition of graphic design	>aesthetics	>typography	visual literacy
GD2-10	508	design constraints	size	users	resources(budget)
GD2-11	473	graphic designers role	aesthetics	content	fine detail
GD2-12	409	more than problem solving	impress	marketing	repeat business
GD2-13	403	design essentials (nothing key)	client satisfaction	attention to detail	time management
GD2-14	394	design constraints	deadlines	resources	client confidence
GD2-15	337	client satisfaction	data	interpretation	solution
GD2-16	339	understand the client	wants	needs	task analysis
GD2-17	334	just do it (pragmatism)	theory	instinctive	clued-in
GD2-18	?	comparison with book design	co-dependence	stages	teamwork
GD2-19	320	design stages	research	development	production
GD2-20	308	problem solving	no eureka	wrestle with it	know that is right
GD2-21	182	definition of graphic design	visual comm.	info sculptor	not self-indulgent

Source / Line		Concept Name	Keyword 1	Keyword 2	Keyword 3
GD2-22	555	design intent	text	reading	method
GD2-23	364	user centred design	involvement	agreement	insurance
SE3-1	539	design is not delivery	delivery	coding	drawing
SE3-2	580	Engineering design	time	cost	constraints
SE3-3	572	Development	design	refinement	iterations
SE3-4	564	Design scope	prototyping	iterations	customer
SE3-5	586	Managing design	prototypes	customer lock-in	sign-off
SE3-6	546	over the wall design	pre-conditions	post-conditions	code solution
SE3-7	630	time constraint	time to sell	time to build	time to convince
SE3-8	552	hybrid design	structure	algorithm	graphic design
SE3-9	598	Good design	accountability	usability	refinement.
SE3-10	288	Constraint Driven design	re-design	constraint unaware	phy constraints
SE3-11	598	notations as communication	specifications	models	agreement
SE3-12	618	Bad design	ad-hoc	non-repeatable	untraceable
SE3-13	288	Constraint driven design	constraints first	engineering view	graphic design
GD3-1	341	notations	paperless	cognitive	prototypes
GD3-2	369	bad visual design	unconnected	divergent	pointless
GD3-3	307	bad design	dad can't use	navigation	doesn't sell
GD3-4	353	design definition	personal	subjective	no rules
GD3-5	165	difficult to change design	ownership	appreciation	attachment
GD3-6	176	cognitive design	mental model	not storyboard	modular
GD3-7	224	design motivation	stuck	boring	process
GD3-8	271	design strategy	get down to it	get on with it	do it again
GD3-9	161	compromise	creative freedom	how it looks	not how it works
GD3-10	156	design process	identification	planning	execution
GD3-11	133	graphic design	constraints	aesthetics	awareness
GD3-12	327	design definition as a process	identification	solving	execution
GD3-13	247	design process	architectural	detailed	structure
GD3-14	?	design goals	inform	access	ease-of-use
GD3-15	66	lack of integration	structure	presentation	integration
GD3-16	?	bad design	obvious	clutter	illegible
GD3-17	393	design influences	market	user	resources
GD3-18	399	good design	coherent	solid	innovative
GD3-19	405	compromise	function	form	structure

Source / Line		Concept Name	Keyword 1	Keyword 2	Keyword 3
GD3-20	?	design goals	inform	orient	visual appeal
SE4-1	692	bad design	cutting corners	unnatural	not related to tasks
SE4-2	640	design importance	multimedia	software	hardware
SE4-3	614	design definition	*user-centred*	*fine art*	usability
SE4-4	632	design tools	CASE	self-help	support
SE4-5	643	black box design (process)	isolate	delegate	automate
SE4-6	655	design notations	diagrams	visualisation	re-use
SE4-7	?	design goals	ease of use	happy customer	interest
SE4-8	678	good design	usability	subjective	doesn't suck
SE4-9	577	estimation	time	effort	doodling
SE4-10	714	design principles	efficiency	ease of dev	ease of maint.
SE4-11	719	design goals	simplicity	quality	elegance
SE4-12	625	design definition	human factors	task analysis	close to user
SE4-13	708	design principles	user centred	easier	nicer
GD4-1	483	design experience	*personality*	*blueprints*	diversify
GD4-2	27	*design as problem solving*	graphic design	structure	visual com.
GD4-3	55	*design as problem solving*	decomposition	information	hierarchy
GD4-4	84	design as problem solving	navigation	text	media
GD4-5	414	design process of refinement	elimination	gestation	refinement
GD4-6	455	design as functionalism	aesthetics	ergonomics	constraints
GD4-7	480	originality	non-mechanistic	thinking laterally	what-if factor
GD4-8	520	influences on design	personality	time	money
GD4-9	442	design is everything	fit for purpose	aesthetics	ergonomics
GD4-10	535	design quality	communicate	help	*happiness*
GD4-11	466	good design	*fit for purpose*	design intention	Soho analogy
SE5-1	370	*design definition*	start with nothing	iterations	customer
SE5-2	450	good design	solid	useable	easy to learn
SE5-3	382	scope of design	volatile	uncertain	prototype
SE5-4	437	design goals	works	solid	innovative
SE5-5	421	design notations	pseudo code	flowcharts	drawing boxes
SE5-6	4	design process	chaotic	in-flux	spanner in works
GD5-1	375	design constraints	rigid	accountancy	battery hen
GD5-2	499	design presentation	access	palatable	general
GD5-3	468	design constraints	the client	critique	lack of awareness

Source / Line		Concept Name	Keyword 1	Keyword 2	Keyword 3
GD5-4	358	design inputs	software engineer	fine artist	boundaries
GD5-5	100	de-motivation	frustrated	not in charge	pull the reigns in
GD5-6	34	good design	concrete	purpose	not airy-fairy
GD5-7	530	design influences	everything	cinema	travel
GD5-8	477	design definition	a science	a technician	problem solving
SE6-1	59	in-house methodology	pragmatic	informal	consistent
SE6-2	492	design as problem solving	problem-solving	communication	understanding
SE6-3	76	design process	prototyping	informal	iterative
SE6-4	142	functionality first	functionality	user-interface	presentation
SE6-5	692	good design documentation	simplicity	concise	unambiguous
SE6-6	679	design process	specification	prototyping	reviews
SE6-7	660	good design (activity)	careful	thorough	consultation
SE6-8	635	notations	doodling	pseudo-code	specification
SE6-9	565	design as planning	link	storyboard	definition
SE6-10	545	design as a roadmap	plan	approach	detail
GD6-1	173	design intent	desire	end-user	want
GD6-2	179	design influences	trends	fashion	inspiration
GD6-3	165	design is everything	multimedia	development	look&feel
SE7-1	170	design definition	structure	pathways	platforms
GD7-1	310	design as navigation	storyboarding	routing	linking
GD7-2	342	need for design language	multimedia	crashing along	fast pace
GD7-3	387	good design as purpose	brief	obvious	seamless
GD7-4	402	end user design	environment	presentation	know the end user
SE8-1	694	re-use	code	library	object-oriented
SE8-2	672	design process	separation	notations	implementation
SE8-3	661	design scope	re-design	Internet	woolly
GD8-1	20	graphic design	visual com.	aesthetic art	multidisciplinary

Appendix 4: Sample Memos

MemoID References	Memo Name	MemoType	Memo Description	Date
Mo-0001-00	Design contradiction?	Operational	Between these code notes quite different views on design are expressed. What is the explanation of this? Genuine confusion on the part of the interviewee, carelessness on the part of the interviewer, or my interpretation?	19/02/98
Mo-0002-00	Design is not delivery	Theoretical	This view of design as everything that is not delivery is one I have first come across from an interviewee with a product designer (Kelly in Winograd-Software Design). Yet here it is being expressed by some-one interviewed as software engineer.	20/02/98
Mo-0003-00	Engineering Design	Theoretical	Note, here he is not talking about engineering design per se but the importance of having a good design in allowing software to be delivered on time and within constraints. Really, he's saying design is all important	20/02/98
Mo-0004-00	design context - constraint	code	Here the SE is constrained by the GD design. This the SE acknowledges as good but it will not lend itself to efficient coding - the GD design won't permit a grid solution and therefore the SE's ability to deliver a good SE design is curtailed.	21/04/98
Mo-0005-00	design strategy	code	The SE's aim is to produce an efficient implementation of the GD design but there are problems (see constraints). The SE wants to be able to CUT IT DOWN, DIVIDE UP THE ELEMENTS and PROGRAM IT BETTER. But cannot do this well without the GD changing the design	20/04/98
Mo-0006-00	design consequences	code	There are a number of consequences here. First the SE cannot produce an efficient AS A RESULT OF the initial GD design. Second, when the SE suggests the GD changes the design the GD 'goes spare'. Third, rather than change the original design to accommodate the SE's requirement. The GD throws out the entire design and starts again.	20/04/98
Mo-0007-00	design as implementation	code	Here, in contrast to the earlier SE(see M0-0002-00) design and implementation merge. This said in context of Web design. Is it generally true? For Se's ? for	20/04/98
Mo-0008-00	design context	code	a formal (on paper) specification from the customer which changes	20/04/98

Appendix 5: Thematic and Meta categories (Data-set A)

5A: Thematic categories

Functionality: What the design does (or should do) ideally set out in a specification and agreed with users [SE6-10]. Maps to criteria for good design [GD1-1; SE2-2]. More stressed by SEs [eg. SE6-4] although also by some GDs [eg. GD??]. Other refs [SE1-19] See also STRUCTURE, PRESENTATION, SEPARATION, ABSTRACTION, DE-COMPOSITION, BALANCE, CONSTRAINTS, ENTROPY

Structure: How the design works. For some purely an implementation issue (code and interface) [SE-19]. Also Informational structure [GD2-10], metaphor of magazine [GD1-7]; navigation [GD4-2]; storyboardying [GD2-10] Design is the shape [SE2-5] Poor structure → bad design [GD2-3] However all structure + poor presentation → bad design [GD3-13]; See also FUNCTIONALITY, PRESENTATION, SEPARATION, ABSTRACTION, DE-COMPOSITION, BALANCE, CONSTRAINTS, ENTROPY

Presentation: How the design looks (and feels) but more than aesthetics. Primarily a GD concern [eg. GD2-11; GD3-15; GD5-2]. Some SE - GDs are too concerned with presentation, move to it too early and do not sufficiently consider functionality [SE1-19; SE6-4]. However becoming more difficult to separate the two due to complexity of interface [SE1-20]. See also STRUCTURE, PRESENTATION, SEPARATION, ABSTRACTION, DE-COMPOSITION, BALANCE, CONSTRAINTS, ENTROPY. COMPLEXITY

Entropy: The name given to the phenomenon whereby qualities and characteristics associated with one discipline are passed on to, or acquired by, the other. Most evidence for this comes from one respondent [GD1] but also found in GD3 and implicit in comments by SE3 and SE4. One consequence of such entropy is that one discipline understands and appreciates more what the other does. [GD1-5; GD1-11; GD1-12; SE3-8; GD3-11; SE4-12]

Problem Solving/Framing: (Problem Solving) -Frequently used to describe design process. Explicitly [GD2-12; GD4-3; GD5-8; SE6-2] or implicitly [GD2-20; GD3-12]. Designer is a problem solver [GD4-2] See also ABSTRACTION, SEPARATION, DE-COMPOSITION. **(Problem Framing)** - Composite category used by researcher to capture observed nature of design process. Characterised by number and frequency of ITERATIONS, level of interaction and COMMUNICATION, use of PROTOTYPES and STORYBOARDS

User-requirements: Users also referred to as Clients and Customers. Evidence here and elsewhere suggests that the term Client is a more appropriate description. Main story is difficulties caused by users not knowing what they want, or changing their minds [ ] and generally of ill defined and shifting requirements [ ]; This results in a complex and volatile design context in which outcomes are difficult to predict. Clients can also be a design CONSTRAINT [GD2-23]. For user centred methods see [SE4-3; SE4-12; SE4-13; GD6-1; GD7-1] See also PROTOTYPES, COMMUNICATION, ITERATION, METHODS

Designer-Motivation: That which *leads to or causes* a designer to undertake a design, and once undertaken, to continue to do it, and to do it well. The absence or diminishment of which causes a design not to be undertaken, to be abandoned once undertaken, or to be done poorly. [eg GD5-1]. Motivations include Technology [ ]; Aesthetics [ ]; Complexity [ ]; - note these are also PULLS but here designer is pro-active in engagement. Also includes Having Fun [ ]; Getting it done [ ]; having a Happy Customer [SE4-7]. See also INFLUENCES. [GD1-4; GD2-12; SE4-7; GD6-1]

Designer-Influences: That which act to *determine or shape* the nature of the design approach or its outcome. Differs from MOTIVATION in that they do not include those factors that *cause* a design but obviously some overlap in terms of impact on outcomes. Include Books, Cinema, Travel, The Market, Experience, Personality. [GD5-7; GD4-1; GD4-8, GD2-1; GD3-17; GD4-1; GD4-8; GD5-7]

Design-Pulls: Term used to describe observed phenomenon whereby designers are lulled into a design effort. Differs from MOTIVATION in that here not a conscious action to engage a design but semi or unconscious action *driven by the design itself.* 3 "pulls" identified – Technology, Complexity, Aesthetics. Identified positive and negative consequences of this. [SE1-4; GD2-21; GD3-2] See also GOOD DESIGN, BAD DESIGN, MOTIVATION, INFLUENCES.

Context –Complexity: Found at two levels (1) low level / explicit as it relates to some specific aspect of the design eg. Technical complexity, interface complexity (2) high level / implicit as a Property of a general condition (causal, contextual, intervening) that makes the design process more difficult eg. Requirements. Management. At both levels act to determine or influence design strategy and outcomes. [SE1-14; SE1-17; SE1-20; SE2-2] "Contra-context" of much lower complexity also observed.

Notations: Generic term used to classify range of approaches to documenting and communicating a design. Includes Doodling, Sketching, Flowcharting. Wide range of approaches within and between disciplines from "not writing anything down" [GD3-1] to writing a lot [GD1-3, SE4-6] to more formal approaches [SE3-11]

Prototypes: Identified both as an approach to design (prototyping) and as an artefact or tool used in the design process. Prototypes produced at different levels depending on nature of design problem / project / environment (including in-house policy) [GD1-7]. Prototypes are an important means of communicating with clients [SE5-3] and of getting agreement on the design. See also ITERATION, STORYBOARDS.

Abstraction: Important cognitive / conceptual process identified usually along with SEPARATION and DE-COMPOSITION. Made more difficult by increased complexity at interface. Need to move from abstract layer to "something concrete" as soon as possible [SE3-7] See also REFINEMENT, PROBLEM SOLVING. [SE1-13; SE1-21; GD3-6]

Separation: The separation of concerns eg functionality from presentation, what from the how, design from implementation, physical from logical design. Identified primarily by SEs as good practice. Made more difficult (and more important) by complexity. Can be difficult but at low (interface) level can routine and boring [GD3-8]. SE claim better at this than GD [SE3-6] but also clear evidence of GD doing this [GD2-11] [SE1-13; SE1-17; SE1-19; SE2-8; GD2-11, SE3-6, GD3-8.SE1-3; SE2-1] See also ABSTRACTION, REFINEMENT, PROBLEM SOLVING

Decomposition: Process of breaking down (problem/design) into more manageable units. Differs from SEPARATION in that it implies hierarchy (top down approach) but process is non linear – does not proceed in even step manner but in practice involves ITERATIONS of de-composition and re-composition. Facilitates (and is facilitated by) informational and navigational STRUCTURE. See also REFINEMENT, PROTOTYPING [SE1-3; SE1-13; GD2-11]

Refinement: aka Elimination, Gestation, Evolution [GD4-6]– the process (strategy) of progressively improving a design. Also known as Trial and Error. Should be limited [SE5-2] otherwise → BAD DESIGN. May result in difficulty in estimating time and costs [SE4-9]. See also ITERATION, PROTOTYPING, PROBLEM SOLVING.

Re-use: Broad category used to define multiple types of re-use – code, notations, methods, tools, techniques, the design itself. Identified as an indicator of good design (poor or low re-use → bad design). However too much re-use is a bad thing. Amount of re-use is determined by nature of problem and design environment. [SE1-18; GD3-8; SE4-6; GD8-1; SE8-3; SE4-4.] See also ITERATION, REFINEMENT, PROTOTYPES

Storyboards: Traditionally used by GDs (and much evidence here) but also by SE's [eg SE2-8; SE3-11; SE6-9; SE7-1]. As with Prototypes important communication aids, used to present ideas, get commitments, get agreement. Also as a means of structuring information, linking everything together [SE6-6], as a navigational tool [GD7-1] and a planning tool [GD3-6]

Iteration: Evidence of iteration within and between stages, between designers and users, between designer and materials of design. Generally viewed as a positive but too much is seen as a sign of bad design "poor communication" [SE5-2] with negative impact on schedule and cost. See also PROTOTYPES, METHODS.

Pragmatism: An approach to design characterised by (inter alia) a desire to "get on with it" [ ] "Just get down and do it" [GD2-18], a sceptical view of theory [GD2-17; SE6-2], rather "rely on what works" [SE2-7]. Also evidenced in comments about good and bad design – "concrete solid" [GD3-18]; "does it work?" [SE5-4]; "concrete" [GD5-6]

Communication: Expressed as a purpose (or intended outcome) of design and as a design activity (communicating with designers and users). Predominantly a concern of GDs but also identified by SE's [SE5-2; SE6-2]. (Good, clear, unambiguous) communication is a criterion of GOOD DESIGN. See also BAD DESIGN, PRESENTATION. Importance of ITERATION, PROTOTYPES AND STORYBOARDS in facilitating communication. [GD1-4; GD1-6; GD1-9; GD2-4; GD3-15, GD8-1]

Collaboration: subsumes NEGOTIATION and COMPROMISE as sub-categories. Closely allied to COMMUNICATION [GD2-2; SE2-8; GD2-11]. Different types and levels. Explicit reference to teams [GD5-4; SE5-3]. Importance to design outcomes emphasised [SE6-7]

Balancing: Action taken to manage design constraints. Recognised as a key design skill. Antecedent or cause to good design. Includes time, cost, and elements of design itself – form and function [GD3-18], functionality and interface [GD1-12]. Also balancing management and design [SE2-6]. [SE1-5; SE1-10; GD1-4; GD1-8] Also includes balancing different approaches (SE and GD). See also CONSTRAINTS, FUNCTIONALITY, STRUCTURE, PRESENTATION, ENTROPY

Good Design: Qualitative descriptions (what it is). Applied to outcome (design artefact) or process of design. Expressed as quality of good designer. Defined by discipline eg SE (robust,); GD (user friendly) but also agreement (functional). Subjective so lacks agreed definition but many common recurring elements. Number of concepts in this category reflects direct question in interview (What is design?) [SE1-5; GD1-1; GD1-5; GD1-8; SE2-3; GD2-6; GD2-8; SE3-9; GD2-8; SE3-9; GD3-4; GD3-18; SE4-8; GD4-10; GD4-11; SE5-2; GD5-6; SE6-7; GD7-3] See also CONSTRAINTS, BALANCING,

Bad Design: Defined largely as negatives of good design. Lack of direct concepts reflect this and the question asked. [SE1-4; GD1-6; SE2-2; GD3-2; GD3-13] See also CONSTRAINTS, BALANCING

Design-Constraints: Those factors (technical and non technical, personal, organisational, social, cultural) that act to constrain a design. Includes STRUCTURE, PRESENTATION AND FUNCTIONALITY. Closely allied to BALANCING [SE1-1; SE1-2; SE1-10; SE1-13; SE1-16; GD2-7; GD2-10; GD2-14; SE3-2; GD3-11; GD3-17; SE4-9; GD4-6; GD5-1; GD5-3; GD5-4. See also BALANCING

Methods: Refers to overall approach to design rather than specific methods. Approaches range from the very informal "back of the envelope" [GD1-3], "chaotic" [SE5-6], "ad-hoc" [ ], to more formal approaches "based on Yourdon's techniques" [SE2-9, SE6-3]. Generally however approaches tend to be informal reflecting the nature of the design effort and the organisations involved. Includes User centred Methods. See also Prototyping, Storyboarding.

5B Meta categories

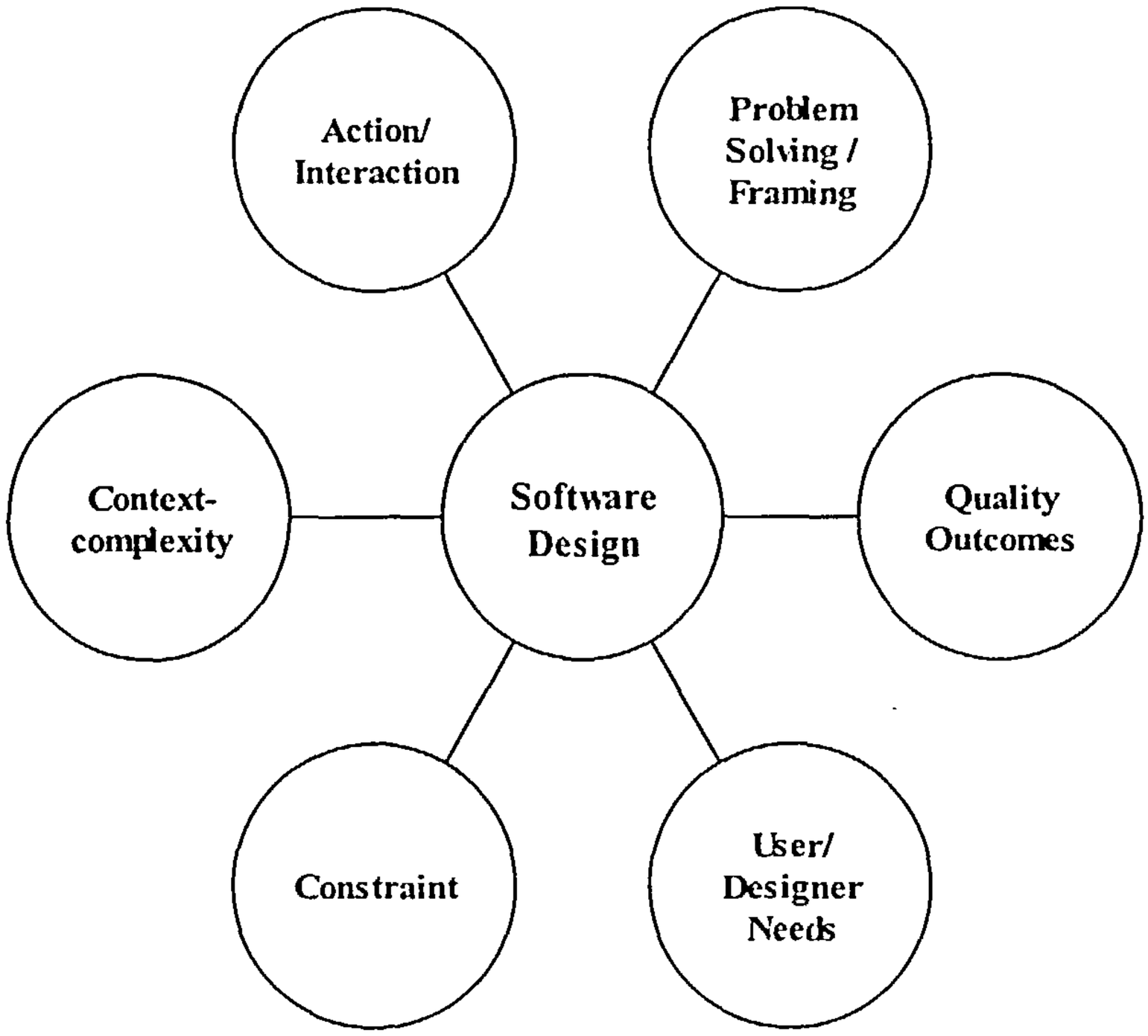


Figure 12: Meta Categories

Appendix 6: Concepts by source (Data-set B)

Source / Line	Concept Name	Source / Line	Concept Name
SE1-1 / 18	Complexity v experience	SE5-9 / 80	Re-use
SE1-2 / 20	greater theoretical complexity	SE5-10	Common backgrounds
SE2-1 / 27	a shallow discipline	SE6-1 / 95	complexity
SE2-3 / 30	not understanding complexity	SE6-2 / 99	over / under structuring
SE2-4 / 31	Re-Use of code	SE6-3 / 100	hiding complexity
SE2-5 / 34	seduced by the computer	SE6-4 / 101	user driven complexity
SE3-1 / 44	Re-use	SE6-5 / 102	fascination with complexity
SE3-2 / 46	better technical environment	SE6-6 / 102	hiding complexity form end
SE3-3 / 47	Re-use of experience		
SE3-4 / 52	approach to programming	SE6-7 / 103	the lure of complexity
SE3-5 / 52	De-composition	SE6-8 / 103	simple programs are harder
SE4-1 / 58	problem solving -	SE6-9 / 104	Re-use of company products
SE4-2 / 59	approach to programming /com	SE6-10 / 104	comparison with other
SE4-3 / 61	approach to programming	SE6-11 / 106	taming complexity / flying anal
SE4-4 / 62	re-use	SE6-12 / 107	holding the abstraction
SE4-5 / 63	Approach to programming	SE6-13 / 108	approach to programming
SE4-6 / 64	approach to programming	SE6-14 / 108	fools rush in
SE5-1 / 73	approach to programming	SE6-15 / 108	approach to programming
SE5-2 / 74	problem of maintenance	SE7-1 / 119	approach to programming
SE5-3 / 74	re-use : experience	SE7-2 / 120	joy of programming
SE5-4 / 75	complexity to simplicity	SE7-3 / 0	sculpting analogy
SE5-5 / 76	approach to programming	SE7-4 / 125	bridge building
SE5-6 / 77	approach to programming	SE7-5 / 129	detective analogy
SE5-7 / 77	abstraction	SE8-1 / 157	breaking the rules
SE5-8 / 0	simplicity	SE8-2 / 157	analogy with writing
		SE8-3 / 158	approach to prog.

SE8-4 / 158	greater complexity		
SE8-5 / 160	understanding		
SE9-1 / 167	bottom up programming		
SE9-2 / 169	re-use		
SE9-3/ 169	experience		
SE9-4/ 170	Re-use		
SE9-5 / 170	Re-use		
SE9-6 / 171	Approach to computing		
SE9-7 / 172	Re-use		
SE10-1 / 176	Approach to prog.		
SE10-2 / 176	Re-use		
SE10-3 / 177	Managing complexity		
SE10-4 / 178	Programmers profile		
SE10-5 / 188	profile		
SE11-1 / 192	Prog. analogy		
SE11-2 / 195	Approach to prog		
SE11-3 / 196	memory		
SE11-4 / 202	Problem solving		
SE12-1 / 214	Approach to programming		
SE12-2 / 214	simplicity		
SE12-3 / 215	Re-use		
SE12-4 / 217	Origin of techniques		
SE12-5 / 217	abstraction		
SE12-6 / 218	Approach to prog.		
SE12-7 / 223	Component re-use		
SE12- 8 / 251	analogy		
GD1-1 / 280	analogy		
GD1- 2 / 281	Pull of the computer		
GD1-3 / 284	Pull / abstraction		
GD2-1 / 289	abstraction		

Appendix 7: Concepts by Category (Data-set A)

Functionality		SE1-19	GD1-1	GD1-2	GD2-4	GD2-11	SE3-1	SE3-2
		SE3-6	GD3-19	GD4-6	GD4-9	GD5-6	SE6-4	GD1-4
		SE6-10	GD4-11					
Structure		SE1-6	SE2-5	GD2-11	SE3-8	GD3-13	GD3-15	GD3-18
		GD3-19	GD4-3	SE6-10	SE7-1	SE2-1	SE6-4	
Presentation		SE1-19	SE1-20	GD1-2	GD1-8	SE2-5	GD2-9	GD2-11
		GD3-2	GD3-9	GD3-14	GD3-15	GD3-19	GD4-2	GD4-9
		GD5-2	GD7-4	GD8-1	GD1-1	GD2-16	SE4-3	SE6-3
Entropy		GD1-2	GD1-9	GD1-11	GD1-12	SE2-6	SE3-8	GD3-11
Problem Solving		SE1-7	SE1-8	SE1-18	SE1-22	SE2-6	SE2-9	GD2-5
/ Framing		GD2-7	GD2-12	GD2-15	GD2-20	GD3-10	GD3-12	GD4-2
		GD4-4	GD4-5	GD5-8	SE6-2	GD12-2	GD3-8	GD2-20
		SE4-4						
User-requirements		SE1-4	SE1-10	GD1-9	GD2-10	GD2-13	GD2-14	GD2-15
		GD2-16	GD2-23	SE3-4	SE3-5	GD3-9	SE4-3	SE4-8
		SE4-12	GD4-1-	SE5-1	SE5-3	GD5-3	GD7-4	
Designer-motivation		GD1-8	GD2-12	GD3-4	GD3-7	GD4-1	GD4-7	GD4-8
		SE5-1	SE5-4	GD5-5	GD6-1	GD5-1	GD2-7	
Designer-influences		SE1-16	GD2-1	GD2-3	GD2-6	GD2-7	GD2-8	GD2-12
		GD2-21	GD3-5	GD3-17	SE4-3	SE4-7	GD4-1	GD4-8
		GD5-7	GD5-4	GD6-2	SE3-2			
Designer-pulls		SE1-9	SE1-4	SE1-14	GD1-4	GD2-9	GD2-21	GD3-11
		SE4-3	GD4-6	SE2-2	SE4-11	GD3-2	GD2-10	GD6-13
		SE2-2	GD2-7					
Context-complexity		SE1-14	SE1-17	SE1-20	SE1-21	SE2-1	SE2-2	GD3-5
		GD7-2	SE3-3	SE4-9	SE1-14	SE1-21	GD1-11	
Notations		GD1-3	GD1-7	SE2-10	GD2-22	SE3-1	SE3-11	GD3-1
		GD3-6	SE4-6	SE5-5	SE6-5	SE6-8	SE8-1	SE8-2
		GD2-23	SE1-18	SE4-9				
Prototypes		SE3-4	SE3-5	GD3-1	SE5-3	SE6-3	SE6-6	GD1-7
		GD7-4	SE6-6	GD2-23	SE3-4			
Abstraction		SE1-12	SE1-13	SE1-17	GD3-6	SE1-3	SE1-20	
Separation		SE1-12	SE1-13	SE1-17	SE1-22	SE8-2		
Decomposition		GD3-13	GD4-3	GD3-7				
Refinement		SE1-10	GD1-7	GD2-13	SE3-3	SE3-9	GD4-5	SE8-3
		SE4-9	SE5-3	SE8-3	GD3-12	SE7-1	SE1-8	SE2-9
		GD3-10						

Re-use		SE1-18	GD3-8	SE4-6	SE4-10	GD4-1	SE8-1	SE8-3
		SE8-2	SE1-22	SE3-12	GD2-20			
Storyboards		SE3-11	GD3-6	SE6-9	GD7-1	SE7-1		
Iteration		GD2-1	SE3-3	SE3-4	GD3-5	SE5-1	SE4-9	GD4-5
Pragmatism		SE2-7	GD2-17	GD3-8	SE6-1			
Communication		GD1-4	GD1-9	GD1-11	GD2-2	GD2-4	GD2-21	GD2-23
		GD3-19	SE4-5	GD4-10	SE6-2	SE6-1	SE3-11	GD7-4
		SE6-6	GD2-23	SE3-4	GD7-7	GD5-2		
Collaboration		GD2-2	GD2-18	GD3-19	SE4-5	SE2-8	GD2-11	GD5-4
		SE6-7						
Balancing		SE1-3	SE1-5	SE1-11	GD1-4	GD2-13	GD1-10	GD2-7
		SE1-21	SE3-10	GD5-5	GD3-10	GD2-11	GD3-9	GD2-10
Good design		SE1-5	SE1-6	SE1-22	GD1-1	GD1-5	GDI-8	SE2-3
		SE2-4	GD2-6	GD2-15	SE3-9	GD3-14	GD3-18	SE4-7
		GD3-18	SE4-7	SE4-8	SE4-10	SE4-11	SE4-13	GD4-10
		GD4-11	SE5-2	SE5-4	SE6-7	GD7-3	SE4-13	SE3-13
		GD3-3	GD3-16					
Bad design		SE1-14	GD1-2	GD1-6	SE2-2	GD2-4	SE3-10	SE3-12
		GD3-2	GD3-3	GD3-16	GD3-19	SE4-8	SE3-13	SE4-1
		GD6-13						
Design-constraints		SE1-1	SE1-2	SE1-3	SE1-5	SE1-8	SE1-10	SE1-11
		SE1-15	SE1-16	GD1-10	SE2-1	GD2-5	GD2-10	GD2-14
		SE3-2	SE3-7	SE3-10	SE3-13	GD3-11	GD3-17	SE4-9
		GD4-6	GD5-7	GD5-3	SE4-9	GD1-3	SE1-21	GD5-1
Methods		SE1-18	GD1-3	SE2-7	SE2-9	SE2-10	GD2-19	GD2-20
		GD2-22	SE3-6	GD3-4	GD3-10	GD3-13	SE4-4	SE4-5
		SE5-6	SE6-1	GD7-2	SE5-6			

Appendix 8: Concepts by Category (Data-set B)

Problem Solving		SE2-5	SE3-4	SE4-1	SE4-3	SE4-5	SE5-6	SE6-2
/ Framing		SE6-12	SE6-13	SE11-1	SE11-4			
Designer-motivation		SE2-1	SE3-5	SE5-1	SE5-4	SE6-3	SE6-10	SE7-5
		SE8-5	SE10-4					
Designer-influences		SE2-1	SE2-3	SE3-3	SE3-5	SE4-4	SE5-3	SE5-10
		SE6-1	SE6-10	SE6-15	SE8-1	SE8-5	SE9-7	SE10-4
		SE11-4	SE12-4	SE12-6	SE12-8			
Designer-pulls		SE2-5	SE3-5	SE4-6	SE5-1	SE5-4	SE6-1	SE6-7
		SE6-15	SE7-2	SE8-4	SE11-1	SE12-8	GD1-2	GD1-3
Context-complexity		SE1-1	SE1-2	SE2-1	SE2-3	SE2-5	SE3-5	SE4-5
		SE4-6	SE5-2	SE5-8	SE6-1	SE6-3	SE6-4	SE6-5
		SE6-6	SE6-7	SE6-11	SE10-3	GD1-3		
Notations		SE4-2	SE4-3	SE5-5	SE12-12	GD1-1		
Prototypes		SE3-4	SE7-3	SE8-3	SE9-1			
Abstraction		SE3-4	SE4-2	SE5-1	SE5-7	SE5-8	SE6-12	SE7-4
		SE12-5	GD1-1	GD2-1				
Decomposition		SE3-4	SE3-5	SE4-1	SE4-2	SE4-5	SE4-6	SE5-1
		SE6-13	SE7-1	SE11-2	SE12-5			
Refinement		SE2-4	SE3-4	SE7-3	SE7-5	SE8-3	SE9-1	SE9-4
		SE9-6	SE11-2	SE11-4				
Re-use		SE1-1	SE2-4	SE3-1	SE3-3	SE4-4	SE5-3	SE5-8
		SE5-9	SE5-10	SE6-9	SE6-10	SE9-2	SE9-3	SE9-5
		SE9-7	SE10-2	SE11-3	SE12-3	SE12-4	SE12-7	
Iteration		SE4-2	SE5-5	SE11-2				
Pragmatism		SE11-4						
Good design		SE5-10						
Bad design		SE5-10	SE6-14					
Design-constraints		SE3-2						
Methods		SE3-4	SE4-2	SE4-3	SE5-5	SE6-13	SE8-2	

References

Adelson, B., (1989) Cognitive research: uncovering how designers design, Research in Engineering Design, 1 (1) 33-42.

Agar, M.H. (1980) The Professional Stranger: An informal introduction to ethnography. London: Academic Press

Agresti, William, W. (1986) New Paradigm for Software Development, IEEE Computer Society Press.

Ambron, S. (1990) "What is Multimedia? in S Ambron and K Hooper (ed.), Interactive Multimedia, Microsoft Press, 5 –9.

Ancona, D. (1990) Outward Bound; Strategies for Team Survival in an Organisation, Academy of Management Journal (33:2) June 1990, 334-365.

Aoyana, Mikio (1998) Web-based Agile Software Development, IEEE Software, Nov-Dec 1998, Vol 15, N06, 56-65.

Archer, S. (1988) Qualitative research and the epistemological problems of the management disciplines, in A. Pettigrew (ed.), Competitivenes and the Management Process, Oxford: Basil Blackwell: 265-302.

Babbie, E. (1995) The Practice of Social Research, Seventh Edition, Washington, USA; Wadsworth Publishing Company.
Bandinelli S., Fuggetta. A., Lavazza, L., Loi, M., and Picco, G. P. (1995) Modeling and Improving an Industrial Software Process. IEEE Transactions on Software Engineering, 21, 5 (May), 440-454.

Banville, C; Landry, M. (1989) Can the filed of MIS be disciplined? Communications of the ACM, 32, 1, January, pp. 48-60.

Barry, C. A. (1998) Choosing Qualitative Data Analysis Software: Atlas/ti and Nudist Compared, Sociological Research Online, vol.3, no.3.

Bateson, G. (1973) Steps to an Ecology of Mind, Granada, London.

Bechofer, F. (1974) Current approaches to empirical research: some central ideas, in J. Rex (ed) Approaches to Sociology: an introduction to major trends in British Sociology, London, Routledge and Kegan Paul.

Becker, H. (1986) Writing for social scientists. Chicago: University of Chicago Press. Beverly Hills, CA: Sage

Beyer, Hugh and Holtzbalt, Karen (1998) Contextual Design: Defining Customer-Centred Systems, Morgan Kaufmann Publishers, INC. San Francisco, California.

Bhaskar, R. (1975) A Realist Theory of Science. Leeds: Leeds Books.

Black, T. B. (1993) Evaluating Social Science Research: An Introduction, London, Sage.
Blum, Bruce I, (1992) Software Engineering: a holistic view, New York, Oxford University Press.

Blum, Bruce I, (1996) Beyond programming: to a new era of design, Oxford University Press

Boehm B. (1979) Software Engineering: R&D Trends and Defence Needs, in Peter Wegner, ed. Research Directions in Software Technology, MIT Press, Cambridge, Mass., 44-86.

Boyatzis, R. E. (1998) Transforming Qualitative Information; Thematic Analysis and Code Development, Thousand Oaks, CA. Sage.

Brinkkemper, S. (1996) Method Engineering: Engineering of Information Systems Development Methods and Tools, Information and Software Technology, 38 (4): 275 – 280.

Brooks, Frederick P. Jr, (1975) The Mythical Man-Month, Essays on Software Engineering, Reading, MA: Addison-Wesley, 1975 (1995 Anniversary Edition).

Brooks, Frederick P. Jr (1986) No Silver Bullet: Essence and Accidents of Software Engineering, Information Processing, Elsevier Science Publishers, North Holland, 1069-1076

Brown, S. L. and Eisenhardt, K. M. (1997) The Art of Continuous Change: Linking Complexity Theory and Time-paced Evolution in Relentlessly Shifting Organisations, Administrative Science Quarterly, Vol. 42, 1-34.

Brown, John, S. and Duguid, P. (1994) Borderline issues: Social and material aspects of design. Human-Computer Interaction 9:1 (Winter, 1994), 3-36.

Bryman, A. and Burgess, R.G. (1994) ed. Analysing Qualitative Data, Routledge, New York.

Budgen, D. (1994) Software Design, Addison Wesley, Wokingham, England.

Bunzel, M, and Morris, SK. (1992) Multimedia Applications Development, McGraw-Hill, New York.

Burrell G and Morgan G (1979) Sociological Paradigms and Organisational Analysis. Heinman, London.

Butterfield, Jeff (1998) The Analyst's view of complex systems projects, ISM, Winter, 1998, 34- 39

Card, S, Moran, T. and Newell, A. (1983) The Psychology of Human-Computer Interaction. Hillsdale, NJ: Lawrence Erlbaum Associates.

Carey, T.T. and Mason, R.E. A. (1985) Information System prototyping: Techniques, Tools and Methodologies, the Canadian Journal of Informational Research and Information Processing, Vol, 21, N03, 177-191.

Carroll, John (1991) ed. Designing Interaction: Psychology at the Human-Computer Interface. New York: Cambridge University Press.

Carroll, J. M. & Rosson, M. B. (1991) Deliberated evolution: stalking the view matcher in design space. Human-Computer Interaction, 6, 281-318,

Carroll, J. M., Singley, M. K., & Rosson, M. B., (1992) Integrating theory development with design evaluation. Behaviour and Information Technology, 11, 247-255.

Carroll, J. M., Alpert, S. R., Karat, J., Van Deusen, M. & Rosson, M. B. (1994) Capturing design history and rationale in multimedia narratives. In Proceedings of CHI'94; Human Factors in Computing Systems, Boston, 192-197. New York: Association for Computing Machinery/ Addison- Wesley.

Carroll, John, M. (1997) Human-computer interaction: psychology as a science of design. International Journal of Human-Computer Studies, 46, 501-522.

Charette, Robert N. (1986) Software Engineering Environments: Concepts and Technology, McGraw Hill Inc. New York.

Charmaz, K. (1983) The grounded theory method: An explication and interpretation. In R. Emerson (ed.), Contemporary field research (pp. 109 –126), Boston: Little, Brown

Checkland, P. (1981) Systems Thinking Systems practice, John Wiley and Sons.
Churchman, C. W. (1971) The design of Inquiring Systems, New York, Basic Books, Inc.

Coffey, A. and Atkinson, P. (1996) Making sense of qualitative data. Thousand Oaks, CA: Sage.

Colgan, L. and Spence, R., (1991) Cognitive modelling of electronic design, in J. Gero (Ed.) AI in Design 91, Butterworth-Heinemann, Oxford.

Conklin, J. and Yakemovic, K.C.B. (1991) A process-oriented approach to design rationale. Human-Computer Interaction, 6, 357-391.

Cotton B. and Oliver R. (1993) Understanding Hypermedia: From Multimedia To Virtual Reality, London: Phaidon Press

Craig. Patricia A. (1991) 'A Graphic Designer's Perspective' in Karat, J. (Ed.) Taking Software Design Seriously: Practical Issues for Human-Computer Interaction Design, Academic Press Inc. San Diego.

Creswell, John W. (1994) Research Design: Qualitative & Quantitative Approaches, Sage Publications Inc. Thousand Oaks, Ca.

Cross, N; Christianns, H and Dorst, K. (1996) eds. Analysing Design Activity, Chichester, John Wiley and Sons.

Csikszentmihalyi, M. (1988) Motivation and Creativity: Towards a synthesis of structural and energistic approaches to cognition. New Ideas in Psychology, Vol 6, 159 – 176.

Dahl, D. J; Dijkstra, E, W; Hoare, C.A.R. (1972) Structured Programming, New York, Academic Press.

Dasgupta, S. (1991) Design Theory and Computer Science, Cambridge Tracts in Theoretical Computer Science 15, Cambridge University Press, Cambridge.

Davies, L. J. and Myers, M. D. (1994) Scholarship and practice: the contribution of ethnographic research methods to bridging the gap, in Glasson et al (ed.) Business Process Re-Engineering: Information Systems Research in the 1990's, Amsterdam, Elsevier North Holland.

Davies, L. J. (1991) Researching the organisational culture contexts of Information Systems Strategy in HE, in Nissen et al (ed.) Information Systems Research in the 1990's, Amsterdam, Elsevier North Holland.

Davies, S. and Castell, A., (1992) Contextualising design: narratives and rationalisation in empirical studies of software design, Design Studies, 13 (4), 379-392.

De Marco, T and Lister T. (1990) (eds.) Software Engineering State of the Art, Selected Papers, Dorser House Publishing, New York.

Denzin, N. K. (1978) The research act: A theoretical introduction to sociological methods (2nd Ed.) New York: McGraw- Hill.

Dospisil, G. and Polgar, T. (1994) Methodology for reducing the complexity of large hypermedia projects, IFIP Transactions A – Computer Science and Technology: 243-255.

Dym, Clive (1995) "Peeling the Design Onion" IEEE Spectrum, June 1995, pp 10-12

Eastman, C.M., (1970) On the analysis of intuitive design processes, in G. T. Moore (Ed.) Emerging Methods in Environmental Design and Planning, MIT Press, Cambridge. MA.

Ehrlenspiel, K. and Dylla, K., (1993) Experimental investigation of designers' thinking methods and design procedures, Journal of Engineering Design, 4 (3), 201 – 212.

Eisenhardt, K. (1989) Building theories from case study research. Academy of Management Review 14 (4), 532-550.

Elsbach, K. D. and Sutton, R.I. (1992) Acquiring Organisational Legitimacy through Illegitimate Actions: A Marriage of Institutional and Impression Management Theories, Academy of Management Journal (35:4) December 1992, 699 –738.

Engestrom, Y. (1993) Developmental work research: reconstructing expertise through expansive learning. In M. Nurminen & G. Weir, Eds. Human Jobs and Computer Interfaces. Amsterdam: Elsevier.

Ennis, C. W. and Gyeszly, S. W., (1991) Protocol analysis of the Engineering Design process, Research in Engineering Design, 3 (1) 15-22.

Ericsson, Anders, A. and Simon, Herbert, H. (1984) Protocol Analysis; Verbal reports as data. The MIT Press, Cambridge, Mass

.

Erlandsen, D. A. Harris, E. L. Skipper, B. L. and Allen, S. D. (1993) Doing naturalistic inquiry: A guide to methods. Newbury Park, CA: Sage

Farhoomand, A.F. (1987) Scientific progress of management information systems, Data Base, 18, 4, Summer, pp 48-56

Feigenbaum, Edward A. (1989) What Hath Simon Wrought? In Klahr, David and Kotovsky, Kenneth (eds.) 21st Carnegie-Mellon Symposium on Cognition. Complex Information Processing: the Impact of Herbert A. Simon, Lawrence Erlbaum Associates, New Jersey. pp 165-182

Fisher, S. (1994) Multimedia Authoring: Building and Developing Documents, Academic Press, London.

Fox, M. (1983) Constraint-Directed Search: A Case Study of Job-Shop Scheduling. Phd thesis. Carnegie-Mellon University. CMU-RI-TR-83-22

Flynn, D. (1992) Information systems requirements: determination and analysis, McGraw-Hill, London.

Galal, H. (1996) An Interpretative Approach to Information Systems Engineering, unpublished Phd. Thesis, Department of Computer Science, Brunel University, UK.

Gallagher S and Webb B (1997) Competing paradigms in Multimedia systems development: who shall be the aristocracy. In Proceedings of the Fifth European Conference on Information Systems, Galliers et al, ed. Cork Publishing, Cork, Ireland, 1113 - 1120.

Gallagher S (1999) In search of paradigms in multimedia systems design: an analysis of software engineering and graphic design approaches using the Kuhnian model, unpublished PhD. Thesis, The Queen's University of Belfast, Northern Ireland.

Gallagher, S. and Webb B. (2000) Paradigmatic analysis as a means of eliciting knowledge to assist multimedia methodological development, European Journal of Information Systems, 9, 60-71.

Garzotto, F., Mainetti, L., and Paolini, P. (1995) Hypdermedia Design Analysis, and Evaluation Issues, Communications of the ACM, 38 (8), 74—86.

Gersick, C. J. G. (1994) Pacing Strategic Change: The Case of a New Venture: Academy of Management Journal, 37 (1), 9-45.

Giola, D. A. and Chittipedi, K. (1991) Sensemaking and Sensegiving in Strategic Change Initiation, Strategic Management Journal, 12, 433-448.

Glaser, B. (1978) Theoretical sensitivity. Mill Valley, CA: Sociology Press.

Glaser and Strauss (1965) Awareness of dying. Chicago: Aldine.

Glaser, B.and Strauss A. (1967) The discovery of Grounded Theory: Strategies for Qualitative Research, Wedenfeld and Nicholson, London.

Glass, R. L. and Conger, S. A. (1992) Research software tasks: Intellectual or clerical? Information and Management, 23, 4, pp 183 – 192.

Glass, Robert L. (1996) "Is there anything 'Time honoured' in the field of software?" the Data Base For Advances in information Systems, 27 (3) Summer, 1996, 16-18.

Glass, Robert L. (1999) "The Loyal Opposition on Design" IEEE Software, Vol 16, N0 2, March/April, 1999,103- 104

Goedicke, M. (1990) Paradigms of Modular Systems Development in Mitchell, R. J. (ed.) Managing Complexity in Software Engineering, Peter Peregrinus Ltd. London, 1-20.

Goel, V., and Pirolli, P. (1989) Motivating the notion of generic design within information processing theory; the design problem space. AI Magazine, 10 (1): 18-36.

Goel, V. and Pirolli, P., (1992) The structure of design problem spaces, Cognitive Sciences, 16, 395-429.

Goetz, J. P. and LeCompte, M. D. (1984) Ethnography and qualitative design in educational research. New York: Academic Press

Gordon, Ian, E. (1989) Theories of Visual Perception, John Wiley, Chichester, UK

Greenberg, S. (1991) Computer-Supported-Cooperative Work and Groupware: an introduction to the special issues, International Journal of Man-Machine studies, 34, 2, 133-141.

Guindon, R., (1990) Knowledge exploited by experts during software system design, International Journal of Man-Machine Studies, 33, 279 – 304.

Guindon, R; Krasner, H; Curtis, B; (1990) Breakdowns and Process During the Early Activities of Software Design by Professionals in De Marco, T and Lister T; eds. Software State of the Art, Selected Papers, Dorset House Publishing, New York, 455 – 476

Hammer, Mike (1995) Reengineering the corporation: a manifesto for business revolution Rev. ed. - London : Brealey

Hartmanis, J. (1989) "Godel, von Neuman and the NP problem", Eatcs Bulletin 38, 101-107, quoted in Bovet, D and Crescenzi, P (eds) Introduction to the theory of complexity, Prentice Hall International, 1994, page IX.

Helander, Martin (1988) ed. Handbook of Human-Computer Interaction: New York: North-Holland.

Hinrichs, Thomas, R. (1992) Problem Solving in Open Worlds; A Case Study in Design. Lawrence Erlbaum Associates, INC. New Jersey.

Hirschheim, R.A. (1985) Information Systems Epistemology: An Historical Perspective in E. Mumford et al ed. Research Methods in Information Systems, North Holland, Amsterdam, 13-36.

Hirschheim, R. A. and Klein, H.K. (1989) Four paradigms of information systems development, Communications of the ACM, 32, 10, 1199 - 1217

Hirschheim, R. A. and Newman, M. (1991) Symbolism and Information Systems Development: Myth, Metaphor and Magic, Information Systems Research, 2, 1, 29-62.

Howard K. and Sharp J.A. (1983) The management of a student research project. Gower Publishing University Press Cambridge.

Humphrey W. (1989) Managing the software process. - (The SEI series in Software Engineering). Addison-Wesley, Reading, Mass.

Hunt, J. G. and Ropo, A (1995) Multi-level Leadership: Grounded Theory and Mainstream Theory Applied to the Case of General Motors, Leadership Quarterly, 6, 3, 379 – 412.

Iivari, J. (1991) An analysis of contemporary schools of IS development European Journal of Information Systems, Vol 1, No 4, 249-272

Isabella, L. A. (1990) Evolving interpretations as a change unfolds: How managers construe key organisational events, Academy of Management Journal, 33:1, March 1990, pp. 7-41

Israel Ben-Shaul. (1995) .A Paradigm for Decentralised Process Modeling. Kluwer Academic Publishers, Norwell, Mass.

Jaccheri, M. L., Picco, G. P. and Lago, P. (1998) Eliciting software process models with the E(3) Language. ACM Transactions on Software Engineering and Methodology, Vol 7, No4, October, 1998, 368-410.

Jacques, Elliot (1991) Executive leadership: a practical guide to managing complexity, . Casson Hall, Cambridge Mass.

Jayaratna, N (1994) Understanding and Evaluating Methodologies: NIMSAD – A Systemic Framework. McGraw-Hill, London.

Jick, T. D. (1979) Mixing qualitative and quantitative methods: triangulation in action. Administrative Science Quarterly, 24, 602-611.

Judd, C.M; Smith, E. R; and Kidder, L.H. (1991) Research Methods in Social Relations, Holt, Rinehardt and Winston, 6th Edition,

Kahn, W. A. (1990) Psychological conditions of personal engagement and disengagement at work, Academy of Management Journal, 33:4, December, 1990, 692 – 724.

Kapor, Mitchell (1991) A software design manifesto: Time for change. Dr Dobb's Journal 172 (January 1991), 62-68

Kickert W. J. M.and van Gigch J. P. (1979) A metasystem approach to organisational decision making. Management Science, 25, 1217-1231.

Kitchenham, B. A. and Jones, L. (1997) Evaluating software Engineering Methods and Tools, Software Engineering Notes, 22 (parts 1-5).

Konechi, Krzysztof (1997) Time in the Recruiting Search process by Headhunting Companies, in Strauss, A. S. and Corbin Juliet, M, ed. Grounded Theory in Practice, Sage Publications,131-146

Kuhn, Thomas S. (1970) The Structure of Scientific Revolutions, 2nd edition, University Press, Chicago.

Kuhn, Thomas, S. (1977) 'Second Thoughts on Paradigms', in TS. Kuhn (Ed.) The Essential Tension, University Press, Chicago, 298-319.

Lammers, S.; Programmers At Work: (1989) Interviews with 19 Programmers Who Shaped the Computer Industry, Redmond, Wash: Microsoft.

Latchem, C., Williamson, J., and Henderson-Lancett, L (1993) Interactive Multimedia: Practice and Promise, Kogan-Page, London

Lawrence, Peter. (1981) Software Design: Methods and Techniques, Yourdon.

Lawson, B. (1997) How designers think: the design process demystified, Architectural Press, Oxford.

Lee, A (1999) Researching MIS, in Rethinking Management Information Systems: An Interdisciplinary Perspective, ed. Wendy l. Currie and Bob Galliers, Oxford University Press, 7 – 27.

Le Moigne. J. L. (1985) Towards new epistemological foundations for Information Systems Systems Research, 2, 247-251.

Lehman M, (1980) programs, Life Cycles and Laws of Software Evolution, Proceedings of the IEEE, Vol. 68. No 9, September, 1060 – 1076.

Lehman, M., and Belady, L. (1971) "Programming system dynamics", given at the ACM SIGOPS Third Symposium on Operation System Principles, October, 1971 (quoted in Brooks, 1975).

Lincoln, Y. S. and Guba, E. G. (1985) Naturalistic inquiry. Beverly Hills, CA: Sage.

Lloyd, P. and Scott, P., (1994) Discovering the design problem, Design Studies, 15 (2), 125 – 140.

Lofland, J. (1971) Analysing social settings: A guide to quantitative observation and analysis, Belmont California; Wadsworth.

Lofland, J. (1974) Analysing Qualitative data: first person accounts, Urban Life and Culture 3 (3): 307-309

Longren and Stolterman (1999) Design Methodology and Design Practice, Interactions, Jan/Feb, 1999, 13-20

Lootsma, B and Rijken, D. (1998) Media and Architecture. VPRO and the Berlage Institute, Amsterdam.

Luther, A. (1994) Authoring Interactive Multimedia, London: Academic Press.

MacLean, A.,Young, R. M., Belloti, V.M.E. & Moran, T. P. (1991) Questions, options and criteria: elements of design space analysis. Human-Computer Interaction, 6, 201-250.

Malhotra, A. , Thomas, J., Carroll, J., & Miller, L. (1980) Cognitive processes in design. International Journal of Man Machine Studies, 12:119-140.

Mallon, B and Webb B. (2000) Structure, causality, visibility and interaction: propositions for evaluating engagement in narrative multimedia, International Journal of Human-Computer Studies, 53, 269-287.

Marr, D (1982) Vision: A Computational Investigation into the Human Representation and Processing of Visual Information, W.H. Freeman, San Francisco

McDermot, J. (1980) R1: An expert in the computer systems domain. In Proceedings of the First National Conference on Artificial Intelligence, 269-271.

Mehrens, W. A. and Lehmann, I. J. (1984) Measurement and Evaluation in Education and Psychology, 3rd edn. Holt-Saunders.

Mellon, C. A. (1990) Naturalistic Inquiry for Library Science: Methods and Applications for Research, Evaluation and Teaching. Contributions in Librarianship and Information Science, N0 64. Greenwood, New York.

Merriman, S. B. (1988) Case study research in education: A qualitative approach. San Francisco: Jossey-Bass

Mesarovic M. D., Macko D. and Takahara, Y. (1970) Theory of Hierarchical, Multilevel Systems, Academic Press, New York.

Miles, M. B., & Huberman, A. M. (1984) Qualitative data analysis: A sourcebook of new methods. Beverly Hills; London, Sage.

Miller L. and Crabtree B.F. (1992) eds, Doing qualitative research , Newbury Park, Calif.; London, Sage Publications.

Minneman, S. and Leifer, L., (1993) Group engineering design practice: the social construction of a technical reality, Proceedings of the International Conference on Engineering Design (ICED93) ed. N. Roozenburg, The Hague, Heurista, Zurich.

Moody, Fred. (1995) I Sing the Body Electronic: A year with Microsoft on the Multimedia Frontier, Hodder and Stoughton, London.

Morrison, J. and George, J. F. (1995) Exploring the Software Engineering Component in MIS Research, Communications of the ACM, 38, 6, 80-91.

Mulhauser M., and Effelsberg, W. (1996) ed. Proceedings of MMSD'96 International Workshop on Multimedia Software Development, Berlin, Germany, IEEE Computer Society Press: pvii.

Myers, Ware. (1999) Early communication key to software project success. IEEE Computer, May 1999, Vol. 32; Number 5, 110-111.

Nachmias, D. and Nachmias, C. (1976) Research Methods in the Social Sciences, Edward Arnold (Publishers) Ltd. London.

Nardi, B. A. (1995) Studying context: A comparison of activity theory, situated action models and distributed cognition. In B. A. Nardi, Ed. Context and Consciousness: Activity Theory and Human- Computer Interaction, 69-102. Cambridge, MA: MIT Press.

Naur, Peter. (1985) Programs as Theory Building, in Peter Naur, Computing: A Human Activity, New York; ACM Press, 37-48.

Neilsen, J. (1993) Usability Engineering, Academic Press, London.

Newell, Allen (1969) Heuristic programming: Ill-structured problems. In J. Aronofsky (Ed.) Progress in operations research, Vol. 3, 361-414. New York, Wiley.

Newell, Allen and Simon, Herbert, A. (1972) Human Problem Solving. Prentice Hall, Inc, Englewood Cliffs, New Jersey

Ng, Peter, A and Yeh, Raymond, T. (1990) eds. Modern Software Engineering – Foundations and Current Perspectives, Van Nostrand Reinhold, New York

.

Norman, Donald and Stephen Draper, (1986) User Centered System Design: New Perspectives on Human-Computer Interaction. Hillsdale, NJ: Lawrence Erlbaum Associates.

O'Hear A. (1989) Introduction to the philosophy of science. – Oxford; Clarendon Press.

Olson, G. M., et al, (1992) Small group design meetings: an analysis of collaboration, Human-Computer Interaction, 7, 347 – 374.

Orlikowski, W. J. and Robey, D. (1991) Information Technology and the Structuring of Organisations, Information Systems Research (2:2) June 1991, 143-169.

Orlikowski, W. J. (1993) CASE Tools as Organisational Change: Investigating incremental and radical changes in systems development, MIS Quarterly September 1993, 309 – 340.

Oshagbemi, T (1999) Managers and Their Time, Blackwell Publishing, Dublin.

Pandit, Maresh R. (1996) "The creation of theory: A recent application of the grounded theory method", The Qualitative Report, 2 (4), 1996.

Parnas D. L. (1971) "Information distribution aspects of design methodology" Carnegie-Mellon, Department of Computer Science, Technical report (Feb. 1971).

Parnas, D. L. (1972a) "A technique for software module specification with examples" Comm. ACM, 5, 5 (May, 1972), 330-336.

Parnas, D. L. (1972b) "On the criteria to be used in decomposing systems into modules" Comm ACM, 5, 12 (Dec., 1972), 1053-1058.

Partington, David (1997) Management Processes in Projects of Organisational Change: Case Studies from Four Industries. Unpublished PhD Thesis, Cranfield, UK.

Partington, David (1998) Building Grounded Theories of Managerial Behaviour from Interview Data, Cranfield Working Papers, SWP 18/98.

Petroski, H. (1985) To engineer is human: the role of failure in successful design, Macmillan, London

Pettigrew, A. M. (1990) Longtitudinal Field Research on Change: Theory and Practice, Organisation Science, 1:3, August 1990, 267-292

Pfleeger, Shari Lawrence. (1987) Software Engineering: The Production of Quality Software. New York: Macmillan.

Poulin, Jeffrey, S. Re-use: Been there, done that. Communications of the ACM, May, 1999, Vol 42, N05, 98-100.

Powell, T.A. (1998) Web Site Engineering: Beyond Web Page Design, London: Prentice Hall.

Punter, T. and Lemmen, K. (1996) The MEMA-model: Towards a New Approach for Method Engineering, Information and Software Technology, 38 (4): 295-305.

Pressman, R. (1982) Software Engineering: A Practitioner's Approach, Third Edition, McGraw-Hill, New York.

Raeithel, Arne (1991) Activity Theory as a Foundation for Design, in Christiane Floyd, Heinz Zullighoven, Reinhard Budde, and Reinhard Keil-Slawik, eds. Software Development and Reality Construction, Berlin: Springer-Verlag, 391-415.

Randell, B; Ringland, G; and Wulf. B; (1994) eds. Software 2000: A View of the Future. The output of a forum sponsored by ICL and the Committee of the European Communities.

Reitman, W. (1964) Human decision procedures, open constraints and the structure of ill-defined problems. In M.Shelly and G. Bryan (ed.) Human judgements and optimality, New York, Wiley, 282-315.

Richards, L and Richards, T (1991) 'The transformation of qualitative methods: computational paradigms and research processes', in N.G. Fielding and R.M. Lee (ed.) Using Computers in Qualitative Research, London, Sage.

Rijken, Dick "Information in Space (1999) Explorations in Media and Architecture" Interactions, May/June, Vol 6 N03, 44-57.

Ritchie and Spencer (1994) Quantitative data analysis for applied policy, in Bryman, A. and Burgess, R.G. ed. Analysing Qualitative Data, Routledge, New York.
pg 218.

Rittel, H. and Weber, (1973) M. Dilemmas in a general theory of planning. Policy Sciences, 4, 155-169.

Rowe, P. G. (1987) Design Thinking, MIT Press, Cambridge, Mass.

Rumbaugh, James. (1991) Object-oriented Modeling and Design. Englewood Cliffs, NJ: Prentice-Hall.

Runco, Mark (1994) Problem finding, Problem solving and Creativity. Ablex Publishing Corporation, Norwood, New Jersey.

Saeki, M. (1998) A Meta-model for Method Integration, Information and Software Technology, 39, 925-932.

Sandstrom, A. R. and Sandstrom, P. E. (1995) The use and misuse of anthropological methods in Library and Information Science Research, Volume 65, April 1995, Number 2, 161 – 199.

Schon, D. (1983) The Reflective Practitioner: How Professionals Think in Action. New York: Basic Books, 1991.

Schon, D. and Bennet, J.L. (1996) Reflective Conversation with Materials, in Winograd T. (ed.) Bringing Design to Software, ACM Press, Addison-Wesley, New York, 171-184.

Shannon, C. E., (1959) Coding theorems for a discrete source with a fidelity criterion. In IRE National Convention Record, Part 4, 142-163.

Simon, H. A. (1969) The Sciences of the Artificial. Cambridge, MA: MIT Press.

Simon, H. (1973) The Structure of ill-structured problems. Artificial Intelligence, 4: 181-201.

Simon, H.A. (1988) 'Creativity and Motivation: A response to Csikszentmihalyi'. New Ideas in Psychology, Vol 6, 177-181.

Simon, H. A. and Newell, A. (1972) Human Problem Solving, Englewood Cliffs, Prentice Hall,

Shneiderman, B. (1980) Software Psychology: Human Factors in Computer and Information Systems. Cambridge, MA: Winthrop.

Shneiderman, B. (1997) Designing Information-abundant Web Sites: Issues and Recommendations, International Journal of Human Computer Studies, 47: 5-27.

Sommerville, I (1992) Software Engineering (4th Edition) Addison-Wesley, Wokingham, England.

Spector, A. and Gifford, D. (1986) A computer science perspective on bridge design. Communications of the ACM, 29 (4) April, 1986, 268 – 283.

Stacey, Ralph (2000) Strategic management and organisational dynamics: the challenge of complexity, 3rd edition, Financial Times Prentice Hall.

Stefik, M. (1981) Planning with constraints. Artificial Intelligence, 16, 111-140.

Strauss A. (1987) Qualitative analysis for social scientists. New York: Cambridge University Press.

Strauss A. and Corbin J. (1990) Basics of Qualitative Research: Grounded Theory Procedures and Techniques. Sage Publications, London.

Strauss A. and Corbin J (1997) eds, Grounded theory in practice, London : SAGE.

Sussman G., and Steele, G. Jnr. (1980) Constraints - A language for expressing almost hierarchical descriptions. Artificial Intelligence, 14, 1-39.

Sutton, B. (1987) The rationale for qualitative research: a review of principles and theoretical foundations, The Library Quarterly, vol 63, no. 4. 411 – 430.

Tang, J.C. (1991) Findings from observational studies of collaborative work, International Journal of Man-Machine Studies, 34, 143-160.

Thomas, J.C. and Carroll, J. M. (1979) The psychological study of design, Design Studies, 1 (1), 5-11.

Ullman, D., Dietterich, T. and Stauffer, L. (1988) A model of the mechanical design process based on empirical data. Artificial Intelligence in Engineering Design and Manufacturing (AI EDAM), 2 (1), 33-52.

Urquhart, C. (1996), Strategies for conversation and systems analysis in requirements gathering: a qualitative view of analyst-client communication", The Qualitative Report, (4:1-2).

Urquhart, C. (1997) "Exploring analyst-client communication: using grounded theory techniques to investigate interaction in informal requirements gathering", in Information Systems and Qualitative Research, A. S. Lee, J. Liebenau and J. I. DeGross (eds.) Chapman and Hall, London, pp 149-181.

Van Gigch, J.P Pipino, L.L. (1986) In search of a paradigm for the discipline of information systems. Future Computer Systems, 1, 1, 71-97

Van Vilet, (1993) Software Engineering: Principles and Practice, John Wiley and Sons, Chichester, England.

Vaughan, T (1994) Multimedia Making it work, McGraw-Hill, London

Verney and Glass (1994) Application Based Methodologies, Information Systems Management, Fall, 1994, 53-57.

Vertelney, L., Arent, M. and Liberian, H. (1991) Two Disciplines in Search of an Interface, in Laurel, The Art of Human Computer Interface Design, reading, Mass., pp 45-56.

Walsham, G. (1995) 'Interpretive Case Studies in IS Research: nature and method' European Journal of Information Systems, 4:74-81.

Webb, B. R. (1996) The role of users in interactive systems design: when computers are theatre, do we want the audience to write the script?, Behaviour & Information Technology, 1996, vol. 15, No. 2, 76 – 83.

Webb, B. R. and Booth M. (1995) Approaches to usability in Multimedia systems development, in G. King (ed.) Software Quality Management III, Computational Mechanics Publications, Southampton, 387 – 397.

Weick, Karl; (1967) Systematic Observation Methods, Handbook of Social Psychology, Addison Wesley, 357-451.

Weitzman, E and Miles, M. (1995) Computer programs for qualitative data analysis: an expanded sourcebook. 2nd Edition. Thousand Oaks, California: Sage.

Wernick, Paul and Winder, Russel (1994) A Plethora of Paradigms: From Definitions of the Term Paradigm to a Philosophy for Software Engineering, UCL Research Note, RN/93/94.

Wernick, Paul (1995) A belief system model for software development: A framework by analogy, unpublished Phd thesis, University College London

Wernick, Paul and Winder, Russel (1996) Software Engineering as a Kuhnian Discipline in Winder, Probert and Beeson (eds.) Philosophical Aspects of Information Systems, UCL Press.

Wertsch, J. (1985) Vygotsky and the Social Formation of Mind. Cambridge, MA: Harvard University Press.

Whitley, R. (1984) The intellectual and social organisation of the sciences. - Oxford : Clarendon Press.

Whitefield, A. and Warren, C., (1989) A blackboard framework for modelling designers' behaviour, Design Studies, 10 (3) 179 – 187.

Wiklund, M. E. (1994) Usability in Practice, Academic Press, London

Williams, B. (1990) Invention from first principles: An overview. In P. Winston and S. Shellard (eds.) Artificial intelligence at MIT: expanding frontiers, Volume 1, pp 433-463. Cambridge, MA: MIT Press.

Winder, R and Wernick, P (1994) Refining a Philosophical Model of Software Development: Tracing Elements of the Disciplinary Matrix, in Lissoni, Richardson, Miles, Wood-Harper, Jayaratna (eds.) Information Systems Methodologies, BCS ISMSG, 203-216.

Winograd Terry, (ed.) (1996) Bringing design to software, ACM Press, New York.

Winograd, Terry and Flores, Fernando. (1987) Understanding Computers and Cognition: A New Foundation for Design. Reading, MA. Addison-Wesley.

Wirth, N. (1971) Program development by stepwise refinement, CACM, Vol 14, April 1971.

Yeh, Raymond (1990) An alternative paradigm for software evolution in Ng, Peter, A and Yeh, Raymond, T. (eds) Modern Software Engineering – Foundations and Current Perspectives, Van Nostrand Reinhold, New York, 1 – 13.

Zachery, Pascale, G. (1994) Showstopper! The breakneck race to create Windows NT and the Next Generation at Microsoft, Little, Brown and Company, London.

Zuboff, S. (1988) In the age of the smart machine: the future of work and power, Heinemann Professional, Oxford..