Chapter 1
Introduction

1.1 Research Motivation and Aims

This research is motivated by an interest in seeking an 'amethodical' approach[1] to software system development[2] (SSD). This interest stems from the author's past experience over nine years of developing Management Information Systems[3] (MISs). A MIS typically involves people from different backgrounds, with different knowledge and experience, and working in various organisational roles, and also includes their complex business processes that could be ill-structured and experience-oriented. In most cases, the developers[4] are totally unfamiliar with such working practices, and the users are not sure what they want, or even what they could have, from the system being developed. The

[1] The term 'amethodical' is used in this thesis to mean that no particular, strict method is used, rather than suggesting that no method is used. The term 'method' is used here as a synonym for 'methodology' and refers to a series of well-structured steps and procedures to be followed in the course of developing a software system [AF95].

[2] In this thesis, the term 'software system development' is preferred to the popular term 'information system development' in order to stress those information systems which are software-intensive. It refers to a process in which human agents construct and use a software system for their practices (cf. [Flo87, HKL95]). A broader perspective on SSD that cannot be completely addressed using conventional concepts of SSD is considered in this thesis. More details are elaborated in Chapter 3 (see especially p.49-50).

[3] A MIS is traditionally defined as "an integrated, computer-based, user-machine system that provides information for supporting operations and decision-making functions' [Awa88, p.5]. A modern MIS is a highly interactive information system that generates information for monitoring performance, maintaining coordination, and understanding problems and new situations [Alt96, p.223]

[4] In this thesis, the term 'developer' is applied to all people who are engaged in software system development, such as analysts, designers and programmers.

requirements for such a system are invariably unclear and volatile, in particular when the factor of reengineering the business process of users is taken into account. Accordingly, the intended system is ill-defined and open.

There are many textbook approaches for developing such a system. In the author's experience, however, most method-based approaches devised to date, from traditional waterfall methods [SS95, STM95] to the popular object-oriented methods [Boo94, CY90, Jac92], are difficult to use[5]. On the one hand, it is clear that the use of a method-based approach providing formally-defined methods, techniques and tools can in principle guide the development of software systems in a systematic and cost-effective fashion. On the other hand, such a technical process, which regards SSD as a manufacturing process within the engineering discipline and as therefore deterministic, mechanistic and rational, cannot easily adapt to a continuously changing environment [Fit96, Gog94]. This is because, in the case of an open, ill-defined system, it is very hard for developers to cope with diverse situations simply by following a set of steps and procedures imposed by a method. Instead, developers usually perform this task by sequences of what L. Suchman has called 'situated[6] actions' [Suc87], which are interactive actions undertaken by the actor in response to the situation of his/her external environment.

For example, object-oriented methods proposed in [Boo94, CY90, Jac92] require conservative assumptions about what is naturally an object. For instance, objects have a certain fixedness of roles and persistent inheritance of methods. This requires that each object must be subject to and must conform to well-defined and explicitly stated rules. In other words, the problem domain, objects and the interactions between objects become

[5] In this respect, the author's experience is in accord with the concerns of the research in [Fit96, RHHR98, And+90], where the practical difficulties of using a method-based approach for SSD are identified. It is also acknowledged that what developers do in the real world is often quite different from what method-based approaches suggest that they should do, but the discussion in this thesis does not take this into account.

[6] In [Gog96], J.A. Goguen identifies the qualities of situatedness as 'emergent, local, contingent, embodied, open and vague'. This description helps to clarify the meaning of the term 'situated'.

relatively fixed or static once they are assumed in a model. Therefore, within this approach, the real world, which is often chaotic and complex, is either ignored or expressed in a diagrammatic form, which is bound to be tangible constrained and formal. Obviously, such a mechanical approach cannot deal with contingent problems in a situated manner.

In addition, in the author's experience, the practice of interacting with users through documentation is not as effective as these traditional step-by-step methods claim. In reality, as indicated in [Fis91, DS97], interpersonal interaction through documentation is passive, error-prone and labour-intensive and has been recognised as the most difficult part of the process of SSD [Bub95, Eas93, Pot93, STM95, VPC98, Zav95]. This is because these traditional methods take it for granted that developers can collect and represent users' needs through text- and diagram-based metaphors (or even a prototype of the intended system), and that users can clearly express their needs and understand what these metaphors mean. Ironically, in such a framework, users are 'outsiders' in relation to the system being developed for them, and developers are at the centre of SSD. Figure 1-1(a) shows the relationship between developers, users, the used method, the used metaphors and the developing system in this developer-centred development[7] of software systems. Despite further improvements that have been proposed, such as user-centred design and Joint Application Design (JAD) [AF95], users are still outsiders, though they do have more opportunities to contribute to design decisions (see Figure 1-1(b)). In the context of both Figure 1-1(a) and Figure 1-1(b), developers still dictate the agenda of activities for ensuring the transition from users' needs to a software system.

[7] Strictly speaking, this developer-centred development should be regarded as method-centred, since the behaviour of each developer is restricted to following the steps dictated by the used method.

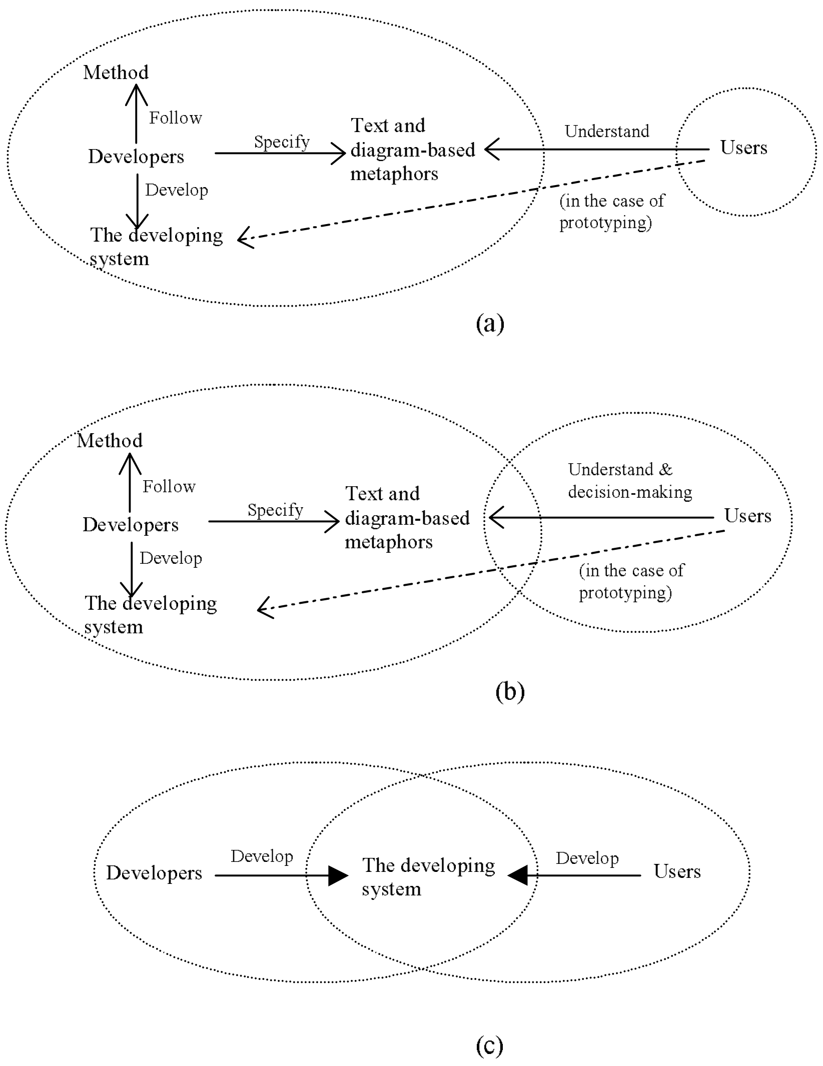

Figure 1-1.
(a) developer-centred SSD with users involved for consulting.
(b) developer-centred SSD with users involved for decision-making
(c) SSD with users involved for co-development

On the basis of this practical experience, the author has been led to ask two fundamental questions about the use of method in SSD:

- To what extent does a method really guide SSD and solve problems arising from the process of SSD?
- Is it possible to devise an amethodical approach[8] to SSD (that is, an approach that does not follow rigid activity patterns), that gives effective support to situated interpersonal interaction?

The first question is highly contentious, but its clear implication is that no method specifies *the* best way to develop software systems. It is beyond the scope of the present research to enter into a detailed examination of this question.

The answer to the second question is presently emerging. On the basis of an historic review of SSD methods, R. Hirschheim et al. point out that SSD is gradually being transformed from a technical process to a social process [HKL95]. This is because it has become increasingly evident that a technical process is unable to cope with social issues, such as job satisfaction, user resistance, learning and interpersonal interaction [CS90, HKL95, Mum95]. Hence, it now widely recognised that SSD is a human activity in which people attempt to make sense of their own and others' actions through interaction [HKN91, Flo95]. Accordingly, several approaches regarding SSD as a social process have been proposed, including SSM (Soft System Method) [CS90], ETHICS (Effective Technical and Human Implementation of Computer-based System) [Mum95] and MULTIVIEW [AF95]. Although they are based on different philosophies, principles and concepts, all these approaches put the emphasis on facilitating interaction between

[8] An amethodical approach should not be confused with a so-called contingent (or eclectic) approach. In a contingent approach, several methods, considered to be complementary to each other, are blended together from the diverse aspects of SSD in order to help developers deal with the various characteristics of the project or domain [AF95, McD90]. Such an approach is usually characterised by a hybrid model in which the behaviour of developers in each step is dictated by one particular method. It also puts the developers at the centre of the development of software systems and involves user participation to a limited extent only.

developers and users who participate in the process of SSD. Their justification for this is that the success of a software system is proportional to the degree to which its users are actually involved in the development of that system. They also point out that the barriers to interpersonal interaction for SSD can be removed by supporting learning through the mutual exploration of knowledge and understanding. These new approaches conclude that there is an alternative to the development of software systems by following fixed activity patterns, and that this alternative is anchored in the social process perspective. However, these new approaches are not the author's concern in the present thesis, since they are still restricted to rigid algorithms.

Two other approaches have greater relevance to the author's interest in seeking an amethodical approach: the professional work practices approach (PWP) [And+90] and cooperative design [Kyn91].

The professional work practices approach (PWP) aims to achieve successful SSD by improving developers' professional work practices. The improvement is to be achieved by learning that combines study (such as reading documents and taking part in academic training) with experience (e.g., of changing working practices). Since PWP recognises that theories and methods are subject to ambiguous interpretation and their true consequences cannot be understood simply through theoretical work, it encourages developers to carry out practical experimentation in their concrete working situations. For example, it proposes the application of different textbook-based methods to the same project in order to gain familiarity with the details of SSD and the use of these methods. In this way, the developers gradually build a thorough understanding of their work habits through practical experience, and as a result the improved development of software systems is supported.

Cooperative design, with its emphasis on user involvement, stresses the way in which users, as developers, actually engage in developing the required software system as shown in Figure 1-1(c). This approach argues that technology should not always be applied in ways that constrain human work, but instead it should encourage reciprocal learning, whereby users and developers teach one another about work practices and technical possibilities through joint experience [CWG93]. In this respect, it acknowledges that a high degree of user participation and well-developed interpersonal interaction between developers and users is imperative for the good design of software systems. In particular, cooperative design practitioners criticise the rationalistic approach of system development, with its roots in scientific objectivism, and specifically avoid presenting any 'step-by-step' method [CWG93, Kyn91]. For cooperative design, no particular activity pattern can be set to guide reciprocal learning and creative design. Methods are thus seen more as resources to use in order to cope with diverse situations, and are not gathered into a single coherent framework [cf. Suc87].

Neither PWP nor cooperative design is, strictly speaking, a method-based approach (more specifically, neither is based on a 'step-by-step' method) [HKL95]. They provide a set of principles, concepts and techniques to support their social process perspective and guide the development of software systems. At the same time, they do not degenerate into an *ad hoc* approach, which would threaten SSD with the same problems experienced prior to the advent of methods, as discussed in [DS97, Pre97]. On this basis, it seems that both approaches might be classified as amethodical.

However, as Floyd argues in [Flo87], a social process view of SSD means that software should be regarded as an *emergent phenomenon* taking place in an evolving world with changing needs, and as the object of the processes of *learning* and *communication* occurring in SSD. This observation indicates that the social process of SSD should not only be characterised by human learning and interaction - it should also enable software systems to evolve in practice in response to these social activities, and form the base of further activities. That is to say, the system being developed should always reflect interpersonal interaction and learning in a significant way. Taking this into consideration, it is not surprising that most approaches that focus purely on social action are criticised for their 'impracticality' and 'low cost-effectiveness' [AF95, HKL95], and involve a time delay in adapting software systems to their rapidly changing environments. It is helpful to regard the amethodical approach which the author is seeking in order to overcome these problems as *a social process for SSD, but with more technical practices to enhance the viability of given social goals*. From this perspective, the PWP approach takes insufficient account of interpersonal interaction [HKL95], and cooperative design provides limited technical practices [CWG93]; hence, neither of these alternatives qualifies as the desired amethodical approach.

Research into Empirical Modelling[9] (EM) at the University of Warwick [Bey97, Bey98, BCSW99] has shed valuable light on the search for an appropriate amethodical approach. Since EM does not involve a definition of well-structured steps by formalised rules and algorithms but rather emphasises the need to improve understanding and create experience through repeated experiments and observations, it should not be viewed as a technical process like that invoked by a traditional formal procedure for SSD. At the same time, by regarding the software system being developed as a computer-based model (cf. [Leh98]) and furthering the development of this software system through modelling, EM enjoys strong technical practices. Through the autonomous interaction with the computer-based model, the modelling process is advanced situation by situation, just as human agents solve problems in everyday life. The situated modelling process enables the modeller (that is, the user of EM) to design and maintain his/her own way of learning and thus enriches his/her experience, but also enables the modeller to adapt the software system (that is, the computer-based model) to its evolving world. Indeed, research into

[9] The details of EM are reviewed in Chapter 2, and Chapter 3 provides a discussion of fundamental concepts underlying EM and other methods for SSD.

EM and its application to a number of case-study software systems have demonstrated its useful technical capabilities and helpful support for learning [Bey97]. However, as indicated above, a social process must take sufficient account of interpersonal interaction between participants. Previous work on EM has yet to reveal the extent to which EM can serve as a social process to guide SSD in a distributed environment [cf. ABCY94, BNOS90]. Hence, it is important to clarify the characteristics of EM from the social process perspective, in particular in relation to user participation and interpersonal interaction.

Extending EM to serve as a social process for SSD is the main motivation for the present research. The aim is to integrate EM with social practices so that it can better support the social process of SSD in general and, more particularly, the interpersonal interaction between developers and users for the purpose of exploring, expanding, experiencing and communicating their knowledge. In motivating and justifying the application of EM to SSD, the thesis examines the following fundamental issues:

- **Essential character**

  What is the essential nature of SSD?

- **Real-world context**

  What is the relevance for SSD of the real-world contexts in which a software system is to be developed and used? How can sufficient account of these contexts be taken during the process of SSD?

- **Human factor**

  What are the roles for human agents in SSD? Are the developer and the user best seen as directors, guiding the process of SSD, or as actors, acting out the process by following the recipe of a method?

- **Social factor**

  What kind of interpersonal interaction in SSD is appropriate? In what respect can interpersonal interaction support SSD?

- **Computer support**

  What role can the computer play in supporting human agency and interaction in SSD?

As far as this thesis is concerned, the most significant issue is that of computer support for SSD. The proposed application of EM principles to SSD forms the main topic of the thesis (Chapter 4-7). It is the radical implications that this proposal has for SSD that motivate the broader agenda, and entail a comprehensive reconsideration of fundamental issues for SSD (see Chapter 2 and 3).

1.2 Thesis Outline

The principal aim of this thesis is to investigate distributed Empirical Modelling (DEM) and its applications to SSD. A framework for DEM is developed by drawing on two important theories in social science: *distributed cognition* [Hut95] and *ethnomethodology* [Gar67]. An application of this framework to requirements engineering is proposed. In addition, a tool has been implemented to support this framework. Also, several case studies are used to illustrate the concepts and principles of this framework. The rest of the thesis consists of seven chapters that are organised as follows.

Chapter 2 reviews and illustrates the concepts and principles of EM that form the basis of this thesis. First (Section 2.1), the concept of *situated activity* whereby human agents solve problems encountered in everyday life is identified. By using the computer as a modelling tool, this human-centred process can be enhanced. It is this kind of human-centred, computer-based process that EM attempts to invoke for SSD. Section 2.2 presents the framework of EM, including its basic concepts and principles, and the process of enacting EM. There then follows a discussion of technical issues supporting EM (Section 2.3). An example to illustrate the enaction[10] of EM is provided in Section 2.4.

Chapter 3 highlights the potential for using EM as an open development model for SSD. Two fundamental comparisons between EM and traditional phase-based process models for SSD, on the basis of their dynamic and static features, are given in the first two sections. One seeks to explore the differences between these two approaches from the perspective of process enaction, and the other focuses on the ways in which they manipulate the collected information. On the basis of these comparisons, the following

[10] As in [STM95, Tul88], the novel word 'enaction' is adopted in preference to 'enactment' to reflect the way in which human action can be closely integrated with computer execution in EM.

section (3.3) considers the use of EM for SSD. The advantages and limitations of using EM for evolving software systems which are ill-defined and volatile in their operational domain in the real world are discussed.

Chapter 4 aims to establish the framework for distributed Empirical Modelling (DEM), drawing on concepts from distributed cognition and ethnomethodology. Section 4.1 describes the reasons for developing DEM and the theoretical background to DEM. A detailed framework for DEM is then proposed (Section 4.2). This section introduces a central concept of DEM – '*pretend play*', whereby modellers shape the agency of agents within the system by pretending to act as these agents do in interacting with each other. One of the main issues of DEM concerns the method of shaping agency. The way in which DEM differs in this respect from AI and previous variants of EM is discussed in Section 4.3. Finally (Section 4.4), three strategies for developing software systems are identified and discussed. The use of different strategies distinguishes software system development by design (that is, developer-centred) from development by evolution (that is, participant-centred).

Chapter 5 is concerned with the implementation issues in creating a tool to support DEM. Section 5.1 discusses the creation of this supporting tool, called dtkeden. A distributed architecture with client/server communication is devised to implement this tool. Also, a new mechanism for supporting distributed synchronous communication in dtkeden is proposed. Then (Section 5.2), different interaction modes implemented in dtkeden for supporting different styles of interaction between modellers are discussed. In Section 5.3, the concept of *virtual agent* is introduced. The implementation of this concept in dtkeden provides a convenient way to specify several instances of a feature in a model by introducing the same definitive script in different contexts. It can also be applied to the dynamic reuse of a definitive script. A comparison between the proposed reuse in dtkeden and component reuse, based on abstract data types (ADTs), is also provided.

Chapter 6 illustrates the framework of DEM and the functionalities of dtkeden through case studies. The first section uses the example of an historic railway accident to demonstrate the concepts of pretend play and collaborative interaction between several computer-based models in a distributed environment. The main concerns of DEM, and the key functionalities of dtkeden are shown in this case study. Section 6.2 highlights the important concept of virtual agent through examples. One example involves the development of a new translator for generating Eden programs from ADM programs. Another two examples illustrate how this concept can be used to generalise reusable definitive scripts. Section 6.3 then focuses on the use of interaction modes provided by dtkeden. Examples of different interaction modes are presented.

Chapter 7 considers the application of DEM to requirements engineering (RE). In the first section, the basic concepts of requirements engineering are reviewed. Three kinds of model for the RE process (REP) are identified, and the difficulties of enacting them are discussed. Section 7.2 then considers the reengineering of the REP in order to reduce these difficulties. Context and human involvement are taken into account for this purpose. A human-centred framework for the situated process of RE, called SPORE, is proposed in Section 7.3. This framework regards requirements as 'solutions to identified problems' in the real world. These solutions are 'cultivated' by people taking part in the REP through their collaborative interaction with each other in an interactive, situated manner. Two examples of cultivating requirements in the framework of SPORE are given in section 7.4.

Chapter 8 brings the major findings and conclusions of the thesis together, discusses the limitations of the present research, and also examines the potential for further research.

1.3 Research Contribution

This thesis is intended to overcome some significant limitations of EM as currently practised. Its main objective is to construct a framework for distributed Empirical Modelling (DEM) that can support the modelling activities of several modellers in a distributed environment. By drawing on the concepts of distributed cognition and ethnomethodology, a framework that highlights not only the distributed perspective on EM but also the principles and concepts of EM is established for DEM. Within this framework, this thesis proposes the concept of pretend play to help modellers to shape the agency of agents within the intended system by acting in the role of such agents. In this way, each modeller's knowledge, associated with the software system being developed or used, can be conveniently explored, easily extended, substantially experienced and effectively communicated. Thus, the difficulties arising from ineffectual interaction between users and developers for SSD, and from discounting the context in which a software system is developed and used, can be significantly alleviated and, as a result, SSD can be better supported.

The work in this thesis also contributes a new process for integrating requirements with their real-world context in order to better support requirements development. The proposed situated process of requirements engineering (SPORE), which regards requirements as 'solutions of identified problems', allows the participants involved in this process to cultivate requirements in an incremental manner. As a result, the problem-oriented requirements development is intertwined with software system development. This close relationship not only facilitates the collaborative interaction between analysts, designers and users, but also promotes a seamless integration of specification, implementation and use of the software system. Furthermore, SPORE, by using computer-based models as a communication medium, also alleviates the communication bottleneck created by passive paper-based communication between participants.

In addition, for software system development, this thesis proposes a new strategy to support the evolution of a software system in the real world. This strategy requires a software system to be developed as an open-ended, computer-based model whereby not only developers but also users can adapt the system to cope with continuous changes, even in its operational domains, in an interactive, situated manner. Through this strategy, the drawbacks of developer-centred development, such as the problems of tacit knowledge, can hopefully be overcome.

In addition to these theoretical contributions, this thesis also develops the practical tool dtkeden for supporting DEM. This tool provides modellers with a distributed environment that has four different types of interaction mode in order to support their collaborative interaction through networked computer-based models. Within this collaborative work environment, a software system is distributed to connected computer-based models and can be incrementally developed in response to the understanding of modellers. The use of dtkeden illustrates in practice the advantages of an amethodical approach to the development of software systems. Moreover, the concept of virtual agent is developed and implemented in dtkeden in order to support the need to use the same definitive script in different contexts. This novel concept is also applied to reengineering an ADM-to-Eden translator to generate more readable Eden programs. It also enables the dynamic reuse of a definitive script, both in order to reduce the size of programs and to overcome the difficulty of maintaining these programs. The thesis includes several case studies to clarify the framework for DEM and demonstrate the functionality of dtkeden. Of these case studies, the animation of a historic railway accident is the most significant [Bey98, BS99, SB98].