Architecting Tacit Information in Conceptual Data Models for Requirements Process Improvement

Williams, Gbolahan

Awarding institution:
King's College London

University of London

Architecting Tacit Information in Conceptual Data Models for Requirements Process Improvement

by

Gbolahan K. Williams

A thesis submitted to King's College London in partial fulfilment of the requirements for the the degree of

Doctor of Philosophy

July 2013

Software Modeling and Applied Logic Group
Department of Informatics
Kings College London, University of London

Author
Gbolahan K. Williams, gbolahan.k.williams@gmail.com
Software Modeling and Applied Logic Group
Department of Informatics, King's College London, Strand WC2R 2LS

Abstract

Despite extensive work in the field of Requirements Engineering, ineffective requirements remains a major antecedent to the failure of projects. Requirements Engineering (RE) refers to the body of methods associated with elucidating the needs of a client, when considering the development of a new system or product. In the literature, challenges in RE have been mainly attributed to insufficient client input, incomplete requirements, evolving requirements and lack of understanding of the domain. Accordingly, this has raised the need for methods of effectively eliciting, analysing and recording requirements.

In the literature, promising methods have been proposed for using ethnography to improve methods for elicitation because of its strong qualitative and quantitative qualities in understanding human activities. There has also been success with the use of Model Driven Engineering techniques for analysing, recording and communicating requirements through the use of Conceptual Data Models (CDM), to provide a shared understanding of the domain of a system. However, there has been little work that has attempted to integrate these two areas either from an empirical or theoretical perspective.

In this thesis, we investigate how ethnographic research methods contribute to a method for data analysis in RE. Specifically, we consider the proposition that a CDM based on explicit and implicit information derived from ethnographic elicitation, will lead to design solutions that more closely match the expectations of clients. As a result of our investigation, this thesis presents the following key contributions: (i) the introduction of an ethnographic approach to RE for elicitation and verification (ii) a rich CDM metamodel and modeling language necessary for defining and recording ethnographic analyses based on implicit and explicit information (iii) a method for mapping CDM's to high level architectural abstractions called ecologies. To compliment this work, an evaluation case study is provided that demonstrates a real world application of this work.

Keywords: Requirements Analysis, Conceptual Data Models, Data Modeling, Tacit Contracts, Ecologies, Model Driven Architecture, Ethnography

Acknowledgements

Any endeavour as consuming as the production of a thesis, necessitates the support and assistance of others. Many people have indirectly been involved in this work, and very much deserve my thanks and appreciation. I therefore express my sincere appreciation for their involvement which contributed considerably to this research.

I express my deepest thanks to my supervisors Dr. Iman Poernomo and Professor Paul Luff for their guidance and encouragement throughout this research. Their vast knowledge and expertise truly made this work possible and helped me develop my research skills and interests.

I also very much appreciate the direction and support of my thesis adviser Dr. Jeroen Keppens for providing the needed guidance during the final stages of this research. I express my thanks to Professor Christian Heath and Robert Stone for their interest and research collaboration which made this work possible.

My sincere thanks go to my committee members for their efforts and contributions which helped to improve this research, and to my colleagues in the Software Modelling and Applied Logic Group at King's College London for their research collaboration. I also extend my thanks to the rest of the Department of Informatics at King's College London for nurturing an excellent research environment.

Finally, I would also like to thank my family and friends for their consistent and valuable support in all my endeavours.

Declaration

This thesis is presented in accordance with regulations for the degree of Doctor of Philosophy at King's College London. I verify that I am the sole author of this thesis, except where explicitly stated to the contrary. The contents of this thesis are a result of my own work, and it contains nothing that is based on collaborative research. No part of the work contained in this thesis has been submitted for any degree or qualification at any other university.

Publications

The following related articles have been published by the author during the completion of this thesis:

[101] G. K. Williams, I. Poernomo, and P. Luff, *Modelling ethnographic analyses for records via tacit contracts*, in Research Challenges in Information Science, 2011.

[100] G. K. Williams and I. Poernomo, *Social Computing Theory and Practice: Interdisciplinary Approaches*, IGI Global, Information Science Reference, USA/UK, 2011, ch. Social Contexts in an Information Rich Environment, pp. 68–84.

Contents

Abbreviations

CDM	Conceptual Data Model
CIM	Computation Independent Model
DDV	Design Data View
IDV	Implementation Data View
LDM	Logical Data Model
MBE	Model Based Engineering
MDA	Model Driven Architecture
PDM	Physical Data Model
PIM	Platform Independent Model
PoIE	Post-Implementation Ethnography
PrIE	Pre-Implementation Ethnography
PSM	Platform Specific Model
RA	Requirements Analysis
RA	Requirements Engineering
RDV	Requirements Data View
SDLC	Software Development Lifecycle
UML	Unified Modelling Language

List of Figures

List of Tables

Part I

Background & Context

1 Introduction

We're entering a new world in which data may be more important than software.

Tim O'Reilly

1.1 Prerequisites for Requirements Analysis

Ineffective Requirements are an industry wide problem and remain a major antecedent to software failure [91]. Requirements Engineering (RE), also referred to as Requirements Analysis (RA) refers to the body of methods associated with understanding the needs of a client when considering the development of a new product or system. It is a critical phase within the software development process [91]. A wide array of research has shown that the success or failure of software projects is closely linked with the effective understanding of client requirements during the early stages of development, and the correspondence of those requirements into the software development process [93] [102].

The software development process refers to the cycle of activities that commit to the design, creation and evaluation of software systems. Accordingly, it outlines a number of prerequisites for the early stages of the software development process:

a) *elicitation*: entails gaining an effective understanding of what the client needs

b) *analysis*: advancing ones understanding of a set of requirements through close inspection and detailed examinations

c) *representation*: embodies methods for outlining and presenting the clients requirements

d) *communication*: conveying the requirements of the client through the software development process

Research Directions

The field of requirements engineering has continued to emerge to meet the growing demands that complex software systems impose on each of the early stage software development objectives. Despite extensive work in the field, challenges specific to the requirements analysis phase of the software development process persist. In particular, requirements for records *(or data)* presents several challenges, notably problems related to the early software development objectives. Addressing the underlying causes of each problem is not a trivial task as requirements analysis is often affected by incomplete and uncertain information [44]. This raises a number of questions for research into effective methods for requirements analysis:

i) How do we conduct 'good' elicitation?

ii) What constitutes effective analysis?

iii) What are methods of representation are sufficient for structuring requirements?

iv) What are the right methods for communicating requirements between interested parties of the software development process.

These questions have been individually answered in the literature but work still needs to be done to develop requirements analysis approaches that integrate and address the objectives of each of the prerequisites for early-stage software development activities.

This thesis aims to advance the field of Requirements Engineering by proposing a method for analysis and evaluation that is guided by each of the above questions. To achieve this, we employ *Ethnography* to address the need for an effective method for elicitation and analysis of requirements. Ethnography is social research method that concerns gaining an understanding of the activities of people within the context of their environment. To address the need for an effective method for the representation and communication of requirements, we employ *Model Driven Engineering* (MDE). Model Driven Engineering refers to a technique aimed at reducing the complexity of systems through the use of models that describe the system at varying levels of detail.

The remainder of this chapter elaborates on the context of this research, and presents an overview of the work proposed in this thesis. The next section discusses the notion of reconciling improved elicitation methods with requirements analysis methods. It introduces the concepts of Ethnography and Model Driven Engineering. Following this, we present the research goals and contributions of the thesis. Finally, an overview of the thesis is given.

1.2 Bridging the Gap

In Requirements Engineering, the expression 'Bridging the Gap' relates to the attempts to reconcile the increasing metaphorical gap between understanding the needs of a client, and prescribing the requirements to be fulfilled when the software system is delivered.

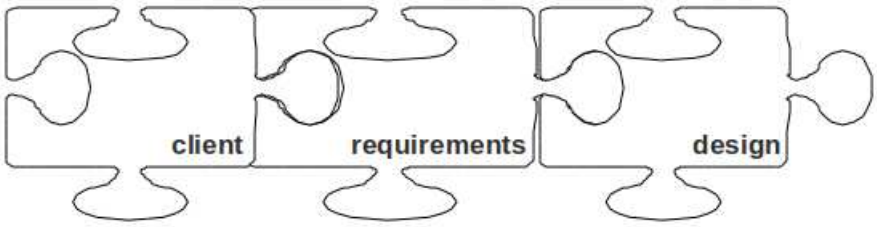


Figure 1.1: Requirements Engineering: Bridging the Gap

Within the context of software development, the expression is often extended to relate the adjoining communication activities between requirements analysis

and design, for a more effective understanding and progression of the clients needs through software development (Figure 1.1).

The problem of bridging the gap between information systems requirements analysis and software design is well known and still open. Jackson [49] used the concept of 'problem frames' to reason about software and requirements concerns at an intermediate level of abstraction. Problem frames characterise classes of problems and are composed of three parts, beginning with the *problem domain* which describes the environment which the problem resides; the *requirement*, which describes the customers needs; the *machine*, which relates to the hardware or software resource that fulfils the requirement.

The approach focuses on software development as the problem to be solved. However, successful software development is rooted in understanding the concerns of the problem domain. To this end, there has been a progressive movement in the last two decades towards the application of ethnography in the software development process to understand various aspects of the business domain [33][46][52][94].

Understanding the Business Domain

It is well known that it is important to have a good understanding of the business domain when developing software systems. It is plausible that a good understanding of the domain will lead to a more precise requirements deliverable. A good requirements deliverable increases the prospects of improved specifications and better design choices further down the development process of a system. To the contrary, not understanding the business domain could lead to ineffectiveness in requirements analysis due to possible omissions of important details of the business context of the system. This underpins the need for effective elicitation methods. To this point, it has been argued in the literature that implicit information is an important aspect of the structure of a domain model [42]. Implicit information refers to information about a domain (or context) that is understood and implied but not explicitly stated. Studies have shown that implicit information *does* get excluded and misunderstood usage contexts lead to poor quality systems [42]

Unlike explicit information, which refers to visible, detectable information about the structure of data in the business context of the system, implicit information may be difficult to detect. The side effect of not understanding the implicit information that drives a context could result in poor assumptions made about the domain model, misunderstanding of usage contexts and characteristics of the system.

In this thesis, we investigate a particular grouping of implicit information that relates to data and we call that information *tacit information*. We contend that tacit information is very beneficial at the requirements/design phase of the software development process. Upfront knowledge of both explicit and tacit information has the potential to improve software quality. Therefore, tacit information as well as explicit information should be recognised at an early stage in the requirements analysis process. However there is the challenge of identifying tacit information which leads to the proposition of utilising social research methods for this purpose.

Utilising Social Research Methods for Elicitation and Business Domain Analysis

Much work in the literature has proven that ethnography is an effective approach to developing an accurate picture of how users, data and functionality should naturally integrate within an information system [12][23][45][89]. As a research method, ethnography is based on the fine grained monitoring of behavioural patterns, often drawing on direct observation, surveys and interviews, scrutiny of video footage, examination of paper trails, and so on. Ethnography can thus be a useful way to identify best practices in an environment, but there are still open questions regarding the translation of the results of ethnographic studies into good design, specifically in relation to the ways in which data is treated in the system [42].

Ethnography presents the description of various views of a system: for example how its composite concepts, roles and relationships evolve over business processes, and how data itself grows as it traces a path from person to person or between various agents in the system. Importantly, it does not view busi-

ness processes as formal workflows, nor data as structured and rigidly defined but instead, seeks to elicit the implicit and explicit aspects of a business that might otherwise be ignored using standard elicitation methods.

When considering integrating ethnography as a method for elicitation in Requirements Engineering, one must understand what types of data it produces. The principle of ethnography takes a naturalistic view of the environment being studied. This approach often makes the ethnographic data being recorded to be eclectic. Thus ethnographic data may contain textual or diagrammatic based representations of the research, transcripts of interviews, audio and video recordings and so on. Ethnography produces a richly descriptive, often anecdotal and accurate report written in a natural language that stands in contrast to more formal requirements documents that already begin to abstract away precise specifications of the system[1]. The question of how to incorporate ethnographic reports into the software development process remains open [19]. Within context, Sutcliffe and Maiden [90] noted the challenge of transforming informal linguistic statements into more formal expressions as a problem central to this area. Further to this, there is also the question of how to purposefully record implicit information at a conceptual level, when describing the domain of a system. There is also the problem of requirements being hidden away in ethnographic reports. This makes it unsuitable for an improved requirements analysis approach. The next section presents the research goals of this thesis and outlines the direction of work that aims to mitigate some of these problems.

1.3 Research Goals

The key objective of this thesis is to investigate the proposition that tacit information is beneficial at the conceptual level in requirements analysis. We pursue the long debated hypothesis in ethnography and requirements engineering interdisciplinary research which affirms that incorporating tacit information at the requirements level, leads to improved design decisions that result

[1]Note, we defer a full description of how requirements elicitation might be conducted according to ethnography to subsequent chapters.

in software systems that more closely match the requirements of the client. The objectives of this thesis can therefore be further broken down into three core research goals.

This thesis presents a renewed effort to:

i) Bridging the gap between ethnography and requirements analysis.

ii) Investigate an approach to carrying ethnographic insights through the requirements analysis stage of software development into design.

iii) Devise an approach to navigating implementation choices at the design level, based on tacit information that has emerged through the above process.

To approach the first two goals, we consider approaches in the literature for defining and communicating requirements specifications. It is our submission that model driven engineering based methods represent the preferred approach to describing and communicating system requirements. This reflects on our overall impression of current developments moving towards model driven approaches to software development. In software development, Model Driven Engineering (MDE) advocates the use of models as first class entities for the representation and development of software systems [10]. From a requirements analysis perspective, this entails describing the specification of the system at the conceptual level constrained by the semantics of a formal definition. This formal definition can be specified using a metamodel, which is an MDE concept that serves the purpose of describing the constructs that form the basis of describing a model.

For this work, this necessitates the development of a requirements analysis approach that is rooted in an understanding of ethnography, and a formal definition for defining data requirements at a conceptual level. Successful developments in this area will facilitate the construction of requirements models based on both explicit and implicit information derived from ethnographic research.

The final goal questions what tacit information tell us about possible implementation choices. We investigate a formal definition of design level architectures and a heuristic based approach to mapping requirements level models to design level architectures. Successful developments in this area will facilitate improved decision making at the design level the approach entails identifying what types of implementation choices are best suited to the tacit insights emerge from ethnographic research.

The overall body of work of this research is therefore centred around the intersection of requirements analysis, ethnography and model driven engineering. It utilises ethnography for eliciting tacit requirements pertaining to the domain and model driven techniques for requirements representation. The next section outlines the thesis with an introduction to each of the remaining chapters.

1.4 Thesis Outline

The work presented in this thesis covers eight chapters, split into three parts:

Part I: Background & Context

Part I states the context and motivation of this research. It also presents a background of the interdisciplinary fields discussed in the thesis.

Chapter 1 - Introduction
The thesis begins by introducing this research and motivating the problem domain. A brief overview of the related fields of this research is given, namely Ethnography, Requirements Analysis and Model Based Engineering.

Chapter 2 - Background and Related Work
In this chapter, the core related areas of this thesis are discussed. The chapter is intended to give the reader some background on broad field of Requirements Engineering, Ethnography and Model Driven Engineering.

Part II: A Tacit Requirements Analysis Methodology

Part II discusses the main body of work of the thesis. It presents an extended introduction to ethnography and discusses our approach to modeling ethnographic requirements. Furthermore, it presents a metamodel and its corresponding formal definition for describing implicit and explicit information for models at a conceptual level based on information derived from ethnographic elicitation.

Chapter 3 - Role of Ethnography in Conceptual Data Modeling
This chapter is dedicated to explaining ethnographic practices and activities. Furthermore, it explains the activities in the software development process, and how ethnography fits in with these activities. Finally, it proposes an approach to applying ethnography during the early and final stages of development.

Chapter 4 - Incorporating Tacit Information within Conceptual Data Models
This chapter is dedicated to looking at Model Driven Requirements analysis predominantly from the angle of the concerns of 'data'. The chapter discusses the various complexities of transferring 'knowledge' to development. Furthermore, it introduces the notions of Tacit Information and Tacit Contracts. The former being implicit information in a data context, and the latter being a design obligation used at the conceptual level to distinguish tacit information from explicit information. Finally, the first part of a two-part case study is presented that aims to motivate the proposition of incorporating tacit information in conceptual models of systems.

Chapter 5 - A Tacit Requirements Metamodel
The fifth chapter elaborates on the concept of tacit contracts, and presents a requirements metamodel necessary for constructing conceptual models of systems based on implicit and explicit information. The formal definition of the model is provided, alongside the second part of the case study presented in Chapter 4 to demonstrate its application over a real modeling example.

Part III: Methodology Application and Conclusions

Part III presents a case study to demonstrate the application of this work in a real world example. Furthermore, it introduces the concept of Ecologies, and proposes a method for moving from CDM's to Ecologies.

Chapter 6 - A Case Study in Auction House Systems Design
The sixth chapter of this thesis is a case study evaluation of this research. The chapter presents a demonstration of ethnographically inspired Requirements Analysis in practice – a case study based on the experience of the development of an auction house system.

Chapter 7 - From Conceptual Data Models to Ecologies and Logical Data Models
This chapter introduces the notion of Ecologies and Ecology Instruments. These concepts underpin our proposed approach to mapping requirements level conceptual models, to design level logical models. For this work, a reference metamodel is given, alongside heuristics for mapping conceptual models to logical models.

Chapter 8 - Conclusions and Future Work
The final chapter concludes the thesis. It gives remarks on our findings and possible directions for future work.

2

Background and Related Work

This chapter sets the scene for this research by providing a background to its related areas. This work is focused on Requirements Engineering, in particular, an investigation into the communication gap between requirement analysis and design. The body of work of this research cuts across three major areas: (i) Requirements Engineering (ii) Ethnography (iii) Model Driven Engineering. The aim of this chapter is to give a detailed overview of these areas according to the context of our work. Therefore, the chapter opens with an introductory perspective on the importance of communication in Requirements Analysis which underpins this thesis. Following this, a background overview of the areas mentioned above is given, and a perspective on related works in the literature. Finally, the chapter highlights the contributions of the thesis.

2.1 The Importance of Communication in Requirements Analysis

Communication is the exchange of information and shared understanding between entities and individuals. It thus represents the interaction between entities in the actions necessary to impart knowledge and meaning. It is a central component of the contemporary information world and a key activity in the context of *software engineering*.

In the software engineering world, communication exists in a number of forms, and is an interaction that is pertinent between people, between states

in a system, and between systems across heterogeneous boundaries. With this perspective, in software development communication can be described as the exchange of knowledge and information between named entities and corresponding contexts.

Software development is preceded by a 'project conception' phase, where a project idea (or *need*) is defined amongst business stakeholders. This hopefully provides the foundation for successfully advancing the various stages of software development when the development process is initiated.

At the outset, the software development process begins with a set of activities aimed at elucidating the *needs* of stakeholders and communicating those needs in an appropriate manner to the various participants of the software development process. These activities are broadly understood under the term *Requirements Analysis*.

The main remit of Requirements Analysis is to bridge the gap between business stakeholders and development stakeholders. We refer to clients as business stakeholders, and participants of the software development process as development stakeholders. Business and development stakeholders comprise a range of roles and responsibilities. Table 2.1 describes a number of these roles. Each of the stakeholders mentioned individually have a set of well defined responsibilities in the software development process. Therefore, communication styles and needs present a different set of challenges because each individual holds a different stake in the development process. Information must therefore be provided to each stakeholder at the right level of complexity and detail. For example, a Requirements Analyst needs to understand how to scope client objectives and be able to translate them into a requirements deliverable that can be leveraged by development stakeholders. On the other hand, designers need to be able to interpret requirements and transform them into implementation designs that developers eventually turn into code.

Each of these activities carries significant importance within the remit of each stakeholders roles and responsibilities. In particular, communication activities in the Requirements Engineering phase of the software engineering process are essential in many aspects to the success of the project as it is the entry point into the development process. Whether it being communication

Role	Description
Requirements Analyst	consults with clients to understand business needs. Requirements analysts may also be referred to as business analysts.
Designer	translates software requirements into technical design
Developer	writes application code to build software and applications
Test Analyst	develops test plans and test cases to ensure software conformance to client requirements, and the quality measures within the organisation
Information Developer	develops user documentation for clients such as user guides and walkthroughs
Client	the organisation that is pursuing the development of a new system. The client may also be referred to as *customer* or *user*.

Table 2.1: Software Development Stakeholders Roles & Descriptions

between stakeholders and requirements analysts to determine system goals and feature expectations of the system; or between requirements analysts and information developers who document the requirements that will ultimately be passed on to a development team for implementation.

2.2 Multidimensional Communication Needs

Software projects have multidimensional communication activities, due to the several lines of communication activities between business and development stakeholders (see figure 2.1, page 15). For a software project to be considered successful, it must fulfil stakeholder requirements and meet the expectations of the client or customer. The success of each communication activity in the development process presents a challenge.

There is wide recognition that communication problems hold considerable weight in the delay and failure of software projects [15]. Despite the importance of communication activities, there is still a wide array of challenges due to the growing complexity of software systems and often arduous demands of business

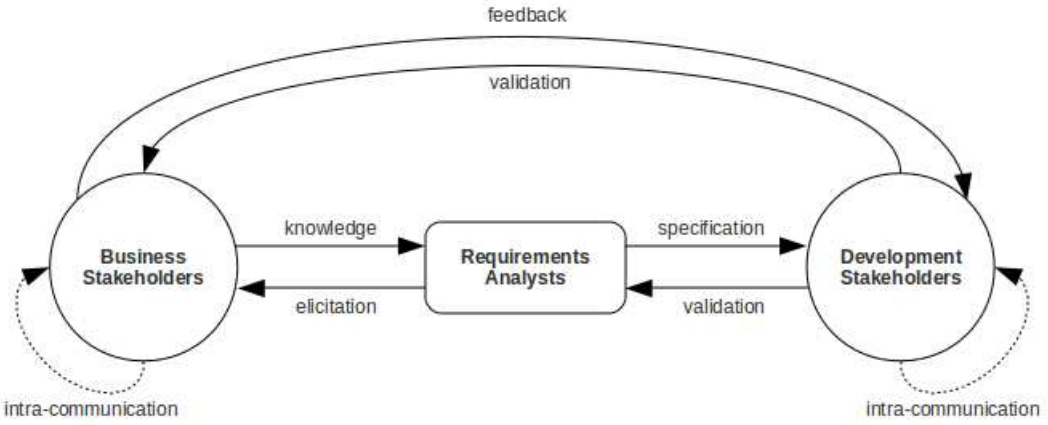


Figure 2.1: Communication Activities between Business and Development Stakeholders

stakeholders.

The problem of communication is that it is most difficult to manage within the Requirements Analysis phase due to the numerous communication activities within the Requirements Analysis process, and the individual importance of each activity. Al-Rawas and Easterbrook [4] described a field study reporting on problems related to this kind. Their study focused on the communication characteristics of the Requirements Analysis process and investigated the problems of communication between disparate communities participating in requirements activities. The study highlighted the following communication barriers in the Requirements Engineering process:

a) ineffectiveness of the current communication channels;

b) restrictions on expressiveness imposed by notations; and

c) social and organisational barriers.

One of the pressing concerns that Al-Rawas and Easterbrook [4] highlighted for future research was to investigate communicational weaknesses of current notations and methods so that those weaknesses can be accommodated for. In this thesis, the problem of communication does not begin with notation. It begins with identifying an appropriate and rich notation for recording requirements, alongside a good 'domain understanding' methodology. Furthermore

we focus on the carriage of requirements downstream through the software development process as we believe that requirements are vulnerable to some loss of precision when crossing boundaries in the software development process, especially between the requirement analysis phase and design phase. Once we have addressed these requirements, only then can we address the problem of notation. The next few sections present a deeper background into Requirements Analysis.

2.3 Requirements Analysis

The early part of software development that focuses on the decisions that are made on what to implement in relation to a clients objectives is referred to as Requirements Analysis. Requirements Analysis involves all the processes and tasks that go into establishing a clients needs and expectations of a newly proposed system.

Requirements Engineering typically succeeds project conception and business planning, and comprises several techniques and activities that have been developed over the last two decades [19, 72]. In accordance with this, a number of core Requirements Engineering activities have been identified [72]: *eliciting* requirements; *modeling* and *analysing* requirements; *communicating* requirements; *agreeing* requirements; and *evolving* requirements. Broadly speaking, each of these tasks represent a finer-grained set of tasks compared to the more generally accepted task definitions in field of Requirements Engineering which consist of *eliciting*, *analysing* and *recording*:

i) Eliciting requirements: It involves communicating with users (clients) and capturing the objectives of the software system. This is a requirements gathering activity. It comprises the activities and processes that enable the gathering and understanding of the goals, objectives, and motives for building the proposed software system [19].

ii) Analysing requirements: This involves understanding and discerning the set of elicited requirements and goals, and creating a specification that defines the customers expectations. This may be done in collaboration

with business stakeholders. Analysing may also incorporate modeling requirements. Modeling consists of a set of a well defined description of a system which can be used to establish high level goals and system tasks [72].

iii) Recording requirements: This consists of an explicit description of the requirements, aim and goals of the software system. It involves creating a requirements deliverable that will be utilised by business and development stakeholders.

Each of these activities are well known. Consequently, they each have established approaches in the literature. The next two sections present and describe some of these approaches in relation to the context of our work.

2.3.1 Methods for Eliciting and Analysing Requirements

There are two basic questions in Requirements Engineering:

i) What does the client want?

ii) Which of the clients requirements are most important?

Each of these questions brings with it, its own set of problems. The first question is one of identification. To address the task of identifying what the client wants, one must first identify who the client is. This process is described as stakeholder identification [87]. The second question is one of prioritisation and necessity. The clients requirements must be interpreted and understood in order recognise how to engage in the fulfilment of the requirements in the development phase of the system. This of course begins with good elicitation and analysis methods. We further explain these points below:

Stakeholder Identification

The IEEE-1471 standard [34] describes a stakeholder as "an individual, team, or organization (or classes thereof) with interests in, or concerns relative to,

a system". Preiss and Wegmann [82] identified several classifications of stakeholders. It is well known that there are different groups of business stakeholders and each share different opinions and concerns. Stakeholder identification therefore ensures that key individuals affected by the project are involved in the development process of the system. In [69], Mitchell et. al proposed a way of classifying stakeholders based on stakeholders possessing one or more of three relationship attributes: power, legitimacy and urgency, each of which was argued to be of importance to stakeholder theory. Coakes [20] devised a diagrammatic model for identifying stakeholders called the "stakeholder web". The web used a holistic view of classifications and groupings of stakeholders. Boundaries in the web showed the wider view of the system and its impact. This allowed the web to be examined where gaps existed, to allow an improvement of representation.

Traditional Elicitation Methods

Requirements Analysis methods have evolved over the years to a level where we have come to expect certain 'traditional' (or foundation) methods of the practice. Traditional methods for requirements elicitation such as interviews and focus groups form the basis of practice [58]. Interviews are consultation meetings between requirements analysts and business stakeholders. Meetings are aimed at helping the requirements analyst gain an understanding of the needs of business stakeholders. Requirements analysts usually approach interviews with prepared questions (or themes) that will be discussed with the stakeholder. However interviews may very well be held in an open-ended format.

Focus Groups bring together stakeholders to participate in guided discussions about a product. Focus group may also be unguided but organised around themes of interest to the requirements analyst. This is done to facilitate discussions between users about what requirements are most important to them, and for the requirements analyst to get a sense of prioritization levels for each requirement.

In addition to these methods, there are other traditional techniques such

Method	Description
Introspection	Amounts to imagining what kind of system one would want if they were the user of the system.
Questionnaires	Uses a set of questions with multiple choice answers (or free text column) to gather specific themes responses from respondents.
Protocol Analysis	Requires participants to engage in a set of tasks whilst narrating their thought process.

Table 2.2: Traditional Requirements Elicitation Methods

as introspection, protocol analysis and questionnaires. Goguen and Linde [33] provide an extensive survey of these methods. Each method is summarised in Table 2.2. For an evaluation of these methods, see Nuseibeh et. al [72].

Analogical Techniques for Elicitation

Analogical techniques for elicitation have seen wide use over the last decade because of the central focus on discovering requirements through developing a deep understanding of the user. Analogical techniques for elicitation cover personas, metaphors and creativity techniques. Each of these techniques aid the requirements analyst in understanding what requirements are important to a user.

Personas represent a typical example of a user of a system [22]. According to [17], a persona is a method for "[providing] an understanding of the system user in terms of his or her characteristics, needs and goals to be able to design and implement a usable system". They are used to position users at the centre of the design process in the context of the system being built. Personas follow the idea of Cooper's [22] methodology for creating personal experiences to create personas that represented the users to be designed. Aoyama [7] developed a persona-scenario based Requirements Analysis methodology to build a rich contextual model of targeted users. On the other hand, Potts [81] debated that human language is metaphoric and it is wise to represent requirements this way. Metaphors make use of representative symbols, actions and abstract concepts to characterise requirements. They provide a way to develop an understanding of requirements in an organisation.

A different form of elicitation aimed at 'discovering' requirements is creativity because of its capacity for driving innovative requirements [78]. In the context of creativity, Maiden and Robertson [63] described requirements as "[...] the key abstraction that encapsulates the results of creative thinking about the vision of a system". Creativity techniques are driven through activities such as brainstorming and analogical reasoning [62]. Brainstorming brings stakeholders together to think up ideas and draw up solutions to problems. Analogical reasoning is much like the idea of metaphors. It is a method that uses well understood concepts to aid the understanding of new ideas. Each of these techniques can be used as exploratory means to develop system requirements.

Analysis Methods

Analysis methods for requirements cover a wide range of methods and techniques. They extend the goal of elicitation techniques to provide a greater understanding of each elicited stakeholder requirement, hopefully with the aim of leading to better requirements management. Some methods include Value Analysis and Requirements Risk Analysis.

Value analysis is the measure of how momentous a system requirement in comparison to other requirements. The concept of value analysis dates back several years [66]. It can be looked at as the measure of importance of a requirement. The problem of value analysis is that it may prove difficult to make such a judgement due to the existence of complex relations, associations or dependences between requirements. Requirements Risk Analysis involves interrogating a set of requirements in order to identify potential risks associated with combining requirements. It also looks to identify the risk associated with introducing new requirements or the risks associated with the removal of requirements. The approach taken by the requirements analyst is to assess whether any requirement can introduce unwanted risk to the system. Requirements Risk Analysis has seen wide adoption of goal-oriented approaches to reasoning about requirements level risks [8]. The techniques originate from Goal-Oriented Requirements Engineering (GORE).

GORE is concerned with using high level goals to capture the objectives of a system. *Goals* are objectives which the system under consideration should achieve [92] for it to be considered a success. GORE provides a way to structure and prioritize requirements. This has an important benefit to the requirements analyst, in being able to identify and avoid irrelevant requirements. In the literature taxonomies have been used to represent goal types [90]. In [90], Sutcliffe and Maiden proposed a process model that used a taxonomy of goal-types to guide analysis. The approach classified goal levels (policy, domain, functional level) according to system states (e.g positive, negative, feedback, alternative, exception repair). Heuristics were used to elaborate or refine the details of each goal class.

2.3.2 Methods for Recording Requirements

Requirements Analysis leads to a deliverable (or document) that is used to communicate the aims and objectives of the proposed system to business and development stakeholders. Ideally, the requirements deliverable will describe all the aspects about what the proposed system will do. The focus of the document is distinctively to outline the goals of the newly proposed system without any concern of the underlying implementation details.

Amongst development stakeholders, the requirements deliverable is most important to the system designer because role of the designer entails translating the requirements deliverable into a design deliverable that will be utilised by developers when implementing the system. Much like with anything, different organisations have individual ideas about what the requirements deliverable should look like and what it should contain. This is perfectly reasonable, as each organisation may have different needs and what may work for one organisation, may not work for another.

The choice of recording requirements is dependent on the kind requirement being described, and the audience that will be viewing the requirements deliverable. Methods for recording requirements include both formal and informal techniques [99]. Some methods include feature lists, user stories, use cases, process specifications and models. Table 2.3 provides a summary of

Method	Description
Feature List	A documented list of a summary of features that the system should support.
Use Cases	Represents a description of how tasks and actions in the system will be executed by users. Uses cases also describe the the roles of users users performing those tasks and the dependencies between tasks and actions that are performed.
User Stories	Uses descriptive sentences to provide a detailed account of the behaviour of a system under certain conditions. It also entails a description of what actions or operations the user must invoke to satisfy a particular goal.
Process Specification	Defines the specification of a process which includes actions, constraints procedures and the treatment of inputs/outputs.

Table 2.3: Methods for Recording Requirements

each of these approaches. An extended discussion on the use of models in Requirements Analysis is given in Section 2.5. Models provide a structured approach to Requirements Analysis where the actions, intents and behaviours of a system are communicated through the use of models[1].

2.3.3 Challenges of Requirements Analysis

There are a number of known problems in the diverse field of Requirements Engineering [19]. Of the many reported problems, there are three recurring themes that keep coming up in the literature [56, 19]:

A. Lack or insufficient input from the client

B. Incomplete requirements and specifications

C. Evolving requirements

These are a set of related fundamental issues in Requirements Analysis. We address each problem as Problem A, B and C respectively:

[1]See Section 2.5: Models as a Means of Communication (Page 33)

Problem A: Lack or insufficient input from the client

Human factors such as absence of key stakeholders, infrequent meetings, distance and language barriers are well known challenges that can limit the input of a client during the requirements gathering stages of a project. Even in the absence of these problems, there are a number of nascent reasons why business stakeholders may not participate sufficiently, or not participate at all during the requirements gathering stages.

One of the main causes of this problem is due to the clients inability to explain in sufficient detail, what requirements and objectives need to be specified in order to build the product. On one hand, the client may only know what the premise for the product is and not know what options are available. For example a clients requirement is "Provide a means to move quickly from place to place in a city". This basic requirement may be satisfied by either a bicycle or a car. However which is correct? Even when the options are known to the client, the client may not be able to explain the intricate details of the product that is of interest. If it was in fact the car that was more suitable to the clients needs, what type of car will be suitable for the clients needs.

The other problem is the requirements analysts inability to express the complex requirements of a system in a way that is easily understood. The leads to the client not being able to identify with the requirements elicited by the requirements engineer. Consequently this means that requirements cannot be verified and validated by the client.

This causes a rippling effect in the software development process because it may cause the requirements analyst to make assumptions where requirements knowledge is absent. It may also mean that the client is unable to verify requirements and design specifications because the client can't visualise the outcome of the project based on the requirements proposed by the requirements analyst. In addition to this, specification and design deliverables are often expressed in specialised notations and terms leading to the clients inability to be able to validate them [4].

Each of these problems have the potential to limit the clients input in the requirements process, thus contributing to the problem of incomplete require-

ments and specifications.

Problem B: Incomplete requirements and specifications

The problem of incomplete requirements leaves much room for error. In addressing the previous issue, lack of user input will no doubt lead to incomplete requirements and specifications when the requirements analyst is unable to discover what the client wants. Even when there is sufficient client interaction, there is still the potential for requirements to get missed due to a lack of understanding of the domain or insufficient attention paid to social factors [32].

Several of the problems that occur in the design and development of software systems derive from the inadequacy of attention paid to understanding the domain. This is especially important when the domain carries contextual and implicit information. It is not surprising that some groups of researchers are adamant that all aspects of a context that can affect a system should be captured [30].

In relation to this, it has been long debated that part of the difficulties of Requirements Analysis are due to social, cultural and technical factors [23, 33, 88]. For example, organizational and legal boundaries can create hindrances during Requirements Analysis. Similarly restrictions placed on sensitive data and key systems in an organisation may lead to incomplete requirements.

Goguen and Linde [33] argued that requirements elicitation cannot be solved in a purely technological way because implicit social context matter. There is also the limitation of current design notations in being able to prescribe qualitative requirements. Qualitative requirements often take a narrative form of definition and focus on implicit aspects of a system. This is a pertinent issue due to the focus on methods for requirements that are explicit.

The problem of incomplete requirements and specifications is clear. Amongst other reasons, an incomplete set of requirements hints at the possible addition or removal of requirements in the future, leading to the problem of evolving requirements.

Problem C: Evolving Requirements

As with any change, there is an element of risk associated with it. Evolving requirements (or dynamic requirements) refers to the problem where business stakeholders continuously modify or re-prioritise the set of agreed system requirements based on a new set of aims and objectives.

Evolving requirements occur for a number of reasons. New objectives and priorities in the organisation may create the need for a new set of requirements. Accordingly, the introduction of new requirements can eliminate the need for existing requirements. Requirements may also be evolved due to the challenges in meeting requirements obligations with appropriate design solutions. Organisational factors may also drive the need to evolve requirements. For example, time, cost and availability of resources may all be important issues that the organisation may need to address [72].

The problem of evolving requirements creates the need for requirements methods that are designed for evolution, and methods that can possibly anticipate 'upcoming' requirements.

2.3.4 Discussion

Requirements Analysis is an important part of the software development process. It is clear that it is important to fully understand the needs of business stakeholders before embarking on any kind of design or implementation of a system. This places a heavy level of importance on how the needs of the stakeholder are interpreted by requirements analysts and development stakeholders. This demands the need for methods that can potentially avoid the problems of limited client input, incomplete requirements and evolving requirements. There is thus the natural progression to the field of Ethnography. Ethnography is one way to address these problems that we would typically face in Requirements Analysis. The elicitation methods in ethnography mean that these problems are less likely to occur. Later on in this thesis, we will be using ethnography with the belief that it is a good methodology that solves some of these problems. Referring back to the demand for notations and methods that support the effective communication of client requirements that was high-

lighted by Al-Rawas et. al [4], we will investigate Model Driven Engineering. We propose that the combination of ethnography and Model Driven Engineering methods, will inform a better communications model for disseminating tacit information. The next two sections will elaborate on each of these areas. Section 2.4 will discuss ethnography as a means of elicitation and analysis and section 2.5 will discuss models as a means of communication. Also discussed are some of the approaches in the literature that purport to address some of the requirements problems that have been highlighted in this section.

2.4 Ethnography as a Means of Elicitation and Analysis

> According to [30], "a proper elicitation must not only capture the customers' requirements, but all the aspects of the context that can affect the system or its use in some way".

There is a considerable amount of literature intimating the use of social scientific research methods such as ethnography in the process of elicitation in Requirements Analysis [59, 24]. *Ethnography* refers to an approach to social research that aims to develop a description that reflects the practices, concerns and perspectives of those (*subjects*) in a setting [75]. Ethnography can be briefly understood as *the study of human activities*, with the individual conducted research activities termed the *'ethnographic study'*.

The concept of ethnography derives from the field of ethnomethodology – a sociological approach to investigating the concerns on the methods people use in every day interaction [31]. In the context of software engineering research, there has been long interest drawn to Ethnography because of its empirical and qualitative methods [38, 57]. This has made it of particular interest in the field of Requirements Engineering [59], as a method for gaining insights into the subject domain.

2.4.1 An Overview of Methods of Ethnography

From a methods-related perspective, ethnographic research activities bear some similarities with traditional requirements elicitation activities. Some examples of activities include focus groups, interviews and introspection[2]. The likeness of both requirements elicitation and ethnographic activities offers the benefit of being able to seamlessly integrate the strengths of ethnography into Requirements Analysis. This has caused much debate in the literature over the necessity of adopting ethnography for the purpose of elicitation if it does not offer anything new, because of the similarities between methods [27]. It has however been convincingly argued that the distinction between both methods is one of attitude and perspective [59, 57] – ethnographers study what is going on from the subjects point of view, while requirements analysts take an application point of view in identifying possible improvement directions for the way development is carried out [83].

Other research activities include 'observation' which involves the ethnographer collecting knowledge about subjects in the *field*. The term 'field' is a technical term that is used in ethnographic research to describe the site or location under study. Similarly *field notes*, is the term used to described data collected in the field. Therefore when ethnographers are conducting research, they are said to be 'working in the field gathering field notes'.

2.4.2 The Principle of Ethnography

Ethnography aims to allow the researcher to understand the existing behaviours, interactions, and knowledge in the domain. It does not purport to identify or propose new forms of behaviours or interactions, or how the environment can be organised differently. Thus ethnographic field notes may contain different types of data [40, 86]:

i) diagrammatic based representations e.g. diagrams floor plans

ii) textual representations e.g. summaries of common and infrequent actions of subjects

[2]See Traditional Elicitation Methods for Requirements Analysis' – Page 18

iii) audio, photos and video recordings etc.

iv) transcripts of interviews

v) statistics based on questionnaires and surveys

Field notes lead to further analysis away from the field, often with some focus on certain aspects of the data which might be important for the aims of the study. At the end of the analysis work, the data collected is refined to filter out relevant from irrelevant data in an effort to effectively showcase important findings of the study.

2.4.3 Ethnography in Software Engineering

> "[...] the working knowledge of the context of use which the user has is at least as vital to the eventual success (or failure) of any system as the technical knowledge of the system designer." *Anderson* [5]

Including ethnography in any software engineering process requires an understanding of the scope, methods, applications and challenges in doing so. In this section, we define the context of use of ethnography according to how we position ethnography in the structure of communication activities between business and development stakeholders. Also discussed are methods of conducting ethnography that inform the software design process. An overview of the applications of ethnography is also given, to give an outline of some usage contexts that are used in the field of Software Engineering.

We recast the communication activities between development stakeholders and ethnographers to include the ethnographer as shown in Figure 2.2. The relationship between the ethnographer and the requirements analyst is much like a partnership. They each have the same objectives, however with a difference in outlook.

As shown in Figure 2.2, there is communication between the ethnographer, business stakeholders and the requirements analyst, in the exchange of knowledge about the newly proposed system. The ethnographer also elicits

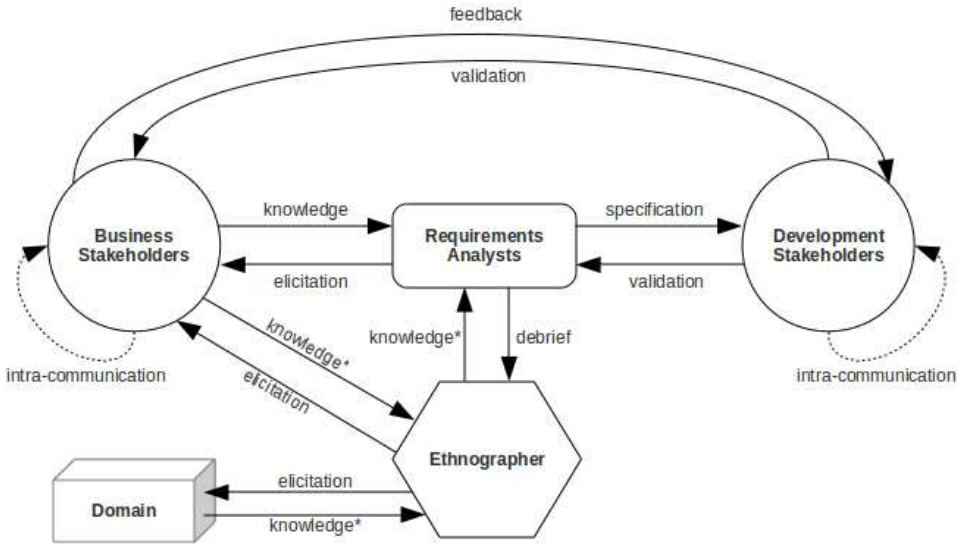


Figure 2.2: Communication Activities between Business, Development Stakeholders and Ethnographers

knowledge from the domain to advance the overall understanding of the requirements of the system.

Context of Use: An asset not a hinderance

Ethnography needs to be refined to better integrate into the disciplines that adopt it as a method of research. For this work, we adopt the notion of *Applied ethnography*. Applied Ethnography refers to the use of theories, methods and findings of ethnographic studies to solve human problems. In this thesis, we are especially interested in applied ethnography, and not ethnography in its authentic form, which also known as classical ethnography. We highlight two important differences between software engineering applied ethnography and classical ethnography [9]:

a) Intensity: Classical ethnographies are notably of high intensity. It is common for classical ethnographic studies to take several months to a few years. As software projects are often constrained by time and budget, the intensity of classical ethnography makes it unfeasible in most, if not all cases for software projects. For pragmatic reasons, applied ethnographic studies

need to be limited in accordance with the needs of a project.

b) Independence: Classical ethnographers often go into the field with an open minded approach to inquiry. This open approach may lead to unguided research directions that are of no use to a software project. In applied ethnography, a context of research is defined to allow particular research queries to be address. This can be to test theories or beliefs eg. (i) to learn about how information is used in the setting (ii) to learn about what concerns users have when using a system.

The intention is that the intensity and independence will both be driven by the needs of the project. In the next section, we present a brief overview of methods for conducting applied ethnography.

Methods for Conducting Applied Ethnographic Studies

> Applied ethnography is the best way to discover the difference between what people say they do and what they really do in their daily lives.
>
> Sanders [85]

It is intuitive that adopting methods and strategies for gaining a clear understanding of a system and how it should work can potentially alleviate some of the problems that may arise when specifying the requirements of a system at an early stage in the software developmental process. In light of this, a number of applied ethnographic approaches have been suggested in the literature. We begin with 4 approach that inform the design process that were identified by Hughes *et. al* [45], followed by visual and video based approaches to applied ethnography:

i) Concurrent Ethnography: ethnographic studies are conducted alongside systems development. Concurrent Ethnography is also known as 'consecutive ethnography'.

ii) Quick and Dirty Ethnography: a method that utilises quick ethnographic studies to provide a general, yet informed picture of the work setting in a relatively short time frame.

iii) Evaluative Ethnography: ethnographic studies are conducted with the purpose of examining already established design decisions. As the name 'evaluative ethnography' suggests, this could be for the purpose of evaluation, verification/validation.

iv) Re-examination of previous studies: an approach that uses previous ethnographic studies to potentially provide an understanding of the design of a new system.

v) Visual Ethnography: refers to the gathering of visual examples of a context (or *evidence*) as part of ethnographic findings [86]. Such evidence may include photos, drawings and audio recordings.

vi) Video Ethnography: focuses on the use of video to capture data in the field. Video Ethnography involves video recording of practitioners and participants in their natural work setting [40].

2.4.4 Challenges of Ethnography

Along with the benefits that ethnography provides, there are challenges. Eriksson et. al [26] provide a review of the problems of ethnography and critiques that have been mentioned in the literature.

One of the biggest challenges of ethnography is the issue of representation. The in-depth nature of ethnographic findings presents a serious challenge to the work of an ethnographer. On one hand, the level of detail of field notes might be in the form of vast amounts of analytical texts. While on the other hand, the reports may be difficult to understand. This in turn might make it difficult for development stakeholders to leverage because there is no context for interpretation by the designer [77].

In addition to this, there is also the challenge of the misconceptions of the intended uses of ethnography in elicitation and what it has to offer. The first

misconception is its suggested use as a replacement of Requirements Analysis. However, the use of ethnography has been mainly suggested as an addition to early Requirements Analysis activities. As previously highlighted, the relationship between the ethnographer and the requirements analyst is intended to be a partnership that aims to enhance elicitation and requirements knowledge obtained from business stakeholders and studying the domain.

Secondly, the similarities between ethnographic modes of inquiry such as interviews, focus group and questionnaires often leave some apprehension over the distinction between ethnographic methods and Requirements Analysis methods. Notwithstanding, each of these methods is different in the aims and objectives taken by the ethnographer and the requirements analyst. Each one takes a different research perspective during elicitation in order to fulfil the purpose of their roles [59].

Thirdly, ethnography has been remarked as superficial [27]. Furthermore, there is confusion over the way ethnographic findings are reported. According to Anderson [5], there is confusion about the way ethnographic findings are reported to development stakeholders. Anderson claims that people that are unfamiliar with ethnography envisage the process as being easy to simply report back, without further analysis.

2.4.5 Discussion

Applied Ethnography has been discussed in the literature as a method of understanding and learning the particulars of a domain. Whatever the cost or cost benefit, or whether applied ethnography or classical ethnography is used, there is a long standing advocacy for its use in the literature [40, 45, 86]. However the challenges of ethnography have drawn some important critiques. Much attention has been drawn to the 'issue of representation' which must be overcome before ethnography can be used effectively in Requirements Analysis. We therefore need a good, software-oriented means of notating and analysing requirements:

i) A language for recording ethnographic insights

ii) A method for communicating design obligations into design

Despite the prominence of Ethnography in Requirements Engineering, we are yet to find any work that proposes its use in a fully model driven context. To commence this effort, the next section investigates the use of models as a means of communication in Requirements Engineering.

2.5 Models as a Means of Communication

There has been a steady increase in the use of models to capture and precisely define the problem domain of software systems in the field of Requirements Engineering [10]. The concept of model-driven Requirements Analysis refers to use of models in the Requirements Analysis and specification phases of the software development process, to communicate the aims and objectives of a system in a way that can be utilised by development stakeholders in subsequent phases of development. In this section, we introduce the concept of Model Driven Engineering, and its related concepts. We also discuss the modelling notations used throughout the remainder of the thesis.

2.5.1 Model Driven Engineering (MDE)

The use of models as a means of reasoning about systems and specifying systems has seen an enormous level of growth and adoption over the past decade. Model Driven Engineering is a methodology that focuses on the creation of models that represent domain concepts and problem spaces at varying levels of abstraction. In Model Driven Engineering, models are treated as first class entities for development. This offers the benefit of simplifying the understanding of complex systems. It also provides a good way of documenting system functionality and behaviour.

The aim of Model Based Software Engineering is to make models the specific area of convergence in software development. The focus area of model driven engineering that advocates the use of models as a means of communicating development artifacts in software engineering is Model Based Software Engineering (MBSE). Of the many approaches proposed to date, the most well

known approach to MBSE is Model Driven Architecture (MDA) [98]. The next section introduces MDA and describes some important abstractions of MDA that underpin this thesis.

2.5.2 Model Driven Architecture (MDA)

Model Driven Architecture was introduced by the Object Management Group (OMG) in 2001, as a common approach to specifying and developing software systems. It has been defined as "*an approach to the development, integration and interoperability of IT (Information Technology) systems*" [68]

MDA defines two key concepts, models and transformations. Models represent the system of discourse. Transformations represent the series of steps that permit the movement from one model to another. MDA relies on the specification of a system in an initial model M_I called the input model. The input model then follows a series of transformations through one or more intermediate models until it reaches the target model M_T.

$$\mathsf{M}_I \rightarrow ... \rightarrow \mathsf{M}_T$$

The basic notion of MDA is abstraction through the use of models and the use of transformation rules to move from a source model to a target model. As such MDA begins with M_I at a high level of abstraction, and concludes with M_T at a low level. The idea is that further detail is added to each model in the transformation steps between the input model, the possible intermediate models and the target model. The next section describes some concrete abstractions in MDA.

An Overview of MDA Abstractions

There is broad understanding of three MDA abstractions in software engineering proposed by the Object Management Group:

i) Computation-Independent Model (CIM)

ii) Platform-Independent Model (PIM)

iii) Platform-Specific Model (PSM)

Computation-Independent Model (CIM)

The Computation Independent Model is the leading model in the transformation process. It represents a "computation independent viewpoint" of a system [68]. The CIM is aimed at representing elements of the system that derive from the domain. This includes domain objects, business functionality, behaviours, interactions between components, and so on. The concept of the Computation Independent Model does not prescribe the use of a particular modeling language. Its role is to provide a high level informational view point that describes the requirements of the system.

Platform-Independent Model (PIM)

The Platform Independent Model is an intermediate model in the OMG's approach to MDA that captures the implementation goals of a system in a platform agnostic manner. Accordingly the model does not dictate any specific implementation detail but represents a general structure capable of directing its transformation towards the specification model of the target platform.

Platform-Specific Model (PSM)

The Platform Specific Model is the final model that describes the specification details of the system for the target platform. Accordingly, the model captures implementation goals in a directed manner by elaborating on lower level system details.

Models in Software Development

Model Driven Development (MDD) is an abstraction of the use of models to create software. When looking at MDA abstractions from the perspective of (MDD), each abstraction layer corresponds with a software development concern: Requirements Analysis (CIM), Design (PIM), Implementation (PSM). Between each model, there is a specialisation of detail of the preceding model.

Requirements Analysis (CIM) → Design (PIM) → Implementation (PSM)

The idea is that the Computation Independent Model (CIM) is used to abstract the view of the domain which represents the conceptual requirements of the system. As such it is often referred to as the domain model. A transformation of the CIM leads to the Platform Independent Model (PIM) which represents a non-implementation specific view of the domain. This is the design logic design detail of the system. And finally the transformation of the PIM leads to a Platform Specific Model (PSM) which represents the implementation view of the model.

MDA Approaches

MDA Approach	Description
Unified Modeling Language (UML)	a visual specification language for specifying, implementing and documenting systems [14]
Meta-Object Facility (MOF)	a standard for specifying the interoperability of model and metadata driven systems [73].
Common Object Request Broker Architecture (CORBA)	a modeling language that specifies interoperability between components on different platforms [36].
Software Engineering Process Metamodel (SPEM)	A metamodel for describing concrete software development processes or families of the same. [37]
Enterprise-Distributed Object Computing (EDOC)	EDOC is a modeling framework for enterprise level distributed object computing. [1]
Common Warehouse Metamodel (CWM)	defines a specification for concerns of metadata interchange in the data warehousing environment [80]

Table 2.4: Notable MDA Approaches

Since its inception, MDA has seen a steady increase in its use in a number of model-driven approaches to software development. This created the need for a number of standards in support of representing and describing models. As a result, the OMG defined a number of standards to address specific do-

main related concerns. Notable approaches are summarised in Table 2.4. Of particular interest to the field of Requirements Analysis is the Unified Modeling Language, because of its inherent ability to support modeling any type of application. In the next section, we present an extended overview of UML, and discuss some of the diagrams and notations that allow the modeling of software systems.

2.5.3 The Unified Modeling Language (UML)

The Unified Modeling Language (UML) is an industry standard created by the Object Management Group (OMG) for documenting visual models of software systems under design and development. It provides a standardised way to specify and construct the structural formation (static view) and behavioural formation (dynamic view) of a system using notational elements to model the intended specification and design goals of a system.

UML Metamodel Hierarchy: A Four-Layer Specification

The specification of UML is defined using an approach called metamodeling. The notion of a metamodel refers to a collection of concepts that specify what can be defined in a model. Metamodels provide the basic set of concepts that support the inclusion of extensions to satisfy the modeling needs of a domain. The ability to represent an occurrence of a model that defines entails the concepts prescribed by a metamodel is referred to as instantiation.

The specification of UML is defined on a four-layer metamodeling hierarchy:

i) The Meta-Metamodeling Layer (M3): This is highest layer and forms the foundation layer of all metamodels. It defines the concepts that enable the creation of metamodels. eg. Meta-Object Facility (MOF).

ii) The Metamodel Layer (M2): The M2 layer is a specification that enables the creation of models. It is an *instance* of a meta-metamodel. This means that all elements in metamodels are concepts that are defined in the meta-metamodel. Metamodels are important because they provide

a way to specify and define new languages. Importantly, they provide a way to be clear about the syntax and semantics of a model. eg. Common Warehouse Metamodel (CWM).

iii) The Model Layer (M1): The M1 layer is a specification language that enables the creation of domain specific constructs. It is an instance of a metamodel.

iv) The Meta-Layer (M0): THE M0 layer is the lowest layer and represents an instance of an M1 model specification.

UML 2.0 Diagrams

UML provides various elements which can be used to model anything from very basic systems to large complex systems. This is possible through three building blocks:

i) Model Elements: represent basic entities and objects within the system.

ii) Relationships: used to describe the interactions between model elements. Some examples include associations, composition, generalizations and dependencies.

iii) Diagrams: used to represent different perspectives of the system and include various representations of detail.

UML consists of a set of notations to depict software systems and can be classified into 2 categories, Structural Diagrams and Behavioural Diagrams.

Structural Diagrams

Structural Diagrams are a structural specification mechanisms necessary to define the static components of a system. This can be anything from architectures to component specifications, to the structure of entities in the system. They can also be used to model relationships between entities and systems. As structural diagrams are static, they do not visualise time or sequence dependence. However, they are complete in both static and dynamic elements.

Structural diagrams in UML serve a number of purposes. They each share a number of features but provide different levels of detail.

Class Diagrams provide a way to outline the structural composition, system interfaces and relationships between entities and classes of objects in the system. A class represents the structure of an object in a domain. As UML class diagrams are used throughout the thesis, an extended summary of notations is provided at the end of the chapter. *Object Diagrams* are used to show the instance view of a system based on a particular modeling concern. *Component Diagrams* provides a way to show the overall architecture of system components and how they fit into larger systems. *Composite Diagrams* represent the internal structure of components and classes. It is also used to model the interaction capabilities of system components.

Deployment Diagrams provide a way to show deployment concerns such as hardware components such as databases and servers and their relationships with system components. *Package Diagrams* show the modularisation of system components and dependencies between modules. They may be used to illustrate the intended functionality of the system.

Behavioural Diagrams

Behavioural Diagrams are a behaviour specification mechanism necessary to visualise the interactions of components in a system. They represent the dynamic view of a system and are capable of prescribing models that define time and sequence based activities. The types of behavioural diagrams available in UML are summarised below.

Use Case diagrams provide a way to specify system behaviours and interactions between system actors. System actors represent entities in the domain which can be anything from components, devices, external systems and humans. A complete set of use cases of the system would define all the intended operations of the system and the intended participants (entities) of the system under consideration. *Activity Diagram* are diagrams that illustrate the actions and workflows in a system. Workflows consist of a prescribed sequence of activities. *State Diagrams* are used to represent the individual states and transitions in a system. States represent specific conditions of a system, while

transitions represent paths between states.

Sequence Diagrams show at a high level how different parts of a system operate, and in what sequence. They can be used to specify the detail of actions in a system. *Communication Diagrams* are used to model the messaging concerns of components of a system. Communication diagrams can be used to illustrate the sequence of messages in a system based on actions, states and interactions. *Interaction Overview Diagrams* are much like activity diagrams in that they used to describe the actions, workflows and sequences in a system. Timing Diagrams are used to describe the behaviours and actions of a system within a period of time.

UML Class Diagram Notation

This section provides an overview of class diagram notations, which are used predominantly in this thesis to communicate various aspects of models. The following concepts will be discussed: (i) Class (ii) Inheritance (iii) Association (iv) Aggregation & Composition (v) Multiplicity

Class

A class is representation of a structure in UML. This structure may represent entities or components. Classes are depicted by a rectangular box with three sections. The first section displays the name of the class. The second section lists the attributes of the class. And finally, the third section lists the operations of the class. Figure 2.3 shows some examples of classes: Vehicle, Wheels and Human. The Vehicle class lists a number of attributes and operations. In UML notation, only the first section (the class name) is required when modeling class. The remaining two sections are are optional, and can be represented as shown for both the Human and Wheels class.

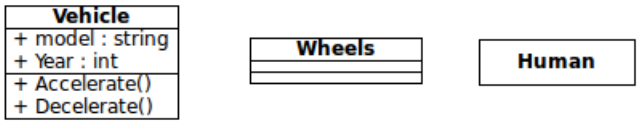


Figure 2.3: Example of Classes in UML

Inheritance

Inheritance is the ability of a class (sub-class) to assume (or *inherit*) the functionality of another class (super-class). The sub-class is referred to as a specialisation of the super-class, and the super-class is referred to as a generalisation of its sub-classes. Inheritance represents an *"is a"* relationship between two classes, therefore sub-classes are said to *extend* super-classes. For example there is an inheritance relationship between a *Vehicle* (super-class) and a car (sub-class) – a car "is a" vehicle. In UML this is represented by a line drawn between the two classes, with a hollow triangle next to the class being inherited as shown in Figure 2.4.

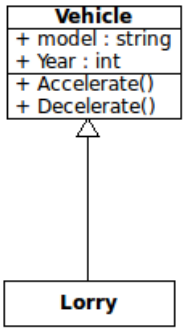


Figure 2.4: UML Class Representation showing Inheritance

Association

An association in UML allows relationships to be created between classes. It is represented by a line with two relationship ends that connect to both the associated classes. In UML associations can be named and association ends can be assigned role names, multiplicities and other properties. The role name placed on the association end of a class denotes the purpose of the association with the other class. If the role name is not present, it is assumed that the role name is the name of the class. Multiplicities enable us to dictate the number of instances of a class that relate to another. For example, a multiplicity of 1 placed on the role end of an association dictates that there is only once instance of the class in the association. We elaborate more on multiplicities in subsequent sections.

Referring back to associations, they can be broken down into: (i) Bi-Directional Associations and (ii) Uni-Directional Associations. In Bi-Directional Association, two related classes maintain awareness of the relationship. For example, Figure 2.5 shows an association between two classes: Vehicle and Person, with the association name owns. The roles can be read as follows: Person *drives* Vehicle. Vehicle is *drivenBy* Person.

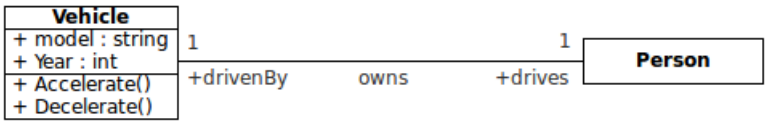


Figure 2.5: A Bi-Directional UML Class Association

In Uni-directional associations, two classes are related but only one class knows about the relationship. Figure 2.6 shows a relationship between Manufacturer and Vehicle in which Manufacturer knows about Vehicle, but Vehicle does not know about Manufacturer.

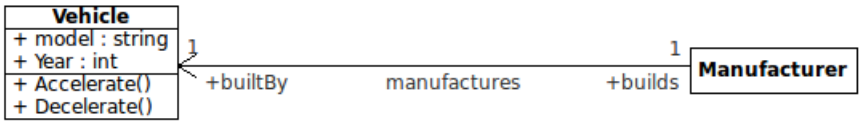


Figure 2.6: A Uni-Directional UML Class Association

Another type of association is the reflexive association. This is a Bi-Directional or Uni-Directional relationship that goes from a class, to itself. It is also referred to as a self-loop or self-relationship. Figure 2.7 shows an example of a Bi-Directional reflexive relationship.

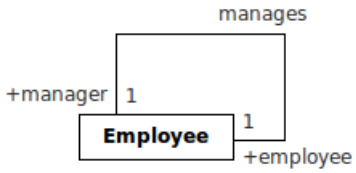


Figure 2.7: A Reflexive UML Class Association

Aggregation & Composition

Aggregation and Composition are special types of associations between classes. An aggregation represents an association between classes that have an "is part of" relationship. Therefore, one class is the whole, and one is the part. For instance an aggregation relationship between Door and Car – a door *is part of* a car. Composition is a type of aggregation with stronger ownership of the associated class. For example, an aggregation relationship between Human and Hand. Aggregation is depicted by a line between the associated classes, with a hollow diamond next to the class that represents the whole. Composition is depicted in UML as a filled diamond next to the class that represents the whole. Figure 2.8 illustrates an example of both composition and aggregation.

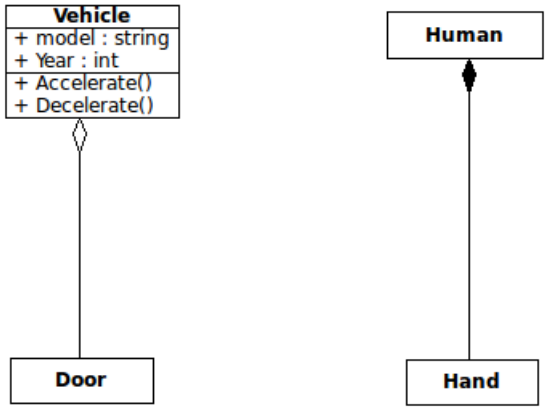


Figure 2.8: UML Class Aggregation

Multiplicity

A multiplicity defines the range of types or instances of the element described in an association between two classes. For instance 1..* means at least one, 0..1 means no instance or one instance and so on. By default UML makes provision of a multiplicity of 1, if there is no multiplicity specified on the relationship end of the association.

In Figure 2.9, the diagram can be read as follows: Once instance of Lorry has four to six Wheels instances. Similarly we can also say one instance of Lorry has two Door instances. Read backwards, we can say one Door instance *belongsTo* one Lorry instance.

Figure 2.9: UML Class Diagram Multiplicity

Example Class Diagram

Finally UML class diagrams enable us to combine the previously mentioned concepts to create more elaborate models. A modest example is presented in Figure 2.10.

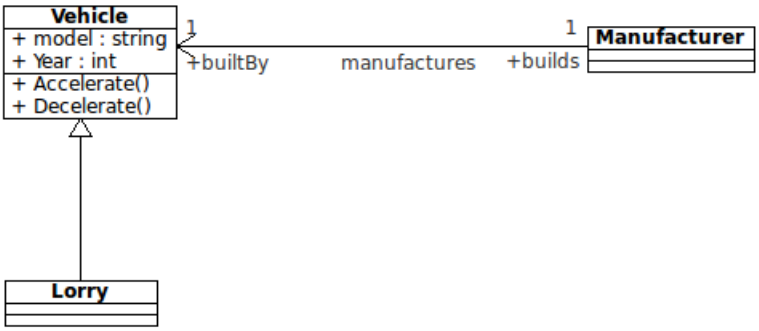


Figure 2.10: UML Class Diagram Example

2.5.4 Discussion

There is much benefit from utilising a model driven approach to development, and specifically Requirements Analysis. From a pure requirements perspective, there is significant benefit to using an MDA approach to modeling requirements.

As technology is constantly changing, so are system requirements. As requirements can be presented at a much higher level of abstraction, MDA-based Requirements Engineering can dampen the impact of changing requirements by making it more manageable to confront changes to requirements level models that would otherwise have a severe impact on the system. Even with the advantages of model based approaches to Requirements Engineering such as UML, it seems unfeasible to say that it is sufficient for all our modeling needs [54]. There are still challenges in utilising UML as a means for communication in Requirements Engineering. Requirements Analysis methods that utilise UML as a method for representation do not currently consider tacit information at the modeling level. This creates a problem when considering conveying requirements of a tacit nature between development stakeholders. In this thesis, we propose an extension of UML that enables us to address this problem.

2.6 Related Work

The work done in this thesis presents a renewed effort to bridging the gap between ethnography and requirements analysis using methods drawn from ethnography and model driven engineering. Thus it aims to enhance previous work in the area of requirements modeling and specification in this area. In this section we discuss key work that has influenced the direction of this research, and elaborate on the motivation for the research directions of this work.

The value of ethnography in software engineering has been recognised for quite some time [46]. To date, current ethnographic studies have been used primarily for *informational* and *insight* purposes, to gain an understanding of the use of systems in various application domains. There have, of course,

been a several studies covering a wide range of application areas. Studies have covered a wide range of application areas including dealing rooms [41], underground control rooms [43] and air traffic control [39, 11, 47]. Although the usefulness of ethnography is demonstrated in these studies, there are some current limitations in the way ethnography is leveraged and thus prohibit the wider adoption of ethnography with the current work practice of system designers. Ethnographic insights are notably limited by the methods used to communicate the insights gained by ethnographers into the design process.

In other areas of software engineering, ethnography has been used in requirements gathering and systems design [95] and systems evaluation [2]. The appeal of using ethnography in Requirements Engineering draws on the growing acceptance that understanding the social character of "real world, real time" activities is an important factor in software design and development [90, 46]. Previous work, such that of [33] and [45] have attempted to address communicating information between various participants in the software development process. However these works tend to deal more directly with social issues.

The most significant prior work in the literature related to the central issues of this thesis is the work of Viller et al. [95] which uses UML use cases and domain models to document ethnographic insights. Our work is rooted in the same principles of the usefulness of ethnography to inform the system design process. Their work investigated a method called Coherence which integrates social analysis with object oriented analysis into the system design process.

Coherence helps identify essential use cases and associated objects, and generate use case models by using a viewpoint oriented approach to requirements engineering to structure and apply categories of social phenomena that have been learned in previous studies to new situations. Viewpoint oriented requirements engineering organise and structure the diverse sources of requirements from the different viewpoints of a system. A viewpoint represents the task-related concerns of a stakeholder of the system [55]. Coherence can be applied in conjunction with other analysis techniques. The approach utilises sequence diagrams and class diagrams. The former is used to represent interactions in the workplace, while the latter is used to model structural aspects

of the workplace.

The approach is similar to the work presented in this thesis in that it is based on reporting results of ethnographic field work to the design process using standard notation (UML). Thus their work also approaches modeling in a manner that requires the use of elements that are not always 'explicit', which would not normally be found in standard conceptual data models. Therefore using UML to represent social aspects of work which may involve both implicit and explicit information. Unlike our approach, Coherence does not lead to a complete model of the system. It will however produce a number of excerpts from the model that result from a number of viewpoints in the system. Our approach takes a different perspective that does not lend itself to modeling excerpts of the system.

Iqbal et al. [48] proposed an approach called the 'EthnoModel'. The approach is based on metamodelling consisting of UML based notations, and an ethnographic framework consisting of three components - 'plan and procedures', 'awareness of work' and 'distributed coordination'. The approach attempts to address two specific concerns: one for the capture of requirements and the second for transforming those requirements into design. A mapping between ethnography and meta-modelling is proposed for dealing with each of the components of the ethnographic framework. For 'plans and procedures', a number of models are proposed which include use case models, concept models, and role-activity models. Use case models are beneficial in providing a detailed description of how work is carried out in an organisation. This is in contrast to documented procedures which have the potential to obscure relevant pieces of information. Concept models are tangible representation of ideas that show the interaction of concepts in a system. Role-activity models focus on the notion of roles and the interaction between them to form the basis of the description of the system. For 'awareness of work', a mapping to object models is proposed. Object models provide a structural view of the system. A glossary of terms can be used to record the terms and vocabulary used in the system. The final component is distributed coordination which can be mapped to sequence models which show the interaction between objects in the system. The overall approach combines an ethnographic approach with different types

of UML modelling to model requirements. However the approach still requires further work in being able to transform the ethnographic analyses into the design of the system.

2.7 Summary

Requirements Analysis plays a key role in bridging the gap between business stakeholders and development stakeholders. Of the many problems reported in the literature, communication problems in Requirements Analysis evoked by lack of client input, incomplete requirements and evolving requirements remain the most pertinent.

In this chapter, we have presented a number of concerns that exist along the numerous communication lines between business and development stakeholders. Key work in the literature have suggested the use of ethnography as a possible means to enhance the requirements elicitation phase of software development.

The success of Model Driven Engineering, specifically modeling approaches such as UML, creates a strong need for techniques and approaches for modeling requirements that combine the advantages of ethnographic elicitation and standard Requirements Analysis practices. However current approaches are not designed with the flexibility required to convey tacit information which is important for various modeling concerns. We need a good language for conveying ethnographic insights. In Model Driven Engineering, there are approaches to defining new languages which allow us to communicate information from one model to another. UML allows us to formalise new languages and systematically think about how models are communicated between different phases of development. This provides the starting point for: (i) a new language for representing requirements (ii) a new way of communicating requirements.

We therefore aim to address the problems highlighted by utilising an ethnographic approach to elicitation, and devising a modeling approach that enables the specification of requirements which considers both explicit and tacit information. However to begin, we must first understand how ethnography fits into the whole approach. The following chapter entails a deeper look into

ethnography and the role it plays in conceptual data modeling. This sets the scene for an investigation into how ethnographic elicitation contributes to data analysis.

Part II

A Tacit Requirements Analysis Methodology

3

The Role of Ethnography in Conceptual Data Modeling

> "[Ethnography is] a way of understanding the particulars of daily life in such a way as to increase the success probability of a new product or service or, more appropriately, to reduce the probability of failure specifically due to a lack of understanding of the basic behaviours and frameworks of consumers." *Salvador et. al* [84]

This chapter sets the scene for how ethnographic studies contribute to a method for data analysis with respect to data modeling, both at the conceptual level and at the logical level. It reflects on the roles ethnography can play when conducted in the early stages and final stages of the Software Development Life Cycle (SDLC). In the early stages of the SDLC, ethnography has a role in requirements elicitation. In the final stages of the SDLC, it has a role in evaluation and requirements verification by looping back into the original set of requirements. This has the benefit of producing a more accurate comparative of the evaluation and a holistic view of the final system. The approach is focused on data modeling, as it links into the conceptual and logical models of a system.

3.1 Introduction

In this chapter, we draw out and emphasise the role of ethnography in software engineering from a data modeling perspective. We investigate the extent to which ethnographic studies can impact positively on the requirements, design and development phases of software systems. From a requirements standpoint, we investigate what useful information ethnographic studies can inform us about software systems and organisational processes. In addition, we investigate how lessons learned from these studies can be reflected in various aspects of the software developmental life cycle, in particular requirements, design, development, evaluation.

The main goal of this research is directed towards improving models for data. Consequently, we limit the scope of our discussion to areas where ethnography impacts aspects of a system that concern data. Evidently this is of substantial interest in contexts of data modeling for Conceptual Data Models (CDM), Logical Data Models (LDM) and Physical Data Models (PDM).

3.2 Ethnographic Elicitation

> [...] if field studies could be seen simply as a methodology to gather a scent of the use situation and the context of use, and not as a method to generate requirements or implications for design, we would be able to understand its value for developers in practice much better. (Eriksson et. al [26])

Ethnography can be viewed as a collection of methods for data collection about a subject, followed by analyses of the collected data and the presentation of findings and opinions. Ethnographic Elicitation is guided by intent and purpose. Though ethnographic studies are conducted in respect of specific aims and goals, in a pure ethnography sense, the process and choices of inquiry are motivated by the ethnographer or as Millen [67] puts it:

> [...] the researcher must make motivated choices about what to study, who to observe, what activities to record, and how to analyze

and integrate the data into valuable insights.

Ethnography has proven to be an important method for understanding activities and work practices in a number domains. From the perspective of software development, its popularity and usefulness in the area of requirements gathering has received due attention in the last decade and has been discussed extensively in the literature. In this chapter, we propose a way of integrating ethnographic methods into the software development process. However before we discuss how ethnography fits into the software development process, we must first understand the process for conducting ethnographic studies.

3.2.1 Ethnographic Study Life Cycle (ESLC)

Ethnography does not impose a way to find a solution or outcome. It has been argued in the literature that there is no one method of doing ethnography [47]. However, all ethnographic studies require a well defined context and purpose of study. It is through the context and purpose that the study can be structured and initiated. The activities of ethnographic studies typically follow a sequence of events. We present a simplified model for the purposes of exposition in this thesis. The model is shown in Figure 3.1. However each activity need not be done strictly sequentially as is depicted in the figure:

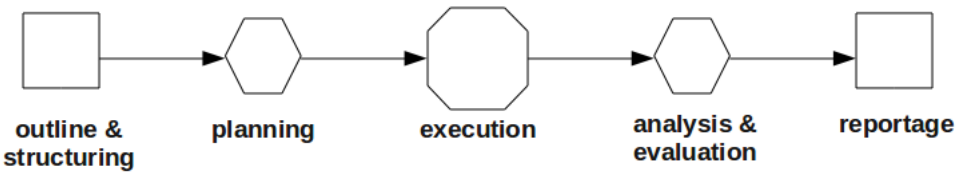


Figure 3.1: Ethnographic Study Life Cycle (ESLC)

i) Outline and Structuring: The process begins with outline and structuring which involves learning about the problem domain and formulating the intent and purpose of the study. In a sense, this phase can be viewed as the *Ethnography-Requirements* phase.

ii) Planning: This is the *Ethnography-Design* phase.The planning phase is where the scope and action process of the ethnographic study is conceived. It is also where roles and responsibilities will be assigned and delegated. Role assignments occur when there are multiple observers.

iii) Execution: This corresponds with all activities that are done by the observers in the field. The aim is to procure as much *relevant* information concerning the problem domain according to the plan developed in the previous phase. This information is not limited to descriptions on processes, organisational behaviours, tacit information and so on.

iv) Analysis & Evaluation: This activity is generally done away from the field. It is were all the findings of the study are gathered together and collated for analysis, interpretation and final documentation. In some cases, this phase may require further investigation of certain topics with key stakeholders for final judgement and conclusions by the ethnographer(s). The final product of this phase is the Ethnographic Document. This document may come in a variety of forms. What is most common, is a documentary text containing all the relevant findings from the study.

v) Reportage: This is the phase of knowledge transfer and the communication of the findings from fieldwork. It is where the ethnographer hands over the findings of the study back to the client or on to the requirements analysts. It could simply take the form of the ethnographer handing over the ethnographic document back to the client, or the ethnographer presenting key findings back to the client. Usually the hand over indicates the end of the study.

3.2.2 The Execution Phase of Ethnography

With a focus on the *execution* phase of ethnography, there are a number of common activities and undertakings of ethnographic field studies which are of particular interest.

Field Observations (Interactive)

The indirect observation technique that involves ethnographic observers assuming a role in the environment whilst maintaining the role of an observer. The technique is also referred to as 'Contextual Inquiry'. This dual role of the ethnographer provides the means for the ethnographer to gain a deep understanding of the environment. Interactive Observation is characterized by:

a) Briefing: describing the motivation and focus of the study to participants

b) Inquiry (embedding): the actual investigation with ethnographers and stakeholders.

c) Debriefing: discussing ethnographic highlights and findings with stakeholders.

Observational sessions can be structured such that they do not obstruct or interfere with the work setting. Also, if planned correctly, observation sessions can be sustained over a long period of time. This is in contrast to other techniques of ethnographic elicitation which may require direct interaction with participants and stakeholders e.g Participant Interviews

Field Observations (Non-Interactive)

The very nature of classical ethnography involves field work. The emphasis on doing field work in the 'natural setting' of the environment is to enable the ethnographer to acquaint himself with the material beliefs and principles of the participants in the environment. This to enable the ethnographer to understand the environment from the point of view of various stakeholders in the organisation - clients, managers, end users and so on.

Interactive Observations could be very useful in working out data relationships between organisational entities internal to the system, and external entities. It is normally through the close monitoring of how processes are run (e.g. analysing the workflows, understanding dependencies etc) that one can truly see the relationships and associations that exist in a system. Actions are observed and understood in context.

Key Participant Interviews

Participant Interviews are in-depth interviews with key stakeholders. This activity involves identifying key stakeholders in advance. This can be done through consultations with various members of the organisation. It is common for participant interviews to follow a set of prescribed process in the form of structured interviews. However there can also be a great deal of improvisation that comes with it.

The subject and theme of the interviews are defined by the ethnographer in order to elicit information from participants. Interviews may take place in person, telephone or videoconferencing. The style and format of the interviews need not be strict, and may be structured to go over multiple sessions. Questionnaires and surveys may also be used as a means of direct feedback on specific areas of interest.

3.2.3 Important concerns of adopting Ethnography in the SDLC

As with all rigorous scientific processes, there are a number of concerns that an organisation adopting an ethnographic approach should be aware of. In this section, we discuss three important concerns of Ethnography which may impact directly or indirectly on software development.

Deliverables

The execution phase of an ethnographic study is primarily aimed at data collection with the intention of post analysis work on ethnographic field notes after the study.

Field notes may not follow a structured format that may be useful and understandable for business and software development stakeholders. This creates an important need for a more formal ethnographic deliverable that serves both business and development stakeholders. An ethnographic report is a refined analytical document of an ethnographers notes and context documentation from the field. The purpose of creating an ethnographic report is to

make ethnographic findings easier to understand and draw information from by stakeholders. For example interview and Q&A notes being collapsed down to only divulge the more useful and necessary pieces of information that were uncovered. It may also be created to fulfil an organisations reporting guidelines and standards and redact sensitive information that may have been obtained during the ethnographic study.

Ethnographic findings (or reportage) may be used plainly to gain a sense of understanding of the system and interactions between key participants. The view of the ethnographer is intended to remain unbiased, without the intention of imposing any design obligations to the designer of the system. In agreement, Jirotka [51] insists that:

> "[...] although ethnographic analyses of interactions in the workplace can highlight systematic and often, robust, features of work practices, they do not and cannot conclude either that these features *should* be preserved or that they *will* be preserved when new technology is introduced.".

It is the designers task to leverage this information during the design process. As such, it does not directly lead to implementation decisions. Ethnographic reportage can be represented in a number of ways. However it is most often delivered as an analytical report, containing various forms of media including natural language texts, photographs and diagrams. Audio and video recordings may also form part of the deliverable. Multimedia may be included in the final deliverable to support conclusions outlined in the ethnographic report based on the analysis of data collected. Multimedia may also be used for illustrative purposes. E.g. to give a visual demonstration of an activity or a process.

In terms of content, the report is presented starting with an introduction of the study which is intended to outline the objectives of the study. It gives a description of the purpose of the study, the anticipated goals and how the study will be executed. The main body of the report contains a detailed description of the findings of the study. The document concludes with a summary intended

to spotlight any particular areas of interest worth reasoning about or investigating further. It will also include the ethnographers concluding remarks, final thoughts and recommendations.

Notations

There is no standard notation in the literature for documenting ethnographic findings in the field. This of course creates differences in opinions regarding how formal its regular form of notation is – notation in written texts. Accordingly, the notational nature of ethnographic deliverables may be perceived as structured and formal in some disciplines, however from a software engineering perspective, it is considered too informal for most situations in the 'recording' and 'representation' aspects of requirements in software development.

Ethnographic field notes are often informal, containing vast amounts of documentary texts. Even the final ethnographic deliverable itself is much considered a lengthy document. This of course makes ethnographic analyses difficult to present in a form that would be beneficial to system designers and development teams. Consequently, this creates a knowledge transfer problem in moving useful findings from ethnographic studies into software development. This, no doubt, creates a gap between the ethnography deliverable and the early part of the software development process.

Time in the Field

There are no fixed rules or guidelines on how long ethnographers should spend in the field, or how long studies should be conducted, to be considered effective. That said, ethnographic studies are often conducted over long periods of time because of the quality of data that can be assembled together over extended periods of time in the field.

However advantageous ethnography appears to be, from a software engineering perspective, this time consuming characteristic that it carries has not gone unnoticed. In light of this there have been efforts to speed up and refine ethnographic techniques to reduce the timely cost of the process. This has led to branches of ethnograrphy like Rapid Ethnography [67].

Rapid ethnography is based solely on the idea that ethnographic activities are time consuming. As noted in Eriksson et. al [26], the processes of rapid ethnography are made more efficient by limiting the focus, scope and research criteria of the field study. Other methods highlighted by Eriksson included using key informants, multiple observers and interactive observation techniques to reduce the time spent in the field. Jeffrey et. al [50] suggested a situation based selection process to determine the appropriate form of ethnography that should be used. For this reason, ethnography need not be perceived as a lengthy process. The question is not so much about ethnography taking a long time to complete, but whether ethnography is worth the time to do it in the first place.

3.3 The Software Development Life Cycle (SDLC)

3.3.1 SDLC Overview

The Software Development Life Cycle is the structure of activities and phases that define the creation, evolution or maintenance of a software system. SDLC approaches can be employed for the purpose of software development. The vast majority of SDLC methodologies incorporate the following activities into the development process:

i) Requirements Analysis

ii) Design

iii) Implementation (or Development)

iv) Evaluation

These activities are invariant phases in the SDLC, as they are always in one form or the other part of the development process of any system. The various activities in the Software Development Life Cycle all relate back to the client / stakeholders needs: From *Requirements Analysis*, documenting the clients aims and objectives of the anticipated system; *Design*, translating

the requirements into software design models and notations as a deliverable for the implementation phase; *Implementation* (or development), fulfilling as executable code, the requirements objectives set out in the design deliverable; *Evaluation*, appraising the system and ensuring that it functions correctly in relation to the original requirements set out according to the clients idea.

In many ways, the SDLC can be seen as series of transformations from the original concept/idea, to the deployed executable system. Using a structured design software development methodology, with the activities described above, we would expect to see the following transformations from conception of the idea to deployment:

[*Concept*]

$\rightarrow$ [Requirements Analysis] $\rightarrow$ [Design] $\rightarrow$ [Implementation] $\rightarrow$ [Evaluation] $\rightarrow$

[*Deployment*]

The clients concept is essentially a deliverable to the Requirements Analysis (RA) phase. This positions RA as a crucial phase of the SDLC, as the consequences of not beginning the SDLC right could result in a poor quality product and an expensive redesign of the system [91].

3.3.2 Categorization of SDLC Activities with respect to Ethnography

The software development life cycle (SDLC) is an established process of activities for developing, implementing and deprecating software systems. The process is currently at a very mature stage, gained through decades of experience and a plethora of software development methodologies. The SDLC is normally thorough, accounting for all the activities that occur during the period of development, until its release. Over the years, a number of frameworks, methodologies and models have emerged, each with their own strengths and weaknesses, but broadly focusing on specific kinds of development and managerial goals.

From small scale to large scale systems, the SDLC may be adopted differently and governed with different rules and constraints, however the overarching task of implementing software remains the same. While the SDLC traditionally accounts for wider reaching concerns such as evolution, maintenance and iteration, all SDLC incarnations *broadly* feature the same set of activities which can be categorised by activities that typically occur prior to implementation, during implementation and after implementation. Our view of the SDLC process is therefore split into three categories which we will now refer to as the **SDLC Categories**:

i) Pre-Implementation: tasks and activities that happen before the anticipated software is built;

ii) Implementation: all activities that occur during the active development phase of the system;

iii) Post-Implementation: tasks and activities that occur subsequent to the development of the system.

We refer to Figure 3.2 which shows a number of these activities divided into the above mentioned categories:

Pre-Implementation

Pre-Implementation considers all the activities that happen prior to the implementation of the system. This has been split into *planning*, *Requirements Analysis* and *design*. It is generally accepted that these are common, baseline activities that occur prior to implementation, regardless of the SDLC methodology and what models or frameworks that are used in-house for development. The software project needs to be planned, requirements need to be captured from clients and stakeholders and formalised for the requirements analyst to translate into documents that system designers can utilise in the design phase of the SDLC. It is commonplace for there to be heavy stakeholder interaction in the pre-implementation phase, as there is a lot to be understood about how the system should operate and any contingencies that should be put in place to mitigate any foreseeable risks to the project.

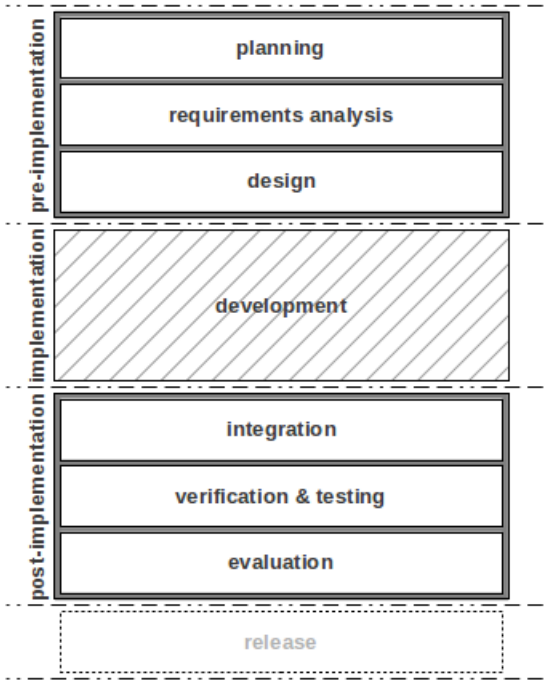


Figure 3.2: SDLC Categories versus SDLC Phases

Pre-Implementation activities also include significant interaction between requirements analysts and systems designers. This may be for the purpose of clarifying and verifying requirements, or arranging extended feasibility assessments in order to determine if certain requirements can be accomplished according to their specification.

Implementation

Implementation follows on from Requirements Analysis and design. The phase encompasses all the activities that go into building the system and accomplishing all the goals of the systems according to the requirements outlined in the pre-implementation phase. Implementation could of course come in a variety of forms depending on the software system being developed, the requirements design set forth in the pre-implementation phase, and the overall approach to building the system. For example in a Model Driven Engineering approach, code generation tools might be employed to transform design level system

models into functional code. This method may be used as an implementation strategy for (i) generating the code for the full system – where possible (ii) generating code for specific components and modules in the system and (iii) generating foundation layer code that the system will be built on top of, which can include libraries, business objects and interfaces . On the other hand, implementation might be entirely written from the ground up by application developers. In a completely different light, customizing off the shelf applications might be viewed as implementation. This would involve configuring the application using existing software packages and/or building on top of modules and software components provided by external software vendors. In practice, it is quite common that implementation involves a mix of the aforementioned approaches.

Post-Implementation

Post-Implementation involves all follow-up activities that occur in the SDLC after the end of development. In post-implementation, the system is not necessarily *ready* for release. Activities such as *integration*, *verification and testing* and *evaluation* need to be carried out. The task of integration focuses on the assembling of the various implemented sub-modules and components of the system into one cohesive system. Verification & testing involves tasks that ensure that the system is functional and works according to the requirements set out in Pre-Implementation. This task also involves benchmarking, quality checking, assurance and debriefing. It is often the case that a system goes through an Evaluation phase following verification and testing. The function of the evaluation phase is to provide an appraisal of the system, outlining things such as (i) any potential quality issues, and (ii) identifying any unforeseen issues that were not originally anticipated. Evaluation can also be seen as a post-implementation form of Requirements Analysis. *Does the system do everything that was originally intended or are there any outstanding requirements that need to be fulfilled?:* Evaluation not only provides the opportunity to verify whether requirements at the pre-implementation phase have been met, it provides the opportunity to determine if these features function

correctly as described. It also provides an opportunity to document new unforeseen usages of the system that were not anticipated during the inception of the development cycle.

3.3.3 Data Modeling

Data Modeling is the process of creating a model representation of the data behaviours in a system that are important to an organisation, in order to highlight the structure of data entities and associations between entities at varying levels of detail. In the data modeling field, there is agreement on four main uses for conceptual data models [97]:

i) Supporting communication between analysts/developers and users.

ii) Helping analysts understand a domain and design decisions.

iii) Providing detailed input between SDLC phases.

iv) Documenting requirements for future reference.

Fundamentally data modeling leaves a strong emphasis on a good understanding of the domain. In the SDLC, the data modeling process begins in the Pre-Implementation phase. The process entails the construction of three models at varying levels of complexity, with each having its own purpose.

The first of the models is the Conceptual Data Model (CDM) which is produced at the Requirements Analysis phase. The CDM typically reveals the entities and associations in the system. It is the highest model abstraction of the domain and is used to show at a high level, the composition and organisation of entities that are important to an organisation or business.

The next model is the Logical Data Model (LDM) which focuses on the design detail of the domain. The LDM is a more detailed specification of the CDM, including details such as member attributes, details of primary keys and foreign keys and so on. The LDM is constructed in the design phase of the SDLC, and is formed following close inspection of the CDM. Data modeling patterns may be used to apply necessary domain model enhancements to the

CDM. Typical enhancements include the removal of duplicate or redundant elements in the data model, establishing relationships and associations between entities and assigning identifiers to entities. It is implementation agnostic and does impose any requirements for technologies used in the subsequent phases of the SDLC.

The final model is the Physical Data Model (PDM) which takes into account actual implementation-level considerations. It therefore includes all details such as constraints and bearings on the data entities represented. It makes specific how the data will be represented on disk, as a data schema showing details such as data types, relationships, associations, constraint definitions, attribute uniqueness requirements, indexing requirements and so on.

The function of software models like the CDM, LDM and PDM is to divulge different levels of detail about the subject areas of the problem domain to different development stakeholders. The data models show varying levels of complexity with the CDM being the most abstracted view of the data entities in the system, and the PDM being the most detailed. Figure 3.3 is illustrative of how each of the models is positioned in the SDLC.

3.4 Combining Ethnography with SDLC Phases and Categories

The types of feedback-information that can normally be obtained through an ethnographic study is based on data obtained primarily through fieldwork observations. This information may specialise to the different activities of the software development life cycle. In order to understand the advantages that qualify ethnography as a structured and systematic mode of inquiry, we must understand what ethnography is not. Ethnography is not:

a) a replacement for Requirements Analysis.

b) a rigid process. It can be refined to fit an organisations software development model.

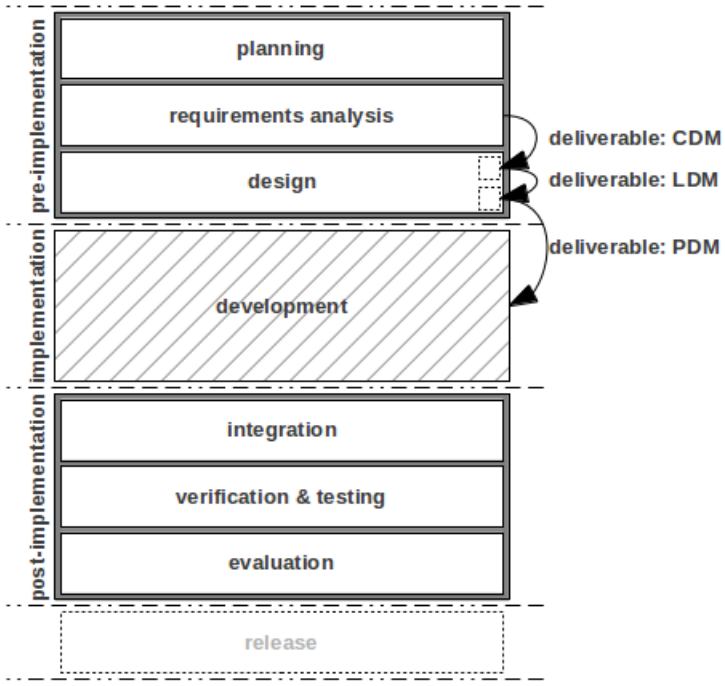


Figure 3.3: SDLC Categories versus SDLC Phases showing Life Cycle Data Models

In accordance, ethnography can be adopted and integrated with standard Requirements Analysis activities.

3.4.1 The Overlap of Ethnography and Requirements Analysis

As Requirements Analysis is the leading activity of the software development lifecycle, it is an important and necessary part of the SDLC, as it provides the basis for all further development work that is carried out in the phases that succeed it. Young [102] defines a requirement as *"a necessary attribute in a system"*. Requirements are important because they dictate what the system must do and the conditions and constraints under which it must do it.

There is a convenient overlap between ethnographic activities and standard *pre* requirements analysis tasks. With respect to software engineering, ethnography can be viewed as requirements analysis in the broadest sense pos-

sible. Requirements Analysis is essentially looking at what the client wants, understanding the wider domain and exploring how things work. This is akin to ethnography – observing natives, their behaviours and their environments. Both requirements analysis and ethnography result in deliverables which can be utilised in subsequent phases of the SDLC.

In Chapter 2, we presented an overview of some of the reported advantages of ethnography. Despite the reported advantages, particularly of ethnography being a strong method for analysis, it does not produce a formal concrete item that can be utilised by system designers and software developers. Though it can be said that ethnographic deliverables can be translated to formal documents, to our knowledge there are no guidelines or heuristics for doing so.

For the most part, ethnographic studies lead to an *ethnography report*. It is a detailed, informal text-based document that *details* rather than summarizes the findings of an ethnographic study. In the context of requirements analysis, the ethnography report adds an additional layer of information to standard requirements deliverables. Voss et. al [96] summarise it as follows:

> "[...] ethnographic studies of work in various settings have been instrumental in uncovering the seen-but-unnoticed aspects of work that have so often escaped attention in requirements-gathering exercises and have therefore not been supported in the resulting systems design."

Requirements Analysis on its own leads to a requirements document which in summation contains an outline that documents the behaviour of the system. The process of requirements analysis typically involves constant communication and interaction with stakeholders and various members of the organisation. This is illustrated in Figure 3.4, which shows the flow of knowledge and information gathered during the ethnography and requirements phases. The flow of knowledge passes through the SDLC. It originates from the problem domain, and is elicited through ethnography. The knowledge flows through the requirements analysis phase and further down the SDLC. The bidirectional arrows show the ongoing interaction between stakeholders and members of the organisation throughout the SDLC, as there is continuous interaction between

stakeholders, the wider organisation and the ethnography and requirements analysis process.

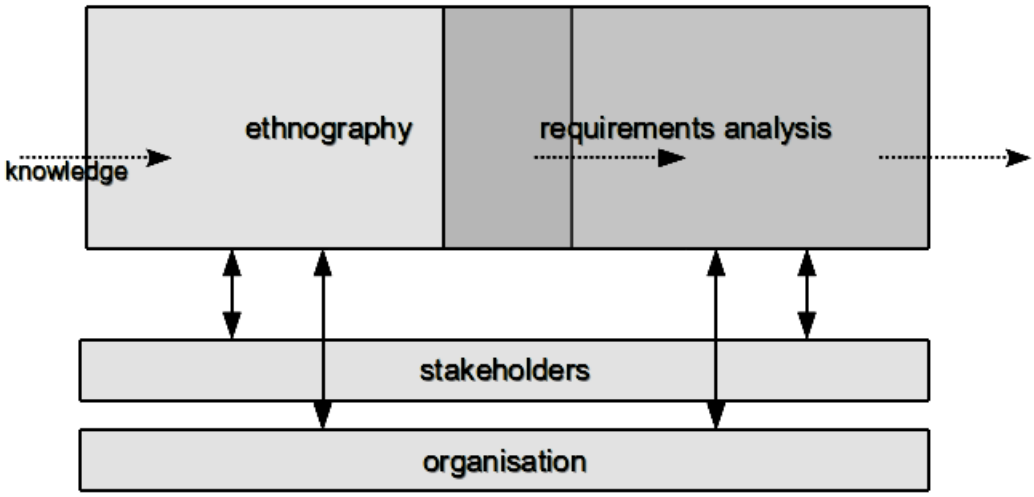


Figure 3.4: Ethnography / Requirements Intersection

3.4.2 Bridging The Gap between Ethnography and Requirements Analysis

Ethnography provides the means to facilitate some of the functions of Requirements Analysis. As such it can be seen as a pre-requirements task which may overlap with standard requirements practices. We posit that the main points of focus cross over. For example field notes from ethnographic observations are essentially documented use cases and activities. From the point of view of data analysis, this is what is being done when an ethnographer is looking at the context of an environment and how data moves around the environment.

Our view is that the findings of an ethnographic study should feed into the requirements analysis process, and should be incorporated in the requirements document. What is worth noting however is that ethnography on its own is not going to replace requirements analysis. It can replace a lot of the overlapping techniques, but importantly the requirements document that is passed on as a deliverable from the requirements analysis phase to design cannot be in the form that ethnography reports are at the moment.

3.4.3 Ethnographically Inspired SDLC Categories

In this section, we discuss the SDLC categories in relation to ethnography. We however limit our discussion to issues relating to ethnography during the Pre-Implementation and Post-Implementation SDLC categories only. To more closely align our view of how ethnography should map to the different SDLC categories described previously, we introduce the concepts of Pre-Implementation Ethnography and Post-Implementation Ethnography.

Pre-Implementation Ethnography (PrIE)

PrIE is the application of ethnography techniques to Pre-Implementation software development activities, in particular *Requirements Analysis*. With respect to ethnography, we contend the following points as keys-areas of importance in requirements analysis:

i) Determining key stakeholders of the system – identifying users, key informants, project sponsors and so on.

ii) Problem Identification and Analysis – detailed analysis of the problem and decision criteria for working towards a solution.

iii) Understanding the characteristics of the environment – discerning the operating conditions of the system.

iv) Recognising potential impact – gaining an understanding of any impacts that may arise as a result of modifying the system and/or gaining an understanding of the impact of introducing a brand new system.

In the literature, ethnography has been suggested as a means of identifying key stakeholders [28]. Stakeholder identification is known to be an important part of requirements engineering. According to Conger [21], a stakeholder refers to *"the people and organisations affected by the application"*.

Note

Given its use in this thesis, unless otherwise stated, we look at stakeholders strictly from the point of view of people within the organisation that are directly affected by the application - i.e the end-users of the system in question, rather than from the much broader definition of a stakeholder which covers any individual that can affect the objectives of the system.

Sharp et. al [87] argue on the importance of identifying the right stakeholders to act as participants of the requirements engineering process. From an ethnographic perspective, it is vital that the activities of current users of the system are understood. This is the motivation for contextual inquiry, structured walkthrough's and other user-centric research methods that investigate user behaviour in context. This however does not seem unreasonable when drawing on the fact that current users are in fact the 'inhabitants' of the environment which they have become accustomed to.

Alongside identifying key stakeholders, developing software systems involves the important step of understanding the characteristics of the environment under which the system will be deployed. In this regard, Foucault et. al [28] described the value of ethnographic fieldwork as indisputable. They also described it as an especially important tool for guiding development teams to understand lesser understood domains. Their case study in [28] presented a series of ethnographically-inspired techniques which allowed key observers to gather relevant information to guide software development.

Ethnography enables observers to establish points of success and failure in existing systems. In this sense, it can be used to evaluate the viability of deploying new technologies. An example of this is the dealing room study by Heath et. al [41]. The study was done to explore aspects of the organisation of collaborative work in a real life setting. It assessed the feasibility of proposals for deploying *future* technologies for trading dealing rooms.

In summary, Pre-Implementation ethnography provides opportunities for researchers to study in-depth, the day-to-day activities of the various stakeholders of a system within an organisation. It can be used as a strategy to review current practices, analyse problems and speculate on possible directions

for improvements. On the other hand, it can be used purely as a method for analysis and requirements gathering. Ethnography is more to do with documenting findings rather than determining what will lead on from the analysis of ethnographic reportage. As such, what pre-implementation ethnography will not tell the designer is what the system will look like after implementation.

Post-Implementation Ethnography (PoIE)

We will use the term PoIE to describe the application of ethnography to Post-Implementation software development activities, in particular *Evaluation*. In the verification and testing phase, PoIE can provide an immediate perspective on the software system or product following implementation. While this might seem very broad, it can be tailored to the evaluation methodology being used by the organisation in question. Post-Implementation Ethnography can also be used to arrange and extract findings from focus groups and pilot studies. This may occur during the final stages of development when the software is released to a small set of users to elicit feedback and suggestions on what works well and what does not. It can also be used as a means to study the impact of the system under different conditions and other post-deployment concerns.

There are several areas where Post-Implementation ethnography has proven useful. For example, a study by Martin et. al [64] explored the organisational aspects that influence software testing. Their study highlighted the importance of reasoning about software testing as a socio-technical process, rather than a technical process.

Post-Implementation Ethnography may continue to occur even after deployment. It may be adopted as a forward looking review strategy for ongoing monitoring of the system in a production environment. In the case of software pilot deployments, ethnographic reportage of 'pilot' users can be used for comparative analyses. Such analyses may be used to verify common Post-Implementation Review objectives such as:

i) Determining stakeholder satisfactions: looking at the effects on the end users and verifying whether the end users' needs have been met.

ii) Identifying any key areas of concern: ensuring what was delivered actually works and potential areas of further development

iii) Cost Validation: Assessing whether the running costs of the project were as predicted. This also includes cost / benefits analyses.

iv) Evaluating how the project was run: measuring the success of the project.

The Post-Implementation Review is essentially a 'lessons learned' document, which is often times a deliverable of a completed project. Another benefit is that Post-Implementation Ethnography provides a structured approach to overseeing user generated feedback data such as questionnaires and user satisfaction surveys. In the case of software pilot deployments, it can be used to efficiently identify any critical areas that may delay final organisation-wide deployment.

The Importance of Ethnography in SDLC Categories

There is considerable empirical evidence suggesting that quantitative and qualitative analysis of systems yields solutions that meet stakeholder requirements more accurately [95].

We have categorised the various roles of ethnography in the SDLC into Pre-Implementation Ethnography (PrIE) and Post-Implementation Ethnography (PoIE) as shown in Figure 3.5. This is important because of the different usage patterns each carries. Adopting a methodology that incorporates pre-implementation and post-implementation ethnography would offer invaluable information, right from potentially enhancing the quality of requirements to providing a way of structuring and analysing system evaluations after implementation.

Ethnography has significant merits within the software engineering domain. Ethnographic reportage provides the underpinnings for a comprehensive evaluative picture of various deliverables in the life cycle of the software. Understanding key requirements elements at an early stage, will go a long way into determining the shape of the design of the system, and what may or may not be included-in at a latter stage in the developmental life cycle of the system.

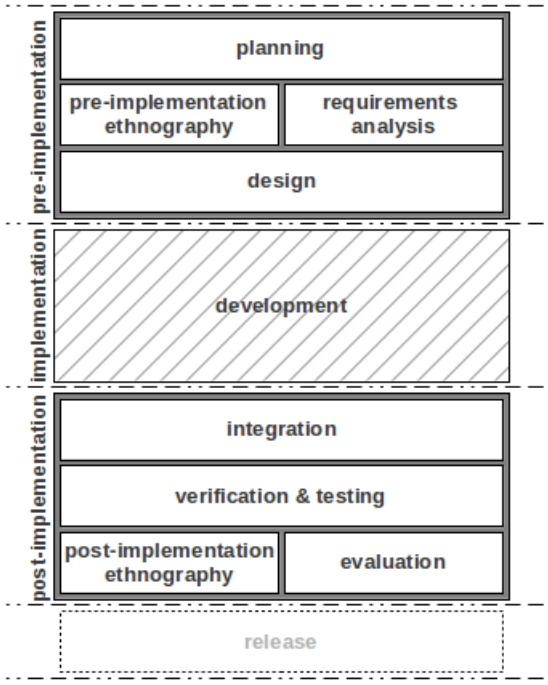


Figure 3.5: SDLC Categories versus SDLC Phases showing Ethnographic Phases

Pre-Implementation Ethnography contributes favourably to the analysis process by giving an *'as-is'* view of the current environment without bias. In this regard, when the ethnography deliverable is passed on to the designer, what Pre-Implementation ethnography will not conclude, is the designers decision as to what features must be implemented in the final system, or the final direction of design.

3.5 Ethnography in relation to the SDLC

Based on the merits of Ethnography in being able to facilitate the process of inquiry and evaluation in an organisation, we propose an approach for data analysis and validation that leverages 'ethnographic' activities in the early and latter phases of the SDLC. This aligns very well with elicitation activities in the Pre-Implementation phase and verification/feedback activities in Post-Implementation.

Our approach is aimed at bridging the gap between knowledge in the conceptual domain which is both explicit and implicit, and the introductory body of the requirements analysis phase of the SDLC. As the approach focuses on data modeling, it aims to improve on the quality of Conceptual Data Models constructed in the requirements analysis phase. This will potentially lead to Logical Data Models and Physical Data Models that are more representative of the data requirements of the organisation. This has the potential to alleviate the communication gap between what the organisation wants, and how this is delivered downstream into the SDLC.

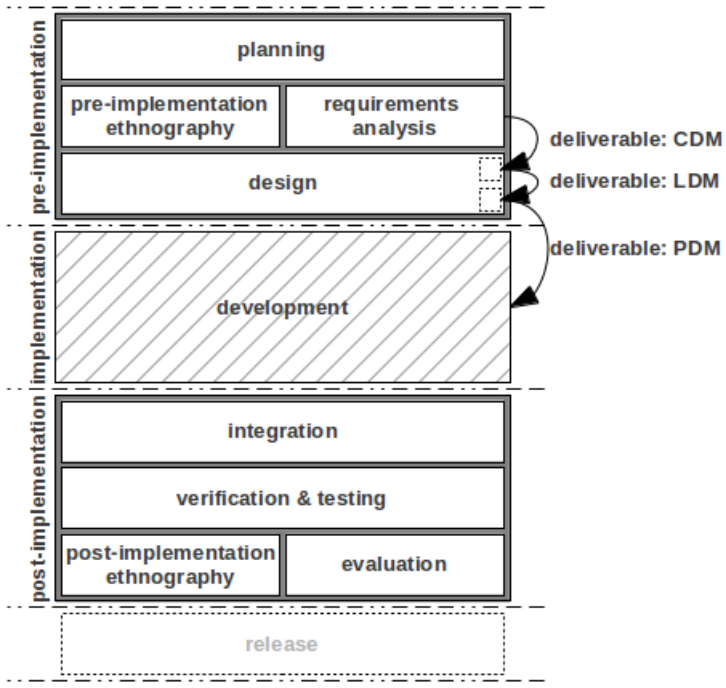


Figure 3.6: SDLC Categories versus SDLC Phases showing Life Cycle Data Models and Ethnographic Phases

Consider the illustration of the approach in Figure 3.6 which shows the SDLC Categories and phases in conjunction with the life cycle data models and ethnographic phases. It shows the overlap of ethnography in both the Pre-Implementation and Post-Implementation phases of a structured design methodology. Also shown, are the expected data model deliverables in the Pre-Implementation and Implementation Categories.

As the very nature of Pre-Implementation Ethnography falls into the area of responsibility of the requirements elicitation phase, it overlaps conveniently with the RA phase. This is the basis for the hybrid-overlapping phase between pre-implementation ethnography and requirements analysis. The idea is that ethnography will be used to make the following kinds of observations about the data context:

i) Relationships between implicit and explicit information (not limited to entities, attributes, roles and associations): An understanding of the behaviours and interactions between entities, attributes and so on.

ii) Invariant properties of the domain: A view of static, non-changing characteristics of the data context.

iii) Modes of use: A description of the different data usage scenarios and usage contexts.

iv) Complex business scenarios and user interactions: An understanding of how users interact with business entities, and an appreciation of business principles or constraints imposed by the domain.

These are the kinds of data concerns that fall through the SDLC and create knowledge 'gaps'. From a data quality perspective, projecting these kinds of insights on to the CDM sets a good precedent for appropriately communicating information requirements to the LDM and subsequently to the PDM. This can aid in identifying problems or ommitted requirements at an early stage.

In Post-Implementation the hybrid-overlapping phase between evaluation and post-implementation ethnography combines ethnographic methods and traditional system evaluation methods. The objective is to stage an open inquiry and validation exercise into the quality of the final product. This can be done by conducting user pilot trials, field observations, interviews with users, soliciting feedback via questionnaires and so on. Purely from the perspective of evaluation, it is reasonable to want to identify requirements that have not been implemented accurately before the final product is deployed. Post-Implementation Ethnography provides a structured approach for this type of

investigation. It may also be beneficial to identify not only what has gone wrong, but what has worked well. This can be helpful in identifying design patterns within an organisation and building expertise around a particular problem area.

3.6 Summary

It is important at the start of the software development cycle, that SDLC stakeholders have an accurate understanding of the problem domain. Furthermore, the communication of stakeholder requirements in between the SDLC phases is an important role in the SDLC. In this regard model driven approaches to software development have been used extensively as a means of communicating data and information requirements between the different phases of the SDLC.

In this chapter we have explained how ethnography can be positioned in the software development life cycle, to facilitate a software development approach that is focused on uncovering detailed requirements that are communicated throughout the SDLC phases. We believe that ethnography contributes to both the early and final stages of the SDLC. The terms Pre-Implementation Ethnography and Post-Implementation Ethnography were introduced to describe an approach that leverages ethnography during requirements gathering, and during the system evaluation respectively.

We contend that there is significant value in utilising implicit information in a model driven approach. The next chapter investigates how implicit information may be derived from an ethnographic study and the premise of utilising it in the conceptual data model of a system.

4

Incorporating Tacit Information within Conceptual Data Models

> "[Conceptual Data Models are] the blue-prints for representing specific data views related to a particular problem domain". *El-Ghalayini et. al* [25].

There is a growing appreciation in the literature concerning the value of incorporating tacit information in Conceptual Data Models (CDM). Conceptual Data Models illustrate from a high level, the data entities within a system, including any relationships and associations that exist between these entities. In this chapter, we investigate the proposition that conceptual data models based on both explicit and tacit (or implicit) information lead to design solutions that more closely match the requirements and expectations of stakeholders. Also discussed in this chapter, are the methods for requirements elicitation that highlight and draw out tacit information in an organisation.

4.1 Introduction

Within the field of knowledge management there are two types of information, implicit information and explicit information. Implicit information is data that is unspoken but understood and inferred by stakeholders about a data context. Explicit Information is information that can be readily identified in a data context.

Traditional data modeling tasks have focused primarily on the representation of explicit data attributes in CDM's. This affords the tendency to construct CDM's solely on explicit information and defer implicit information to use case descriptions, user stories etc. It is questionable whether a 'blueprint' of a system lacking meaningful implicit information can be regarded as complete and accurate, and whether or not these facets of information should be expressed in the CDM.

The notion of incorporating implicit information in data models draws on the belief that there is another dimension that needs to be explored which involves knowledge that is implicit, embodied and not articulated [53]. It is this information that is most commonly referred to as *tacit information*. However, in the context of software modeling, the terms tacit information and implicit information are used interchangeably in the literature.

The embodiment of tacit information in conceptual data models has the primary goal of communicating a clearer picture of the data model through an arguably richer description of the CDM. This is done by including all relevant implicit and explicit Forms of Data in the data model of the system. This process falls at the intersection between ethnographic elicitation and requirements analysis, in the Pre-Implementation phase, where conceptual data modeling is aligned.

4.2 What is "Tacit Information"?

Tacit information is any form of information that is not explicit, but inherently understood by the users and stakeholders of the system. The word *tacit* itself as defined in the Oxford English Dictionary means *"understood or implied without being stated"* [74]. The notion of tacit information originates from the concept of tacit knowledge which was introduced by the Hungarian philosopher-chemist Michael Polanyi in his much quoted book *'The Tacit Dimension'* [79]. Polanyi contested that all knowledge embodies tacit elements to the extent that knowledge cannot be entirely explicit. He worked under the assumption that *"we can know more than we can tell"*. It is this residing knowledge that is termed *tacit knowledge*.

There is a personal quality in tacit knowledge that makes it hard to formalise and communicate [70]. Nonaka [70] insists that there is a firm connection in an individuals tacit knowledge that is deeply embedded in their actions, commitment and involvement in a context. This idea can be expanded when considering the perspectives that entail individual and shared experiences when looking at requirements for software systems. Individual and shared experiences create a personal perspective of the system. Understanding these experiences enables the development of a common perspective of the knowledge that is understood by different stakeholders of the system.

This is of course in contrast to explicit knowledge, which is defined by Nonaka et. al [71] as knowledge that "[...] can be expressed in formal and systematic language and can be shared in the form of data, scientific formulae, specifications, manuals, and so forth". This represents any precise knowledge conscious to the minds of stakeholders.

At the highest level of abstraction and data representation there is much benefit in creating conceptual data models that embody both implicit and explicit information. There is the benefit of a formal communication line between the requirements analyst and the designer. This facilitates granular level reporting of requirements details to the design phase via the CDM. The advantage here is that upfront knowledge of implicit information about the system may ultimately lead to more informed design decisions. We contend that these design decisions will tie-in more closely to the intended goals and objectives of the stakeholders of the system.

4.3 Transferring Knowledge to Development

There is a division in terms of the way knowledge is represented at the requirements level and at subsequent phases of the software development process. Between each level, from requirements to design, through into implementation, there is a complexity shift in representation, with each phase adding more specific implementation detail to the conceptual information that was initially conceived and passed-in at the requirements phase.

Established approaches for transferring knowledge of a data context to

development include Entity Relationship Diagrams (ERD) [18], Unified Modeling Language (UML) [14], and derivative works of these approaches. Conceptually, these approaches are intended to document the data centric features of a system that are important to stakeholders.

Entity Relationship Diagrams show the logical representation and structure of an organisations data. The main components of ERDs are Entities, Attributes and Relationships. *Entities* represent important categories of concern of a system that data will be stored. *Attributes* are the properties and characteristics of entities, however entities themselves may be characteristics of other entities. As such, they can also be attributes. *Relationships* represent the organisational associations between entities of the system. Though there are several modeling notations used in requirements engineering, ERDs are seen as the typical CDM deliverable.

UML shows both the structural and behavioural characteristics of a system in order to understand how data is navigated within a system. At the core of UML, are a set of notations and diagrams each with different intents and purposes. UML draws on techniques from data modeling, thus making it largely sufficient as a form of notation for laying out CDM's. Class Diagrams are arguably the most used type of UML diagrams for representing data models. They are useful for illustrating requirements to stakeholders at varying levels of detail. They can also be used for laying out very detailed design guidelines for developers.

The highlights of conceptual modeling are clear in the literature [3]: they enable more efficient communication of data requirements which enables a shared understanding amongst business and development stakeholders. Even as the model evolves into other forms, as it traverses different stages of the SDLC, beginning with a more accurate model should alleviate potential issues with miscommunicated or error prone requirements. CDM's also provide a blueprint of the business domain that serves as a valuable resource to stakeholders that are new to the domain. e.g. designers, developers and project managers. In addition, they serve as a form of documentation of data requirements.

Given these highlights, what is clear that ERD's and Class Diagrams do

well is convey explicit information about relationships and associations between data entities of the systems they describe. However, what they do not do well, and what is less often seen in conceptual models is the embodiment of tacit information. Looking at how data models are moved downstream to subsequent phases in the SDLC, a conceptual data model that does not describe implicit associations and attributes will lead to logical data models that are either incomplete, or LDM's that deviate from the original requirements completely. This would also create problems when the PDM is being layed out.

The topic of transferring knowledge to development thus has its problems in the formal documentation of tacit information in high level system models (or conceptual data models). From the perspective of data, the knowledge captured by ethnographic studies is ultimately lacking in CDM's. This captured knowledge *is* tacit information and should be equally pertinent in Conceptual Models of systems alongside explicit information.

4.4 Ethnographic Perspectives on Data

Jeffrey and Troman [50] define ethnography as "[...] research taking place over time to allow a fuller range of empirical situations to be observed and analysed to allow for the emergence of contradictory behaviour and perspectives. Time in the field, alongside time for analysis and interpretation allows continuous reflections concerning the complexity of human contexts". Concerning data, there are many questions that can be drawn from this. In particular, why and how ethnography will help with defining software models at the requirements analysis level, specifically with conceptual data models.

One of the most common advantages of ethnography that has been highlighted in the literature is its potential to act as an open-ended strategy to both *requirements elicitation* and *requirements verification*. The next few sections provide key information on these areas, and highlight some of the most salient aspects of software engineering-based ethnography.

4.4.1 Knowledge without Conceptual Data Schemas

Since ethnographic studies produce a vast amount of reportage on the problem domain, it is important to analyse the data and make it relevant as a fit-for-purpose deliverable to the requirements/design phases of the software development process. Current forms of ethnographic reportage do not fit this criteria, thus creating a need for methods to communicate results of studies to designers in a form that is more well understood [46]:

Hughes et. al [46] argued on the need to develop techniques to present ethnographic findings to designers as a prerequisite to including ethnography in the requirements engineering process. The very nature of ethnographic deliverables makes this a challenging task. This is due to the predominant use of natural language text, supplemented by drawings and other forms of multimedia such as audio and video recordings.

Despite the possibility of ethnographic reportage producing structured information and key pieces of data, ethnography itself is not data modeling. Ethnographic studies do not produce data models or schemas. At least, not in the software engineering sense - hence the title, *knowledge without conceptual data schemas*. Data modeling may be a task that might be included as part of a particular ethnographic software development methodology. However traditionally, ethnographic reportage is often produced as an unbiased analytical document of the findings and observations of an environment.

Conceptual Data In An Organisation: Understanding Tacit Aspects

The various strands of information that contribute to the structure of data entities and relationships in an organisation can be referred to as 'conceptual data'.

Understanding the conceptual data within an organisation draws on the understanding of the tacit aspects of the domain. Tacit aspects of an organisation can be any type of information or domain knowledge about business data or entities in an organisation. Tacit information is hard to identify and convey and is often the reason why it is missed when conceptual data models are being formalised.

In addition, the very nature of tacit information makes it hard to formalise. There are several formal notations that are good at capturing process activities and representing the architectures of systems. In comparison to ethnographic reportage, there are things that current notations will not pick up that ethnographic report would be able to because of the benefit of documentation in free text without the restrictions of a formal language or notation.

The purposeful planning of ethnographers to identify tacit aspects of an organisation reduces the potential of omitting this information in the requirements analysis phase when conceptual data models are being developed. The fact that observers are sent out into the field of study makes them prime candidates to become acquainted with the tacit constructs of an organisation. Observers would be able to uncover organisational behaviours that would often be missed during the standard types of meetings that developers and requirements analysts have with stakeholders.

4.4.2 Ethnographic Elicitation for Data

Ethnographic reports have some interesting characteristics from the perspective of data modeling. Ethnographic reports provide an alternative way of documenting usage. They contain fuzzy use-cases and are very detailed and precise. This may be particularly useful in determining how entities and records are used within the system, and the relationships and associations between them.

From a data analysis perspective, the main function of ethnography as a mode of inquiry is to understand *Forms of Data*. This refers to the composition, characterisation and context of use of data. The composition of data is not limited to its explicit attributes and its general constitution. It also includes any implicit properties and attributes that are either inherited through association with other Forms of Data, or inherited via semantic or domain-centric relationships.

If we were trying to assess the proficiency of UML in being able to transfer data modeling requirements to development, by determining what can and can't be modelled in UML, one place to look at is form of data. It is one of the

areas in which UML is used extensively - class diagrams, data transformation using state charts etc.

Methods

In this section, we discuss two main methods of data collection in the execution phase of the Ethnographic Study Life Cycle (ESLC) that aid in understanding Forms of Data in a domain.

- Field Observations: This is a very common data collection method. For data analysis, the process begins with developing a set of objective research questions:
  - What are the implicit and explicit data entities, and relationships between these entities?
  - What is the context of use of data in the domain?
  - What are the boundaries of data?

  Answers to these questions can be gleaned from observing what pieces of information users interact with and how data flows through the organisation. Furthermore, looking at the types of information exchanged between business users can reveal what the data entities in the environment are and what the boundaries of data are. Finally, analysing storage mediums and stored records will enable the ethnographer to develop an understanding of what the entities, attributes and relationships in the organisation are.

- Key Participant Interviews: This is a widely used part of field research. It involves planned meetings with business stakeholders to gather information to meet the data collection objectives of fieldwork. Key participant interviews may either be (i) structured or (ii) unstructured.

  Structured interviews are formal, and often involve using a set of preplanned questions and material which may be reused and presented to other participant interviewees. Structured interviews are important for gathering consensus amongst a group of key stakeholders.

Unlike structured interviews, unstructured interviews are quite informal and casual. However, they are still guided by the aims and objectives of the study. They are important for open ended investigations of particular areas of interest to the ethnographer.

4.5 The Medical Records Case Study (Pt. 1)

Central to our work is the question of whether tacit information about data obtained from ethnographic studies facilitates or undermines the quality of information represented in conceptual data models built at a requirements level. In this section, we investigate what types of information can be understood about a data context when utilising ethnographic observations. The medical records case study by Heath and Luff [42] will be used as the problem statement in this section to motivate the viewpoint of ethnographic software requirements analysis. As such, the case study will take an ethnographic perspective.

The study investigated the work practice in primary health care within the United Kingdom and probed the inclination of practitioners using paper records in spite of the 'Value-Added Medical Products' (VAMP) computerised records system already provisioned to replace traditional paper medical record cards. The findings of the case study highlighted some flaws in the design choices of the provisioned system due to the ways in which various design elements were embodied in the new system. The authors concluded a flawed design. There were considerable misgivings due to the way in which the system constrained the examination and entry of data by practitioners.

4.5.1 Overview

As outlined in the study, the traditional paper medical record consisted of an A5 envelope comprising a number of cards and pieces of paper, including but not limited to referral and discharge letters, doctors notes, medical test results etc. On the envelope itself, the patients personal details were written. These details included the patients name, address, date of birth and their National

Health Service (NHS) number.

The information stored in a patients' medical record mainly consisted of consultation notes with every new consultation requiring a new consultation note entered into the patients medical record card. Below is an example of a consultation note, taken from the case study.

3/12/86 c. Dog bite
Rf (.....)
Tetanus Toxoid 0.5 ml

The information recorded in the consultation note consisted of a date, the location of the consultation, a record of the complaints and illnesses reported by the patient and the GP's diagnosis. The information recorded in a typical consultation note consisted of a number of elements. Heath and Luff classified these into elements relating to: (i) the occasion of the consultation - corresponding with the date and location (*c.* for consultation being held in the surgery, *v.* for a home visit); (ii) the complaint or illness - corresponding with the patients symptoms or the doctors diagnosis; (iii) the management of the complaint - corresponding with the doctors treatment, referrals and so on.

Important background notes

i) The patients medical record was stored at the patients *registered* general practice.

ii) The medical record followed the patient. All patients were required to have a single medical record that was not duplicated or fragmented between multiple medical practices.

iii) On the topic of information sharing, it was not immediately clear how the cards were shared between GPs from different medical practices - for example, in the case of a medical referral or relocation of the patient between medical practices. However it was been noted in the study that the medical record card was made available to doctors whenever there was a consultation with a patient.

iv) Only the death of a patient could result in the deletion of the patients medical record. However, the record may have continued to persist for up to six months after the patients death, should any contingencies or enquiries about the patient arise.

From a data analysis perspective, it is clear that a patients medical card is an important source of data, for accessing information pertaining to the patients health, as it provided a medical history which includes an account of the patients previous consultations, illnesses, diagnoses, treatments and referrals. It is therefore a vital piece of information that requires good accuracy and accessibility.

4.5.2 Ethnographic Analyses

The analysis of the VAMP system highlighted a number of flaws in the design choices of the designers of the medical records system due to the ways in which various design elements were captured and embodied in the system. Drilling down into the ethnographic report presented in the case study makes it more apparent as to why a data model based on explicit information will not be appropriate for the VAMP system. Our understanding of requirements analysis for data at this level stems from our notion of 'Forms of Data'.

One important observation that was made was the use of inference in the medical record to derive information. For example the proximity of dates between two consultation notes inferring a relationship between both consultation entries. This could either be through a symptom relationship, diagnosis relationship or treatment relationship. The presence of 'proximity' information on a CDM through one of the relationships described previously, may form the basis for deciding to choose a particular kind of user interface for viewing identical or homogeneous pieces of information over an interface for viewing unconnected information. To elaborate on our discussion on this topic, we draw some extracts showing various patient records from the case study, as shown in Table 4.1

The records in Table 4.1 may appear brief and unsystematic, however the expertise of the medical practitioners allowed them to use the information

24.2.95	a.	all tests normal feeling low tearful fragile start prozac see 10d for inj R tennis elbow
11.8.95	c.	Not good- irritable Pain better, but drowsy in morning restart diary
28/9/85	c.	Vomited x2 in night Maxalon 10bd (300m)
22.9.95	a.	Diary OK for 1 wk R elbow pain -less tender ˆ Doth to 150
12/1/86	v.	Died 12.30am
3/12/86	c.	Dog bite Rf (.....) Tetanus Toxoid 0.5 ml
22/4/86	c.	cold also rheumatism cert 1/52 Paracetamol

Table 4.1: A sample of a patient's Medical Record

proficiently to carry on their practice. As the extract shows, a single entry can contain various kinds of information linking both the patients diagnosis, prognosis and treatment. It is not immediately clear which record attributes are required to be recorded besides the date and the location of the consultation. It would seem that it is determined by what the GP deems necessary to be logged in the medical record and not necessarily in conformation to any rules regarding the documentation of medical texts. There is also no particular order in which symptoms, treatments and diagnoses are recorded.

From a data analysis perspective, it is worth considering how various key components of a medical record should be defined, and any possible relationships

or associations between the attributes that it is composed of. With that in mind, we look at some key elements of medical records:

i) Consultations: consist of a meeting between one or more medical practitioners and a patient to discuss a medical complaint. At every consultation, a new consultation note is created to document the details of the consultation. This implies a many-to-one relationship between a patients consultation records and the same patients medical record. Consultation notes go into the patients medical record, to log the patients medical history of: (i) cases dealt with (ii) referrals (iii) treatments and so on. As it is possible for multiple cases to be discussed during the same meeting, multiple diagnoses, prognoses and treatments can be issued within the same meeting as is the case with the following example:

22/4/86	c.	cold also rheumatism cert 1/52 Paracetamol

ii) Diagnosis: A medical diagnosis is the identification of a medical condition or problem determined by known symptoms in a patient. For documentation, diagnoses were recorded in the patients consultation notes during each consultation. In the case study, it showed that not all consultations notes contained diagnoses explicitly recorded within them. However, this did not imply that a condition was not diagnosed. What was observed were deductions and inferences made by practitioners based on the presence or absence of the diagnosis attribute. This highlighted the use of tacit information, in the skill of the medical practitioner being able to deduce key pieces of information even when there was an omission of the data. As an example, to a competent medical practitioner, the following consultation note showing the symptom and treatment might imply a digestive disorder in the patient. However the diagnosis is not explicitly recorded:

iii) Treatment: This is a remedy to an illness that has been diagnosed during a medical consultation. As with the example from the previous point,

28/9/85 c. Vomited x2 in night
Maxalon 10bd (300m)

the treatment prescribed for the associated symptoms is 'Maxalon 10bd (300m)'. The flexibility of the medical records system did not prevent a record of treatment from being deferred to a later date. For example 'prescribe painkillers if pain continues'. It was also possible that treatments were not recorded explicitly and may link to treatments from previous consultations as is the case in the following example:

11.8.95 c. Not good- irritable
Pain better, but drowsy in morning
restart diary

The phrase *restart diary* indicates that the patient should follow the same course of treatment prescribed during the preceeding consultation. This creates a tacit relationship with the preceding consultation note.

Heath and Luff argued that "the various items which constitute entries there do not have a fixed and determinate sense". This was true for a number of behaviours of some of the sample consultation note extracts. Each of the extracts above has shown occasions where information in the notes was omitted. However practitioners were still able to derive various pieces of information about data that was not included in the consultation note based on information that was present. This was possible through an understanding of the tacit knowledge of the domain. In the next section, we outline some of the tacit observations of the system that one might consider when modeling the system.

4.5.3 Observations

The case study outlined a number implicit associations and relationships between records. Observations of consultation notes showed patterns of inference through which preceding consultation notes in a patients medical record might be relevant to the current consultation at hand. We define these 'relevancy' relationships through occurrence relationships and relationships pertaining to the consultation attributes - symptom, treatment and diagnosis.

The 'applicability' of the preceding notes is understood to be important from the perspective that their symptoms, diagnoses and treatments have some relevance to the current consultation note. These relationships are described below:

i) Date Relationship: A date relationship exists between consultation entries in a patients medical record when the consultation entries fall within a time window of a few days to a week. This time window may be adjusted on demand by the medical practitioner. The occurrence of records that have this relationship commonly fall within last few consultation notes from the current consultation at hand.

   Domain Implication: The domain implication of this relationship implies that the current problem(s) or illness(es) being reported by a patient may have some connection with previous problems reported during previous consultations within the specified time window (or at least, the last few consultation entries).

ii) Symptom Relationship: A symptom relationship exists between consultation notes where the 'symptom' attribute between two consultation entries contains identical or closely similar data. The similarity of the attributes between consultation notes is one that is understood through the expertise of the medical practitioner.

   Domain Implication: This relationship implies that consultation entries with identical or similar symptoms often relate to the same issue or problem reported by the patient. Therefore, the diagnosis and treatments of the problem may be similar. A symptom relationship combined with a date relationship on a pair of consultation entries, may imply the persistence or progression of a medical problem.

iii) Diagnosis Relationship: A diagnosis relationship exists where the 'diagnosis' attributes between two consultation entries are identical or closely similar. This similarity is also understood by the expertise of the medical practitioner.

Domain Implication: A diagnosis relationship between consultation notes implies a connection between medical symptoms reported, and may infer suggestions for treatment.

iv) Treatment Relationship: Similar to the symptom relationship and diagnosis relationship, the treatment relationship exists where the 'treatment' attribute between two consultation notes is identical or similar.

Domain Implication: A treatment relationship may imply that the diagnosis between each note was the same. The understanding is that the prescribed treatment may relate to a problem reported by the patient on a previous occasion.

4.5.4 Conclusion

Based on our observations and analysis of the case study, we have identified tacit information that was not found in the VAMP system. We posit that if ethnographic techniques are utilized during requirements analysis, then in addition to explicit information, two aspects of records will be elicited which are important to the notion of a record:

a) a set of observations detailing implicit forms of information encoded within a record.

b) a set of observations detailing the mode and context of use of information

Each of these aspects of records were visibly overlooked in the VAMP system. According to observations, the VAMP System did not incorporate tacit information that considered relationships between consultation notes attributes, in particular symptom, diagnosis and treatment. As such, important requirements guidelines were not communicated to the design phase of the system. Consequently, this led to a software deliverable that was inadequate. Specifically, UI design choices were were not optimal for use by medical practitioners.

4.6 Summary

The problem of notation in requirement analysis is well known [4]. This is especially problematic when considering tacit information and how tacit information should be communicated within the SDLC. There is an understanding that tacit information is an important dimension of data in an organisation. Accordingly, in this chapter, we have discussed how ethnography can be used to uncover tacit information. The aim of identifying tacit information draws on the knowledge that gaining a better understanding of the domain, will lead to a more effective set of requirements, which will hopefully lead to design decisions that produce a product that meets the clients requirements. This however, all begins with modeling tacit information in the domain at the conceptual level.

The medical records case study by Heath and Luff [42] was used as a motivating case study to illustrate how both tacit information and explicit information are divulged in a data context. Also discussed was why tacit information may be a relevant dimension of data to consider when designing systems.

The next chapter takes the position that tacit information should be made explicit in the conceptual data model of a system, as it provides a more accurate model of domain. We pursue our contention that this will lead to a software deliverable that more closely meets the requirements of stakeholders. As companion, we investigate how tacit information can be formally incorporated into the SDLC of software systems from a Model Driven Development (MDD) perspective. We therefore utilise Model Driven Engineering (MDE) techniques to build a language for this purpose.

5

A Tacit Requirements Metamodel

> "A significant factor behind the difficulty of developing complex software is the wide conceptual gap between the problem and the implementation domains of discourse" *France et. al* [29].

Conceptual Data Models (CDM) provide a shared understanding of the business domain of a system. In this chapter, we propose a rich CDM metamodel that is able to impart both explicit and implicit information obtained via Pre-Implementation Ethnography in view of bridging the communication gap between requirements and design. The chapter begins by introducing the notion of a *Tacit Contract*, which is a design level obligation aimed at enhancing requirements-level data schemas by highlighting implicit information in a CDM that a designer must consult in the subsequent phases of the SDLC. A CDM metamodel is proposed for this purpose. Finally, we present a formalisation of the CDM metamodel alongside a case study as a motivating real world example that uses our modeling language to construct its CDM.

5.1 Tacit Contracts in Requirements Analysis

The primary aim of data modeling is to identify entities in the problem domain and possible relationships and associations amongst these entities. A large and increasingly growing body of research continues to support model and data-driven approaches to software development. The process of data modeling

fulfils a number of purposes in software engineering: from visualising high-level conceptual models to detailing with high complexity, the structure of the storage data schemas via physical data models.

The beginning of a typical data modeling task starts with conceptual data modeling in the requirements analysis phase of the SDLC. There are no constraints or limits on the expressiveness of Conceptual Data Models in the field of data modeling. For this reason, there are several possibilities in terms of scope when looking purely at what a CDM captures. From the perspective of describing the structure of a domain, it would be beneficial for a *rich* CDM to capture all the dimensions of data that characterize a domain.

Indeed, at the requirements level, tacit information can be maintained adequately through comments and annotations on a data model. However, such an approach, on its own, offers no guarantee that key forms of tacit information will be carried across into design.

While a rich CDM that contains both explicit and implicit information may be beneficial at the design phase of the SDLC because of its comprehensive nature, the level of detail may be overwhelming to even the most experienced data analysts. In addition, there is the challenge of correctly discriminating between implicit and explicit information in the model. For this reason, it may be useful to provide a way of communicating what aspects of the CDM concern explicit properties of the model, and aspects that concern implicit properties which must be acknowledged when moving from the Conceptual Model to the Logical Model.

We introduce the notion of a *Tacit Contract*, to specify the categories of information in a CDM that are an implicit nature. More formally, a Tacit Contract is a design element that is found in a requirements deliverable such as a Conceptual Data Model that distinguishes elements of a CDM that are of an implicit kind, from explicit elements. By 'design elements', we are referring to components of the CDM which include entities, attributes, relationships and so on.

The purpose of a Tacit Contract is not just to make the distinction between different kinds of information in the CDM, it is also intended to be used to communicate the importance of design elements that are not explicit, and

add meaning to explicit elements. What this does between the requirements and design phase, is communicate a correct representation of the model across the phases by creating a design obligation between the requirements analyst and the designer. For example, it may be important to visually express time sensitivity in a model between entities by incorporating a temporal relationship in the CDM. This convention creates design obligations for the designer and ensures a consistent understanding that implicit elements in a CDM are understood to be tacit.

5.2 Towards A Tacit Requirements Metamodel

While requirements stage data schemas represented as CDM's, can be captured adequately with a classifactory notation such as UML class diagrams, UML alone (or comparative notions such as E-R diagrams) are not suitable to capture tacit forms of information. This is due to the fact that UML, E-R diagrams and so on, are not directly intended to be used to represent implicit information. What would normally happen is schemas will be represented in classifactory notations, and tacit information would normally be represented by use cases or extra notations. The problem with such a method is that these extra notations are not guaranteed to be carried across to design.

In this section, we propose a tacit requirements metamodel that is used to construct CDM's that permit both implicit and explicit forms of data. This is done via Tacit Entities, Attributes and Relationships. The abstract syntax of the metamodel is presented using the class diagram shown in Figure 5.1[1].

[1]Figure 5.1: Metamodel for defining Comprehensive Conceptual Data Models (Page 97)

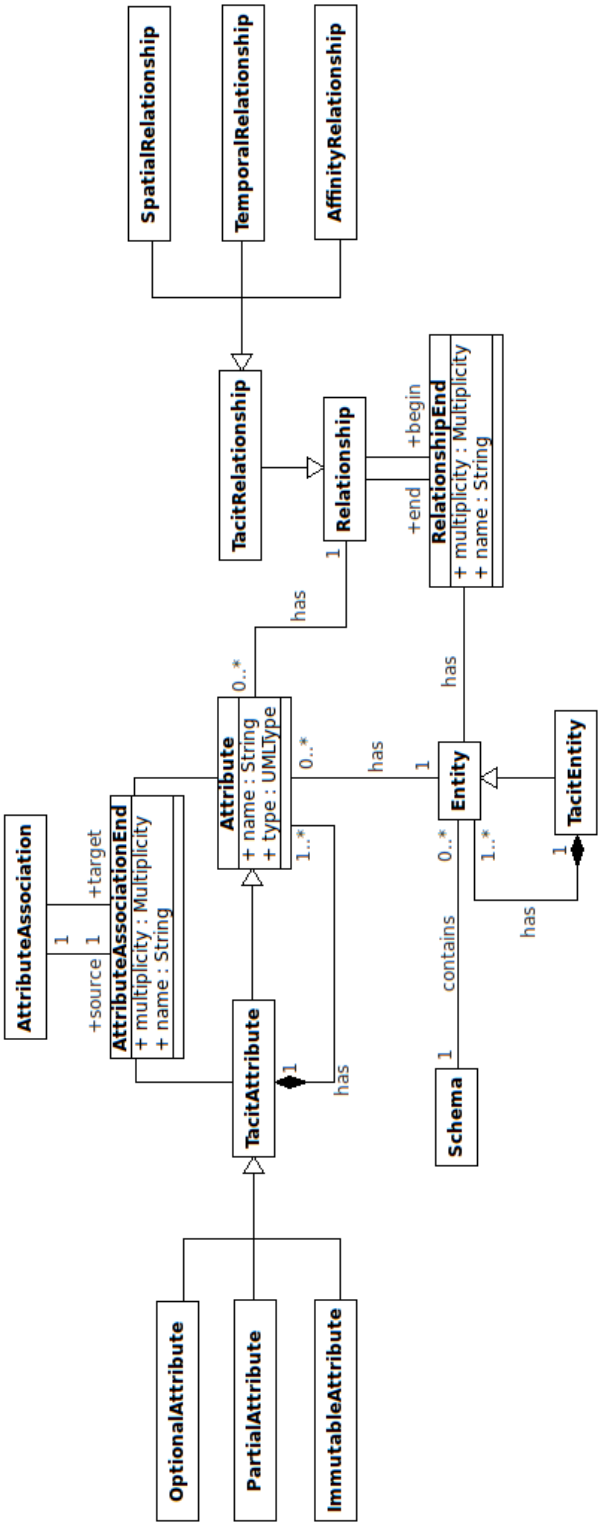


Figure 5.1: Metamodel for defining Comprehensive Conceptual Data Models

Schema is the root element in the metamodel. An instance of Schema *contains* 0..* Entity instances. The Entity metaclass allows us to define data concepts within the schema. Data concepts in the model can either be explicit or tacit. Explicit data types are instances of Entity which is super-class of TacitEntity. The TacitEntity metaclass allows us to create entities based on implicit information. The composition relationship between TacitEntity and Entity allows the creation of more complex structures.

RelationshipEnd and Relationship allow us to define associations between Entity instances. They also allow us to define associations between Attribute instances. Relationship is a super-class of TacitRelationship which is a generalisation of a number of tacit relationships: (i) SpatialRelationship, relating to the space between Entity or Attribute instances (ii) TemporalRelationship, relating to 'time' based relationships between Entity instances; and (iii) AffinityRelationship, relating to the likeness between Entity instances.

Moving back to the Entity metaclass, the metamodel permits entities, and relationships to each have attributes. Therefore there is a *has* relationship between Entity and Attribute, and the same between Relationship and Attribute. Each of these associations also hold a multiplicity of 0..* on the relationship end of the Attribute metaclass.

The Attribute metaclass is a super-class of TacitAttribute. TacitAttribute has a composition relationship with the Attribute. TacitAttribute is a generalisation of a number of tacit attributes in our model: OptionalAttribute, PartialAttribute and ImmutableAttribute. AttributeAssociationEnd and AttributeAssociation allow for defining relationships between Attribute types and TacitAttribute types.

The model is generic in that it does not strictly specify components in the CDM. If a new domain specific CDM component is needed, the appropriate metaclass can be extended.

Model Notation

UML notations are important. They enable models to communicate the semantics of the domain. UML Notations can be adapted to fit the context of a particular domain through various extensibility mechanisms. Different

notations are used to serve different purposes. For example, *Class Notation*: used to define the structure of domain objects in a system; *Object Notation*: used to represent actual instances or implementations of domain objects; *Use Case Notation*: used to capture system behaviours and actions of entities in the domain, and so on.

We propose a syntax for the modeling language that will be used to mark up and communicate the tacit semantics of the domain in the CDM. This is done by using one of the extensibility mechanisms of UML which permits extending the visual syntax of UML class diagrams.

The visual syntax of UML class diagrams can be extended via a specialization profile, consisting of "stereotypes" [14]. A stereotype is an element that provides a way of refining the meaning of an element in a UML model. Any visual element of UML can be equipped with a refined, specialized semantics by means of a stereotype annotation.

We utilise a profile of UML that enables us to explicitly identify implicit information within our model. Model elements annotated with the tacit stereotype enable us to distinguish tacit information from the explicit data schema itself. We prescribe any visual element of the UML class model, in particular classes, attributes, associations and so on, with the *tacit* stereotype, which is our specialization semantics. The stereotype is written as follows:

$$<< \boldsymbol{tacit} >>$$

The convention is that the requirements model is annotated with this stereotype, where the model holds identifiable elements that are tacit and distinguishable from elements in the model that refer to constructs of the explicit data schema. In the next section, we describe a formal semantics of the metamodel.

5.3 Constructive Types for Pre-Implementation Ethnography

> "One of the things which the discussion of ethnography might be instrumental in starting, perhaps, is an examination of the dogma of formalism and the proper scope of design sensibilities" *Anderson* [5]

Given the openness of the metamodel to be able to prescribe models that incorporate multiple dimensions of information, in particular explicit and implicit information, there is an important requirement for the choice of formalism for defining the 'open' semantics of the metamodel. The formalism must be capable of representing the naturalistic view of Pre-Implementation ethnographic findings. For this reason, we employ Constructive Type Theory (CTT) as the formalism for defining our model semantics.

Constructive Type Theory is an open approach to constructing rich type systems [65] [60]. It is most often used as a means of describing system functionality, specifying logical properties of systems, reasoning about systems and tracing their execution. Constructive type theory can also be employed as a framework for defining logics, with each new logic defined by its own set of rules, behaviours and semantics. The approach is based on Martin-Lof's idea of formal type theories for mathematics [65]. Martin-Lof envisaged mathematics as a kind of 'open system'; like a game in which the mathematician defining the system is free to invent new rules that establish new ways of doing maths. For example, a mathematician might have a set of rules for dealing with natural numbers. When a new rule is required to work with other numbers, for instance rational numbers, a new set of rules is added.

Background & Notations

Constructive Type Theory is understood through the reasoning of the notion of types. The notion of a *type* extends that of the theory of *sets* in constructive mathematics. Types are formed by specifying the criteria for their construc-

tion. Constructive Type Theory is based on Constructive Mathematics where a proposition P is determined by the type of its proof p. This can be written as follows:

$$\mathsf{p} \in \mathsf{P}$$

The statement reads 'p is a proof of the proposition P', or 'p is an inhabitant or term of P' etc. The concept of a proof being an 'inhabitant' of its proposition allows for possibility of multiple proofs of the proposition. For instance, let $\mathsf{K} = \{\mathsf{p}, \mathsf{q}, \mathsf{r}\}$ be a set of proofs of the proposition P. Any of the proofs p, q or r are considered 'inhabitants' of P. Therefore the following statements are valid: $\mathsf{p} \in \mathsf{P}$, $\mathsf{q} \in \mathsf{P}$, $\mathsf{r} \in \mathsf{P}$.

In Martin-Lof's philosophy of type theory, we can prove P, by creating an object of type P. This follows the principle of mathematical constructivism, which asserts that the proof of an object is in its ability to be instantiated. Martin-Lof's notation **p : P** is different but it shares the same behaviour as $\mathsf{p} \in \mathsf{P}$ in meaning that p is a proof of the proposition P.

The expressiveness of CTT creates the possibility to be specific about certain properties and traits of a system through the formulation of axioms and rules of the system. From a requirements engineering perspective, this can enable a systems analyst to write structured rules about the specification of an information system, and specifically for data modeling concerns. CTT is a richer and more expressive language than other logics. We are able reason about systems, given a sets of use cases or systems rules that need to be formalised.

Reasoning about Systems

In the formalism that we are employing, when reasoning about systems, there is no "objective" truth. Objective truth is a set of statements that are independent of perception or theory. In this formalism, we say that a truth of a 'thing' is verified by a logical statement about it. For example, we write:

$$\mathbf{World} \models \phi(\mathbf{F})$$

This statement reads: 'World verifies the logical statement F'. As is illustrated in the statement description, $\models$ means *verifies* (or *checks*), F is some logical statement, World is a description of some context (for example, the classical truth for a system). As an example, let S represent a system and F_s represent the specification of the system. This statement can be written as follows:

$$S \models \phi F_s$$

This is the same as saying the system S conforms to its specification F_s (use-cases, design models and so on).

In Martin-Lof's notion of logical calculus and his concept of what it means to judge a truth, every judgement in S consists of a term t and a proposition P, such that $t : P$. The t allows us to navigate the way in which the current proposition has become true. The term t should be thought of as a trace which gives us an audit trail of how that fact has come to be. This is of course determined by the rules of the system that permit the term to be instantiated.

In general, the term t will encompass a sequence of functions (f_0, f_1, f_2, ..., f_n) that operate over basic business entities and data types that are represented in the model. This enables us to create rules over entities that are implicit or explicit.

Using terms and functions, and being able to operate over business entities and data types, we are equipped with the flexibility and control needed to define use cases that involve:

a) Relationships between both explicit and implicit data types

b) Modes of use, and context of use

Rules

We consider a number of necessary concepts that will enable us to define the rules of a system:

a) Introduction rules allow us to define conditions under which a new proposition holds from an old set of circumstances

b) Elimination rules allow us to use current knowledge to form new knowledge of a different kind.

In addition to these rules, there are also general rules for the "entities" and data types employed in a system specification. This is more like standard Martin-Lof type theory, or even defining classes in Java. For example:

$$\frac{}{\text{Person is a type}}\ \text{(Formation Rule)}$$

$$\frac{\text{n:string} \quad \text{p:string} \quad \text{a:int}}{\text{Person}\{\ \text{name} = \text{n};\ \text{postcode} = \text{p};\ \text{age} = \text{a}\} : \text{Person}}\ \text{(Constructor)}$$

5.4 The Formalism for the CDM

In this section, we present the formalism of the tacit requirements metamodel[2]. In the formalism description, we will refer to the metamodel with the symbol M. The formalism consists of a list of elements where E is a metaclass type and e_1, e_2, ..., e_N are elements in the metamodel M.

We first define the atomic propositional structure of the formalism over model elements. This is expressed as follows:

$$\mathbf{Rel}(e_1, e_2, ..., e_N)$$

Definitions:

i) Rel is a predicate over the elements e_1, e_2, ..., e_N in the metamodel M.

ii) Rel can be a relationship in the metamodel operating over a set of elements of the model such as entities or attributes.

iii) The language contains a set of logical operators: $\vee$, $\wedge$, $\Rightarrow$, $\neg$.

For each of the logical operators, we have a number of *introduction rules*:

$\vee$ Rule

The $\vee$ rule states that if we have a proof of A or a proof of B, we can introduce $A \vee B$, which reads 'A or B'.

$\vee$ Introduction Rule

$$\frac{a : A}{inl(a) : (A \vee B)} \ (\boldsymbol{I_1} \vee) \tag{5.1}$$

$$\frac{b : B}{inr(b) : (A \vee B)} \ (\boldsymbol{I_2} \vee) \tag{5.2}$$

[2]Refer to Figure 5.1 on Page 97 for the metamodel

∨ Elimination Rule

$$\frac{\mathsf{a} : (\mathsf{A} \vee \mathsf{B})}{\mathsf{fst(a)} : \mathsf{A}} \ (\boldsymbol{E_1} \vee) \tag{5.3}$$

$$\frac{\mathsf{b} : (\mathsf{A} \vee \mathsf{B})}{\mathsf{snd(b)} : \mathsf{B}} \ (\boldsymbol{E_2} \vee) \tag{5.4}$$

∧ Rule

The ∧ rule states that given a proof of A and a proof of B, we can introduce A ∧ B, which reads 'A and B'.

∧ Introduction Rule

$$\frac{\mathsf{a} : \mathsf{A} \quad \mathsf{b} : \mathsf{B}}{\mathsf{pair(a,b)} \mathbf{:} \ (\mathsf{A} \wedge \mathsf{B})} \ (\boldsymbol{I} \wedge) \tag{5.5}$$

where pair(a,b) is a function defined over the terms a and b.

∧ Elimination Rule

$$\frac{\mathsf{a} : \mathsf{A} \quad \mathsf{b} : \mathsf{B}}{\mathsf{fst(a)} \mathbf{:} \ \mathsf{A}} \ (\boldsymbol{E_1} \wedge) \tag{5.6}$$

$$\frac{\mathsf{a} : \mathsf{A} \quad \mathsf{b} : \mathsf{B}}{\mathsf{snd(b)} \mathbf{:} \ \mathsf{B}} \ (\boldsymbol{E_2} \wedge) \tag{5.7}$$

$\Rightarrow$ Rule

The $\Rightarrow$ rule states that if we have a proof of B which has A amongst its set of assumptions, we can introduce $\mathsf{A} \Rightarrow \mathsf{B}$, which reads '$\mathsf{A}$ implies B'.

$\Rightarrow$ Introduction Rule

$$\frac{\begin{array}{c}\mathsf{a} : \mathsf{A} \\ \vdots \\ \mathsf{b} : \mathsf{B}\end{array}}{\mathsf{f}_{\Rightarrow}(\mathsf{a},\mathsf{b}) : (\mathsf{A} \Rightarrow \mathsf{B})}\ (\boldsymbol{I}\ \Rightarrow) \tag{5.8}$$

$\Rightarrow$ Elimination Rule

$$\frac{\mathsf{a} : \mathsf{A} \quad \mathsf{p} : (\mathsf{A} \Rightarrow \mathsf{B})}{\mathsf{b}\mathbf{:}\ \mathsf{B}}\ (\boldsymbol{E_2}\ \wedge) \tag{5.9}$$

$\neg$ Rule

The $\neg$ rule (or not rule) states that if we have a proof of A and $\neg\mathsf{A}$, this leads to a contradiction or absurdity.

$$\frac{\mathsf{a} : \mathsf{A} \quad \mathsf{b} : \neg\mathsf{A}}{\mathsf{absurdity(a)}\mathbf{:}\ \mathsf{B}}\ (\neg) \tag{5.10}$$

where B is anything. This means that we have a contradiction. Our inference system can conclude anything, as if it is corrupted.

Application Rule

The application rule states that terms can be combined to form new rules.

$$\frac{\mathsf{b} : \mathsf{E_1} \quad \mathsf{a} : (\mathsf{E_1} \Rightarrow \mathsf{E_2})}{\mathsf{app(a,b)}\mathbf{:}\ \mathsf{E_2}}\ (\boldsymbol{E}\ \Rightarrow) \tag{5.11}$$

Metamodel Schema Template

The formalism for the metamodel can be expressed via the following rule:

$$\frac{\mathsf{e_1} : \mathsf{E_1} \quad \mathsf{e_2} : \mathsf{E_2} \quad ... \quad \mathsf{e_N} : \mathsf{E_N}}{\mathsf{f}(\mathsf{e_1}, \mathsf{e_2}, ..., \mathsf{e_N}) : \mathsf{R}(\mathsf{E_1}, \mathsf{E_2}, ..., \mathsf{E_N})} \quad (5.12)$$

The metamodel schema template reads as follows: given the set of proofs and their types $\mathsf{e_1}$:$\mathsf{E_1}$, $\mathsf{e_2}$:$\mathsf{E_2}$, ..., $\mathsf{e_N}$:$\mathsf{E_N}$, the following statement can be concluded: $\mathsf{f}(\mathsf{e_1}, \mathsf{e_2}, ..., \mathsf{e_N}) : \mathsf{R}(\mathsf{E_1}, \mathsf{E_2}, ..., \mathsf{E_N})$.

The function $\mathsf{f}(\mathsf{e_1}, \mathsf{e_2}, ..., \mathsf{e_N})$ (hereafter, we refer to this as F) is a function over the terms and $\mathsf{R}(\mathsf{E_1}, \mathsf{E_2}, ..., \mathsf{E_N})$ (hereafter, we refer to this as R) is a relation over the types in the pre-condition. F corresponds to a use case of the system and R corresponds to explicit or implicit relationships or formulaes over the Types of the terms provided. This can be read by saying that use cases are terms of data relationships in the system.

We follow Martin-Lof's theory which permits new rules to be defined, to tell us when combinations are valid. Let $\mathsf{e} : \mathsf{E}$ denote that e is a term of E where E is metaclass instance of the metamodel M.
For example, take E to be a metaclass instance and R to be an instance of TacitRelationship. The proof of R is a function over the proofs E.

$$\frac{\mathsf{e_1} : \mathsf{E} \quad \mathsf{e_2} : \mathsf{E}}{\mathsf{f}(\mathsf{e_1}\mathsf{e_2}) : \mathsf{R}(\mathsf{E})} \quad (5.13)$$

Looking more broadly, elements E in the model M, may represent any conceptual type according to the metamodel (Figure 5.1[3]). For example, an arbitrary conceptual type $\mathsf{E_N}$ can be an instance of: Entity, Attribute, Relationship. Implicit types may also be represented using the *Tacit* specialisations of their explicit counterparts: TacitEntity, TacitAttribute and TacitRelationship.

This modeling approach is intended to be employed by an analyst whose

[3]see page 97

task is to model observations derived from Pre-Implementation Ethnography leading to the establishment of a set of rules. If an unusual situation is observed, the analyst is free to add new introduction, elimination and formation rules to accommodate it. As the system evolves, rules may be modified as required.

5.5 Why Constructive Type Theory?

There is the question over the use of Constructive Type Theory, and not the more widely used means of specifying requirements and rules such as use cases. Type theory gives us all the benefits of an open and extensible formal system. Rules can be added and deleted on demand. This extensibility feature is unusual for a formal system, and semantically we are allowed to do this in CTT.

From a requirements point of view, the rule system can mirror system requirements even through evolutionary phases. For instance, when stakeholders introduce new requirements or when existing requirements are modified. In addition to this, CTT permits the literal definition of rules and facts. It allows the analyst to correctly specify use cases, usage scenarios and data relationships between entities.

Consider the following statement about an employment contact policy: 'An employee has an employment contract with their employer'. In this statement, Employee, Employment Contract and Employer represent entity types. For simplicity we will model Employer as the type Company, and Employee as the the type Person. The rule can thus be written as follows:

Let Employs(A,B) be the predicate over two terms A and B of type Company and Person respectively, denoting that A employs B.

$$\frac{c : \mathsf{Company} \quad p : \mathsf{Person} \quad e : \mathsf{Employs}(c,p)}{q : \mathsf{EmploymentContract}(c,p)} \qquad (5.14)$$

The term q is the proof term of the type EmploymentContract(c,p). An additional rule can be added to say that, if a person has an employment contract with an employer, that person is employed.. Let IsEmployed(A) denote that the A is employed, where A is of type Person.

$$\frac{q : \mathsf{EmploymentContract}(c,p)}{e : \mathsf{IsEmployed}(p)} \quad (5.15)$$

Through logical inference, rules 5.14 and 5.15 allow the following derivation: 'A person is considered to be employed if they have an employer'.

$$\frac{c : \mathsf{Company} \quad p : \mathsf{Person} \quad e : \mathsf{Employs}(c,p)}{t : \mathsf{IsEmployed}(p)} \quad (5.16)$$

This can also be written as shown in rule 5.17 and illustrated in the metamodel instance in Figure 5.2.

$$(c : \mathsf{Company}) \wedge (p : \mathsf{Person}) \wedge (e : \mathsf{Employs}(c,p)) \Rightarrow \mathsf{IsEmployed}(p) \quad (5.17)$$

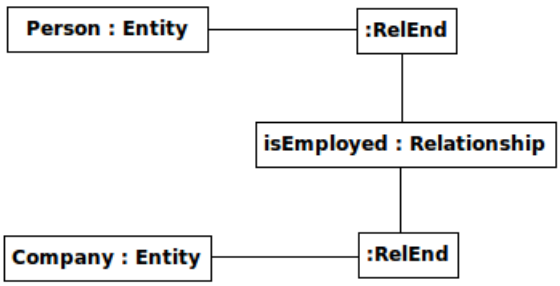


Figure 5.2: Metamodel Instance Example of Employment Contract

Formalising Rules

The formalisation step focuses on extracting a particular set of requirements of the form typically found in requirements analysis methods: specifying the

main use cases, identifying business objects, entity relationships and so on. These tasks are typical of most RE methods. We gain significant advantage from using CTT as the means of representing these requirements:

a) Openness and extensibility: The meaning of any symbol (or object) is given by the rules of its construction – how it forms a concept, or how it forms data. Every particular system (mathematical or otherwise) is determined by a set of rules that are open to addition/deletion. Our method allows us to create a new logical system (a new logic) for every information system, rather than using a pre-defined logic to do so. This alleviates some of the issues encountered where a particular system is constrained to using existing logics that may have been generalised to solve particular types of modeling problems.

b) Tracability: This is a novel notion of data in which an information system is seen to be a means of manipulating concepts and information – the semantics comes through the traces. Traces are created when we build application rules over terms in the system. A concept is a logical specification of how things stand currently and can include predicates over values. A concept's value or validity is given by its contextual history, that is, by a *trace* of how information has been processed to "construct" or "support" that concept. Concepts are seen as retaining an individual persistence, but their support or construction is mutable over time. Essentially, it enables us to audit and determine how we arrived at a given fact by means of functions over basic entities as types. These functions correspond to applications of use cases. Therefore it provides a trace over use cases.

5.6 The Medical Records Case Study (Pt. 2)

As a motivating example of formalising the rules of a system using CTT, we revisit the Medical Records Case Study by Heath and Luff [42] that was introduced in Chapter 4[4]. This section begins by introducing what we view as

[4]Medical Records Case Study Pt. 1: Chapter 4, Section 4.5, page 85

a 'Naive Conceptual Model' (NCM) which is a model of a domain that is constructed solely on explicit information. Following this, a more comprehensive 'Tacit Conceptual Model' (TCM) is introduced alongside a formalisation of its rules in Constructive Type Theory. Finally, a comparison of both the NCM and TCM is given to highlight the main differences between each model, and the advantages of the TCM.

5.6.1 Naive Conceptual Model (NCM)

We will be using class diagrams in the Unified Modeling Language (UML) to construct the conceptual data model of the medical records system. If the data model of the medical records system is constructed solely on explicit information, we argue that a software designer will arrive at the UML class diagram very similar to the one shown in Figure 5.3.

This is of course one possible solution in the UML class structure amongst several alternatives. However, what all the solutions will have in common will be a data schema based on each of the explicit attributes that each record is composed of.

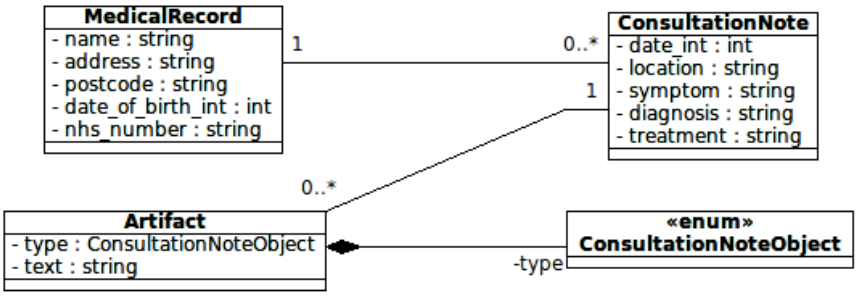


Figure 5.3: Initial Model for the Medical Records System

The schema begins with the MedicalRecord class, which describes the main structure of attributes of a patients record in the VAMP system. The MedicalRecord class consists of the patients personal details and medical record number. The attributes of the class have been modeled based on the information that was recorded on the A5 envelope that makes up each patients medical record.

The schema shows a relationship between MedicalRecord and ConsultationNote with a multiplicity of **0..∗**. This denotes that an instance of MedicalRecord can 0 or more instances of ConsultationNote associated with it. The ConsultationNote is the type structure that represents the data collected during a medical consultation visit. The attributes of the ConsultationNote correspond with the attributes that were implied in the case study to be recorded during medical consultations.

In addition, as there was the possibility for the medical practitioner to add additional types of information to each consultation note, the option to add such information is facilitated through the 0..* relationship between ConsultationNote and Artifact. The structure of the Artifact class is such that it can represent any kind of data that can be added to a consultation note. Each ConsultationNote instance may be associated with a number of Artifact instances which may be one of the enumerated types of ConsultationNoteObject. This is intended to be used to model any kind of additional data that is not encoded as an attribute of a consultation note, but is part of a consultation note by composition or association. Examples include: medical referral notes, discharge letters, doctors notes and so on.

5.6.2 Tacit Conceptual Model (TCM)

Introduction

In this section, we present a tacit conceptual model of the VAMP system, in view of the ethnographic findings presented in the case study by Heath and Luff. We begin with a summary of what is known about both the implicit and explicit relationships in the model.

Model

The TCM will be modelled based on both the explicit and implicit data that was presented in The Medical Records Case Study (Pt. 1)[5]:

i) A set of attributes:

[5]Section 4.5, page 85

a) The required attributes recorded in each consultation note include: date and location (hereafter, we classify these as *process* attributes).

b) The key medical concepts recorded during a medical consultation 'optionally' include: symptom, diagnosis, prognosis, treatment (hereafter, we classify these as *model* attributes).

ii) A set of relationships: From the analysis of the findings of the case study, we know that a number of relationships may exist between the consultation note records in a patients medical record – persistence of illness, continuation of treatment and so on. In addition, we know that the 'presence' or 'arrangement' of attributes can infer meaning through attribute relationships in the data.

iii) Openness: As was highlighted in the case study, the production of a consultation note does not involve applying a set of rules that ascertain what must be included in it.

Given the brief overview of the relationships and associations of the model elements in the system, we propose an improved requirements-level model for the medical records system, as shown in Figure 5.4 [6].

[6]Figure 5.4: Tacit Conceptual Model (TCM): Medical Records System (Page 114)

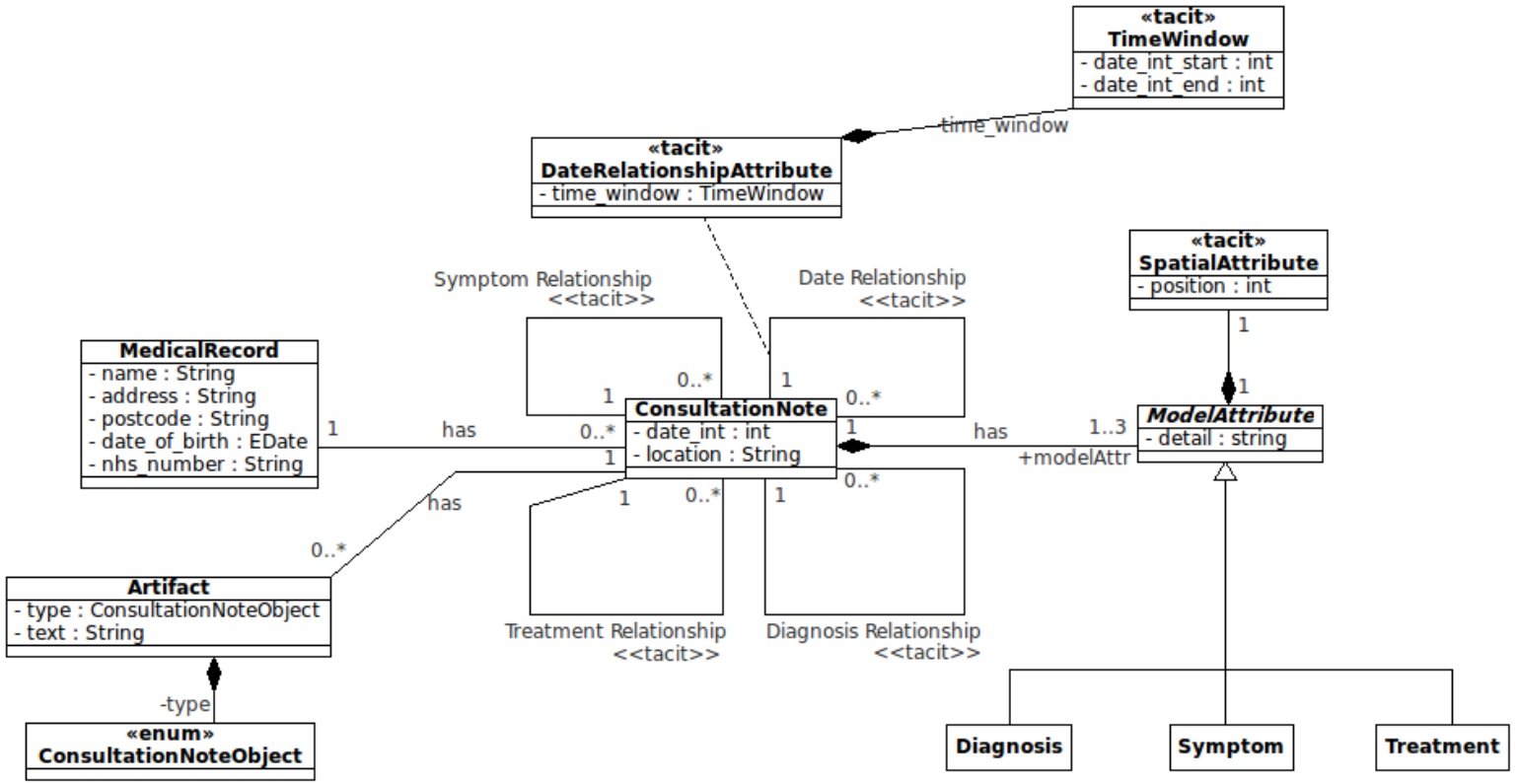


Figure 5.4: Tacit Conceptual Model (TCM): Medical Records System

The model begins with the MedicalRecord class which encapsulates the structural attributes of its paper counterpart. These attributes were modelled based on the data recorded on the A5 envelope. MedicalRecord has a *has* relationship with ConsultationNote. This reflects the requirement of 0 or more ConsultationNote instances belonging to a MedicalRecord instance, as shown by the 0..* multiplicity on the relationship end of ConsultationNote

ConsultationNote is composed of two attributes. It has a *has* relationship with ModelAttribute. ModelAttribute is a generalisation of attributes of ConsultationNote that are medical oriented. That is - Diagnosis, Symptom and Treatment. Each of these attributes form part of a ConsultationNote instance. The filled diamond on the relationship end of consultation demonstrates this aggregation – a ModelAttribute instance *is part of* a ConsultationNote instance. On the relationship end of ModelAttribute, the relationship is constrained by a 1..3 multiplicity, denoting a minimum of one ModelAttribute instance in the relationship, and a maximum of three.

In order to denote the characteristic of position pertaining to instances of ModelAttribute, a composition relationship exists between ModelAttribute and SpatialAttribute. SpatialAttribute is a tacit entity in the model, which has a position attribute to record the position of the ModelAttribute instance.

ConsultationNote comprises a number of reflexive associations. These associations are tacit relationships between ConsultationNote instances that support the implicit relationships between themselves. The relationships include: DateRelationship, DiagnosisRelationship, TreatmentRelationship, SymptomRelationship. An association class on the DateRelationship association allows us to model the behaviour and extra constraints of the association that a standard association cannot describe. The DataRelationshipAttribute is used to add a TimeWindow instance via a composition aggregation, in order to constrain the set of ConsultationNote instances that hold this relationship.

Finally, as consultation notes may contain auxiliary information such as medical referral notes, discharge letters, doctors notes and so on, a relationship exists between ConsultationNote and Artifact. An instance of ConsultationNote may contain **0..*** Artifact instances. Each Artifact may be one of the enumerated types of ConsultationNoteObject

As the model shows, there are entities and relationships annotated with the <<tacit>> stereotype to distinguish them from explicit elements in the CDM. Each of the tacit elements in the model are instances of elements defined in the CDM Metamodel[7]. For example DateRelationship, SymptomRelationship, TreatmentRelationship and DiagnosisRelationship are all specialisations of TacitRelationship. In the next section, we extend the semantics of the model by describing the rules that formalise the tacit information in the model.

Rules

We look at the date relationship (hereafter, DateRelUseCase) from the medical records case study as an example of expressing data rules using this formalism. Therefore, given DateRelUseCase, let:

- C represent the ConsultationNote class.
- DateRelationship(X,Y) will denote the predicate "there is a date relationship between X and Y", where X:C and Y:C.

Therefore:

$$\frac{c_1 : C \quad c_2 : C \quad \alpha : \text{DateRelationship}(c_1, c_2)}{\text{DateRelUseCase}(c_1, c_2, \alpha) : \beta} \tag{5.18}$$

The expression combines C as an explicit type and a date relationship to express the property.

[7]Metamodel for defining Comprehensive Conceptual Data Models: Figure 5.1, Page 97

Medical knowledge $\boldsymbol{\beta}$ is a set of medical truths over set of tuples of type symptom, diagnosis, treatment. Let MedicalRel(A,B) denote that there is a medical relationship between A and B, where A and B can each be a symptom, diagnosis or treatment.

$$\begin{aligned}\boldsymbol{\beta} \equiv\ & \mathsf{MedicalRel}(\mathsf{c_1.diagnosis},\mathsf{c_2.diagnosis}) \bigvee \\ & \mathsf{MedicalRel}(\mathsf{c_1.diagnosis},\mathsf{c_2.treatment}) \bigvee \\ & \mathsf{MedicalRel}(\mathsf{c_1.diagnosis},\mathsf{c_2.symptom})\end{aligned} \tag{5.19}$$

Diagnosis Relationship - DiagRel:
Let DiagRel(A,B) denote that there is a diagnosis relationship between A and B, where A and B are instances of ConsultationNote.

$$\frac{\mathsf{d_1} : \mathsf{DiagRel}(\mathsf{c_1},\mathsf{c_2}) \quad \mathsf{d_2} : \mathsf{DiagRel}(\mathsf{c_2},\mathsf{c_3})}{\mathsf{DiagnosisInference}(\mathsf{d_1},\mathsf{d_2}) : \mathsf{DiagRel}(\mathsf{c_1},\mathsf{c_3})} \tag{5.20}$$

We can say this because of the transitive property. If $\mathsf{DiagRel}(\mathsf{c_1},\mathsf{c_2})$ and $\mathsf{DiagRel}(\mathsf{c_2},\mathsf{c_3})$ then by the transitive property over the 3 elements, we can say $\mathsf{DiagRel}(\mathsf{c_1},\mathsf{c_3})$. Therefore:

$$\mathsf{DiagRel}(\mathsf{c_1},\mathsf{c_2}) \wedge \mathsf{DiagRel}(\mathsf{c_2},\mathsf{c_3}) \Rightarrow \mathsf{DiagRel}(\mathsf{c_1},\mathsf{c_3}) \tag{5.21}$$

Symptom Relationship - SymRel:
Let SymRel(A,B) denote that there is a symptom relationship between A and B, where A and B are instances of ConsultationNote.

$$\frac{\mathsf{s_1} : \mathsf{SymRel}(\mathsf{c_1},\mathsf{c_2}) \quad \mathsf{s_2} : \mathsf{SymRel}(\mathsf{c_2},\mathsf{c_3})}{\mathsf{SymptomInference}(\mathsf{s_1},\mathsf{s_2}) : \mathsf{SymRel}(\mathsf{c_1},\mathsf{c_3})} \tag{5.22}$$

$$\mathsf{SymRel}(c_1,c_2) \wedge \mathsf{SymRel}(c_2,c_3) \Rightarrow \mathsf{SymRel}(c_1,c_3) \tag{5.23}$$

Treatment Relationship - TreatRel:
Let TreatRel(A,B) denote that there is a treatment relationship between A and B, where A and B are instances of ConsultationNote.

$$\frac{t_1 : \mathsf{TreatRel}(c_1,c_2) \quad t_2 : \mathsf{TreatRel}(c_2,c_3)}{\mathsf{TreatmentInference}(t_1,t_2) : \mathsf{TreatRel}(c_1,c_3)} \tag{5.24}$$

$$\mathsf{TreatRel}(c_1,c_2) \wedge \mathsf{TreatRel}(c_2,c_3) \Rightarrow \mathsf{TreatRel}(c_1,c_3) \tag{5.25}$$

As can be seen all tacit relationships become predicates and the semantics for predicate is linked up to a trace over the record. This becomes a use case. The next section makes a comparison the Naive Conceptual Model and the Tacit Conceptual Model.

5.6.3 Model Comparison

In this section, we take a look at how the Naive Conceptual Model[8] compares with the Tacit Conceptual Model[9]. From the onset, a few observations can be made about the two models.

Firstly, aspects of the design that pertain to explicit data and functionality (e.g. ER diagrams, UML Diagrams and so on) are invariant, thus remaining the same. That is, the kind of data schema that will be inferred from any kind of UML analysis of both of our models will more or less be the same.

[8]Naive Conceptual Model: Figure 5.3, page 111
[9]Tacit Conceptual Model: Figure 5.4, page 114

Secondly, with regards to design, the core schema types are invariant. The following invariant classes (or entities) exist in both diagrams:

i) MedicalRecord

ii) ConsultationNote

iii) Artifact

The NCM and TCM are both valid conceptual data models, each showing a different level of detail representing the structure of the requirements schemas. As is shown in the NCM, the data schema of the medical records system can be captured adequately using a classificatory notation such as UML class diagrams. However the UML alone (or comparative notations such E-R diagrams) is not suitable to capture tacit forms of information. This is because, by definition, such diagrams are employed as a means of defining design models, and not for implicit actions, relationships and associations between entities. In practice, such forms of data would be represented by use cases, extra annotations, comments on the model or user stories.

The Tacit Conceptual Model provides a range of requirements feature improvements over the original schema:

a) Tacit Information:
   The TCM highlights various forms of data that are otherwise hidden in the NCM such as the tacit elements and relationships. The TCM might differ slightly in comparison to the naive conceptual model, but there now we have extra relationships to denote the tacit information and attributes of the data in the system. For example, relationships between consultation notes based on symptom similarity via a SymptomRelationship on the ConsultationNote class.

   Association classes on tacit elements in the TCM designate additional forms of data on elements in the model. The TCM shows an example of representing data about a tacit relationship through the use of association classes. DateRelationshipAttribute is an association class in the schema that holds

fields to support the data requirements of a date relationship between two consultation notes. By marking up classes in this way, the analyst will be communicating how an aspect of data in the model should be addressed. In this case, the model shows the need of the eventual design of consultation notes to be addressed in a way that the time window is taken into account.

b) Detailed Representation:
A good conceptual model is a good form of documentation of the data requirements of a system. As CDM's are typically the source models of LDM's, getting the level of detail right at the start of the process can have a positively significant impact on development. In light of this, the first step of the modeling process is to determine what dimensions of data need to be captured. Constructing a TCM based on a metamodel that permits both implicit and explicit data will provide the right features for coverage of loose and fine grained detail.

A metamodel that allows the construction of tacit elements provides the flexibility of giving a more accurate representation of the structure of concepts and the attributes they embody through CDM's that instantiate them. In contrast, constructing a CDM based on explicit data alone may not be clear enough in the design phase. For example, in the naive conceptual model, symptom, treatment and diagnosis were all direct instance fields of ConsultationNote. This structure does not express the relationship between ConsultationNote instances based on these attributes. In the tacit conceptual model, an association was made between ConsultationNote and a new ModelAttribute class with a multiplicity of **1..3** to express the optionality of symptom, treatment and diagnosis through a generalization relationship with these classes.

5.7 Discussion

When tacit information is expressed in a conceptual data model, it takes the physical form of an explicit member element. From a data modelling perspective, this can be seen as tacit information making the transition from

implicit knowledge to explicit knowledge. Tacit Contracts are offered as practical proposals for communicating design obligations to a designer. The literal understanding of tacit information in a data model should be understood from the perspective of providing an additional facet of information to aid in the understanding of explicit information in the domain. While such elements are explicit in the model, they must still maintain their inherent implicit meaning. As such, this meaning is preserved by a stereotype which provides the distinction between elements that should be considered as explicit, and elements that should be viewed as implicit.

The contention of our work has been that highlighting tacit aspects of records in conceptual models plays a key part in improving the RE process by allowing the designer to come up with a design level solution that better fits the domain. Looking at traditional data modeling, associations and classifactory relationships are normally made between explicit entities and data members. The fact that the TCM encapsulates tacit information which may not be implemented explicitly in a subsequent model, for instance the LDM, shows the type of complexity that can be obtained not only through using our metamodel, but also from the fact that its formalisation supports this feature.

Even from a storage perspective, although the schema shape of the TCM changed to accommodate some new features (in comparison to the NCM), at a database level this can be represented in a completely different way. Refinements can still be made in the succeeding Logical Model when the TCM moves to the next phase of the SDLC. The emphasis is on the CDM being a blueprint [25], a real characterisation of the structure of the business domain.

We note 3 important highlights:

Systematic Inquiry

The need to develop a comprehensive CDM incites the obligation to conduct a systematic inquiry of the main structure and outlook of the business domain. This thesis has highlighted Pre-Implementation Ethnography as a means of executing elicitation activities in the requirements gathering phase. Ethno-

graphic reportage can impact the requirements model in a number of ways. Ethnography invokes an exploratory mode of inquiry. It is through an open research method like this that tacit information in a business context can be uncovered.

Tacit information itself can be complicated. While ethnography can be broad, the scope of an ethnographic study can be confined to elucidating the role of entities in a business context to understand usage contexts. As is shown in the case study, a number of new relationships were added to the model after an extended analysis of the usage contexts of the records.

Stakeholder Communication

Communication is an important aspect of software development and maintenance. As such, it is important to convey the right information to the stakeholders of a software project. It is also important that needs of business stakeholders are understood by developers and communicated appropriately between other participants of the SDLC. Conceptual Data Models play a key part in uncovering mismatches of both expectations. Concerning this, we consider Tacit Contracts as an important contribution.

Tacit Contracts are a means of contractually binding the designer to address key aspects of a record's implicit and explicit information profile obtained from Pre-Implementation Ethnography. Tacit Contracts demand that the designer provides a solution based on the profile of the CDM. Notwithstanding this condition, the actual way each requirement is addressed is deferred to the design phase of the SDLC. We contend that the correspondence of tacit requirements in the model may very well entail a radically different design to match the data requirements. We do note however that in the design phases, the CDM will be used in conjunction with other requirements deliverables to outline various aspects of system design (platforms, interfaces, architecture and so on).

Dimensions of Data

It is important to understand the complexity of incorporating different dimensions of data in a model. We argue that the relationships between entities in a model based solely on explicit information such as the NCM, are not as diverse as those that may be found in a TCM. Explicit relationships between data elements (entities, attributes etc) are not as cross cutting as tacit relationships. Looking back at the case study, it showed that relationships may exist between attributes due to properties that pertain to space, time, dimension etc. The nature of tacit based elements (attributes, entities, relationships etc) purports to be one of a complex form which may not have a straight forward translation or mapping to models at the Logical Level (the LDM) and models at the physical level (the PDM). Tacit based elements are more complicated as they embody a lot of information about the records they describe.

5.8 Summary

This chapter has taken the position that tacit information should be made explicit in the conceptual data model of a system, as it provides a clearer picture of the business domain model. We introduced the notion of Tacit Contracts as design level obligations that communicate important implicit features of a business context to a designer. In order to support Tacit Contracts as a design level obligation, a metamodel was proposed for defining Conceptual Data Models capable of supporting both implicit and explicit information. The metamodel was accompanied by its formalisation in Constructive Type Theory. As a demonstration of a real world application of our modeling language, the medical records case study was revisited showing how an analyst might approach constructing a conceptual data model based on ethnographic reportage elicited in the Pre-Implementation phase. In the next chapter, we demonstrate our approach on a larger case study problem on the development and trial of an auction house sales sheet system.

Part III

Methodology Application and Conclusions

6

A Case Study in Auction House Systems Design

This chapter presents a case study on the development and trial of a sales sheet system that was conducted at a U.K auction house. The study is aimed at demonstrating the movement of analysis models from requirements gathered through ethnographic field observations, into design and implementation. The case study utilises Pre-Implementation Ethnography techniques as a mode of inquiry and elicitation. Furthermore, Post-Implementation Ethnography is used as a method of evaluation and review of the final system.

The chapter begins with an overview of the case study, including the motivation for selecting the auction domain. This is followed by an outline of the problem setting which sets out the aims and objectives of the case study. The chapter then gives some background on the auction domain, in order to provide some context to the study. This is followed by an overview of the approach and development process that covers each of the activities and methods within Pre-Implementation, Implementation and Post-Implementation. Finally, the evaluation and discussion present the findings and conclusions of the case study.

6.1 Introduction

This chapter presents a case study that applies the main concepts of ethnographic requirements analysis and model driven engineering in software development. It was beneficial to conduct a case study, as it allowed further investigation into a real world situation that adopted the model driven approach to requirements analysis that has been presented in this thesis. It was also beneficial in being able to better understand the the merits and limitations of this work.

The case study centres around the software development approach described in this thesis, which utilises methods drawn from ethnography in the early and final phases of software development for elicitation and evaluation respectively. The approach is 'model-driven', as it relies on the use of models to express system elements of the problem domain in requirements, design and implementation models.

The application domain of the case study is *auctions*. Specifically, we investigate the usage problems of a sales sheet system used in the sales room of an auction house. Selecting the auction domain as the problem setting offered some key advantages. It allowed us to illustrate several notions:

i) Modeling Relatively Complex Data Scenarios:
The auction house salesroom is an interestingly complex setting. It presented a number of data modeling challenges on account of the tacit information derived from social and behavioural factors.

ii) Implications of Multi-party Interaction on Design:
Unlike the medical records study presented in previous chapters, the sales sheet problem moves from a two to three party interaction scenario to multi party interaction, involving several participants. This ultimately created the need for further consideration of the data management needs of the system.

iii) Conceptual Modeling & Design Level Guidance:
We show how tacit requirements obtained from observational analysis of

the problem setting can be used as a basis for conceptual design at the requirements level, and interface design guidance at the design level.

iv) Technical Solution Guidance:
The proposed system pointed to a number of possible technical solutions that each satisfied the system requirements. We show how tacit information in the constructed conceptual data model is able to inform the choice of design and implementation directions in a system.

In addition, there were a number of practical reasons for selecting the auction domain. There was a significant amount of interest from a nearby auction house in trialling a new sales sheet system. This had the added advantage that the auction house permitted open access for research, which was beneficial to any form of evaluative trials or studies that needed to be planned. The next section, describes the purpose and scope of the case study.

6.2 Scope and Purpose of Study

The subject of our analysis work was to investigate the use a Sales Sheet System at Peter Wilson Fine Art Auctioneer, and a trial of a possible replacement system. Peter Wilson Fine Art Auctioneer is an established Auction House in the U.K that specialises in auctions of fine art and antiques.

The particular focus of the study was to understand the use of the sales sheet system between the auction sales room and the admin office, and conduct a trial with a newly developed system that replaces the old system. This piece of work involved:

i) studying the existing system

ii) eliciting requirements for a new system

iii) developing a new system

iv) evaluating the new system.

It is important to have some background of the auction domain to aid in understanding of the context in which we present our analysis work. As such, in the next section, we present a broad overview of the auction domain.

6.3 The Auction Domain

An auction is a market mechanism which facilitates the buying and selling of goods (lots) governed by a set of rules and constraints between market participants, both buyers (bidders) and sellers (vendors), and any specific rules implemented by the auction house. The rules of an auction determine how lots are traded, and how lots are allocated, including price variations. According to McAfee and McMillian (1997), an auction is:

> [...] a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants.

Given the open nature of auctions, in that rules can be bespoke to any particular type of auction system, there are several variations to auction systems. In this case study, we restrict our discussion exclusively English Auctions, otherwise known as 'open ascending price auctions'.

The English Auction

The English auction is a well understood auction system and is one of the most commonly known auction formats. It is an auction system where bidders compete against each other for the purchase of lots by proposing a purchase price (also called *bid price*), with each bid price between bidders increasing from the last maximum bid price that was proposed by another bidder. The action of a bidder proposing a bid price is called a *bid*. A bidder is said to *outbid* another bidder, when he/she proposes a bid price that is higher than the last proposed bid. During the sale of each lot, the maximum bid price is recorded by an official host of the auction house, called the auctioneer. A winning bidder (also referred to as *maximum bidder*) is declared by the auctioneer when

a bidder proposes a bid price that no other bidder is willing to outbid. This follows with the obligation that the maximum bidder purchases the lot, at the winning bid price.

Auctions involve a number of participants: (i) Auctioneers (ii) Market Participants (iii) Vendors.

i) Auctioneers: Auctioneers act as brokers or intermediary between bidders and vendors. The Auctioneers role is to conduct an auction and ensure that it is governed according to house rules of the auction system in use eg. English Auction. Part of this responsibility is to accept bids for lots being sold on behalf of vendors, and to declare lots sold when there is a winning bid price on a lot.

ii) Market Participants: The visitors of the auction, make up the 'market participants'. We find it necessary to categorise market participants into bidders (or prospective buyers) and spectators. Bidders are market participants that are interested in buying goods at the auction. On the other hand, spectators have no intention of buying goods and thus do not participate in the bidding process. In this chapter, we restrict our view of market participants to bidders, and we refer to them as such.

iii) Vendors: Vendors refer to the actual owners of the lots being sold. The vendor assigns a collection of goods to the auctioneer or the auction house to sell on his behalf. This process may accost a commission to be paid by the vendor to the auction house, as a fee for finding potential bidders who may purchase the lot.

With the preceding overview serving as a background, we describe the organisation of auctions and bidding process in more detail in the next two sections.

6.3.1 Auctions

Auctions begin with the auctioneer reviewing a list of lots that have been selected to be sold in the auction. The auctioneer does this to gain some

familiarity with the entire catalogue that is for sale, and to take note of any high profile lots that may attract a significant number of bids.

The organisation of the auction sale process is such that lots are sold in lot-number order. This ordering is contingent upon the conventional sales process being done without any problems e.g. (i) the withdrawal of a lot by a vendor (ii) late entry of a very sought after lot (iii) an invalid sale occurring due to bidding by an unregistered participant.

At the start of the auction, the auctioneer introduces each lot by announcing the details of lots that are up for sale according to the sales catalogue for the current auction. The details announced by the auctioneer may include the items catalogue number, a description of the item and a starting price which bidders must bid against. With respect to the starting price, special rules of the auction system may permit the vendor of the lot to set a minimum price at which the lot can be sold for and nothing less. This minimum sale price is called the *reserve* price. As such, a lot with a reserve price can only be sold at the auction if winning bid price for the lot is at least equal to the reserve price set by the vendor. The auctioneer must take this into account when conducting bids between market participants, and so the starting bid price of the lot may be adjusted based on the reserve. If the maximum bid does not meet this price, the good will remain unsold in that auction.

Following the auctioneers introduction of a lot, the auctioneer invites competitive bids from market participants. Participation in the bidding process requires bidders to be registered to do so. During bidder registration, each prospective bidder is assigned a unique buyer ID, which they must submit to the auctioneer at the close of every sale that they are appointed 'winning bidder'. If the lot has a reserve price, the auctioneer must first verify that the highest bid price at least meets the reserve before announcing the lot as sold. If the reserve price is not met, the auctioneer may record the maximum price that the lot attracted, but the lot will remain unsold. Also, in situations where an auction item receives no bids from market participants the good will also remain unsold at the auction and may be auctioned again at a later date. The process continues until the auctioneer has introduced the last lot for sale for the session.

6.3.2 Variations to the Bidding Process

While the most predominant activity of an auction is bidding, there are several variations to the form which may be implemented differently across auction houses: (i) restrictions on how bids are offered by bidders (ii) restrictions on what types of participants may take part in the auction (ii) limits on the maximum and minimum bids for lots (iv) affordances that concern the way bids can be made e.g. 'remote bidding' which permits bidders to place bids away from the auction house via telephone dial-in, the internet, email and and so on. In addition to this, some English auction houses permit 'absentee bidding'. This permits a bidder to place a bid prior to the auction. In this case, the bidder submits their maximum bid price for a lot that will be for sale at the auction. The auctioneer then bids on behalf of the bidder during the auction according to the bidders requirements. Bids of this kind are referred to as *commission bids*.

6.3.3 Summary

This section presented an overview of the auction domain, specifically in relation to English Auctions, to give some context to the domain of the case study conducted at Peter Wilson Fine Art Auctioneers. The next section presents the details of the case study which apply our approach to software development that use ethnographic and model driven engineering techniques to (i) review an existing sales sheet system at an auction house (ii) develop a new sales sheet system (iii) evaluate the newly developed system.

6.4 Approach & Development Process

In this section, we describe the development approach of sales sheet system at Peter Wilson Fine Art Auctioneers. The section is split into four subsections. The first subsection presents an overview of the methods used for data analysis and interpretation in the case study. This is followed by three subsections that capture the activities used in the case study, in each of the SDLC categories

as presented below and illustrated in Figure 6.1:

i) Pre-Implementation

ii) Implementation

iii) Post-Implementation

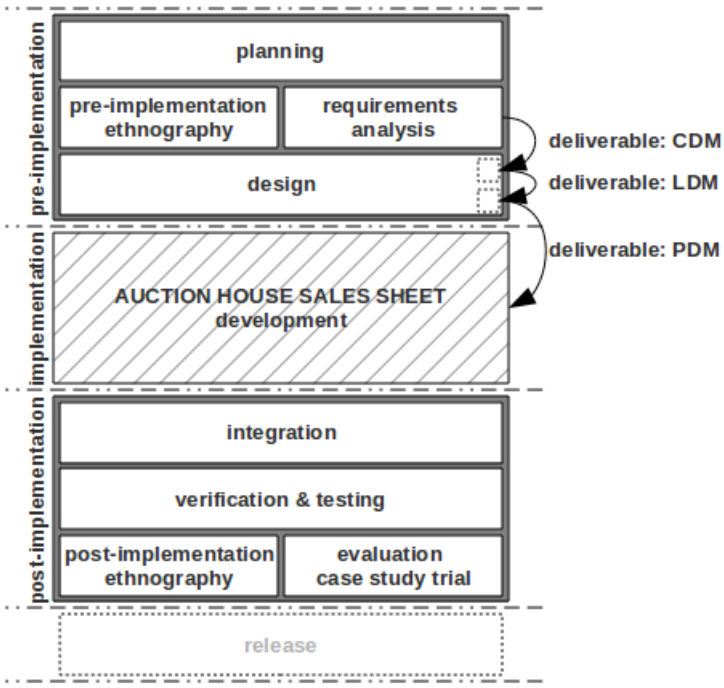


Figure 6.1: SDLC for Auction House Sales Sheet Development

6.4.1 Data Analysis and Interpretation

The approach used for data analysis and interpretation involved using observational techniques and interviews to gain an understanding of the setting to uncover the different roles and responsibilities of the users of the system. It also involved an assessment of the needs of each of the users based on the experience of the system that was already in use. Therefore sample sales sheets were collected in order to build an understanding of the data model of the system and understand how the data was used in the different contexts of the

sales flow. Before we expand our discussion on this, we present a set of key interests and the principal activities that guided the analysis work:

i) Understand the Domain: Our approach was to begin by developing a deep understanding of the work practices of staff at the auction house with respect to their use of the already deployed paper based sales sheet system.

ii) Client Interaction: The purpose of the study was made clear to the staff at the auction house and was designed to be as unobtrusive as possible. Details regarding potential areas of interest within the auction house for data collection were highlighted. In addition to this, all critical information regarding access and the use of the data were answered and agreed upon upfront.

Data collection was a formal part of our overall approach and guided by a set of data collection activities. The activities were divided by the three SDLC categories as presented in Table 6.1. It was important that the physical setting was understood, and was thus necessary to obtain useful information about the existing system from key stakeholders. For this work, data collection was carried out by multiple observers. This provided the benefit of having several perspectives on the subject. It also reduced the time appointed for 'ethnographic inquiry' at the auction house.

For Pre-Implementation, data collection was primarily done through interviews and participant observation. Initial interviews were initially unstructured, which to an extent allowed a broad exploration of how each of the stakeholders believed the auction house was conducted. Furthermore, a number of semi-structured interviews were carried out at a later date in order to investigate potential areas of interest that arose from the initial set of unstructured interviews. Pre-Implementation Field notes included a broad description of the different entities in the auction house, and their activities and interactions. Diagrams and photographs were also used to capture dynamic behaviours in the field.

For Post-Implementation, the main source of data collection was through field observations and video ethnography. Our approach to video data collection was influenced by the extensive guide for video analysis written by Heath

SDLC Category	Principal Activities
Pre-Implementation	• Observation • Unstructured and Structured Interviews • Needs Assessment • Stakeholder Consultation
Implementation	• Iterative Review • Rapid Development
Post-Implementation	• Video Ethnography • Interactive Observation • Structured Interviews • Evaluation

Table 6.1: Data Analysis and Interpretation Activities

et. al [40]. The initial step was to identify the right data subjects to film, and what locations were best to film.

6.4.2 Pre-Implementation

Pre-Implementation ethnography can be instrumental in gaining an understanding of a domain. It brings together a number of research and analysis activities that occur before any kind of formal requirements analysis phase. In this section, we describe the use of Pre-Implementation Ethnography in the early stages of the software development process of the sales sheet system for Peter Wilson Fine Art Auctioneers. We show how both explicit and implicit requirements were informed through (i) interviewing key participants (ii) analysing the current sales sheet system (iii) observing the environment in its normal working condition. Final remarks are given concerning key findings and particular points of interest for the design phase.

Needs Assessment

Needs assessment is the task of working out what is expected (or required) from a newly proposed system. This can be done based on the experience of an already deployed system. This form of exploratory research may be arranged between researchers and key stakeholders, or it may occur away from the field after sufficient data has been collected. In both cases, it is done to ensure that the proposed system captures what is needed by the client.

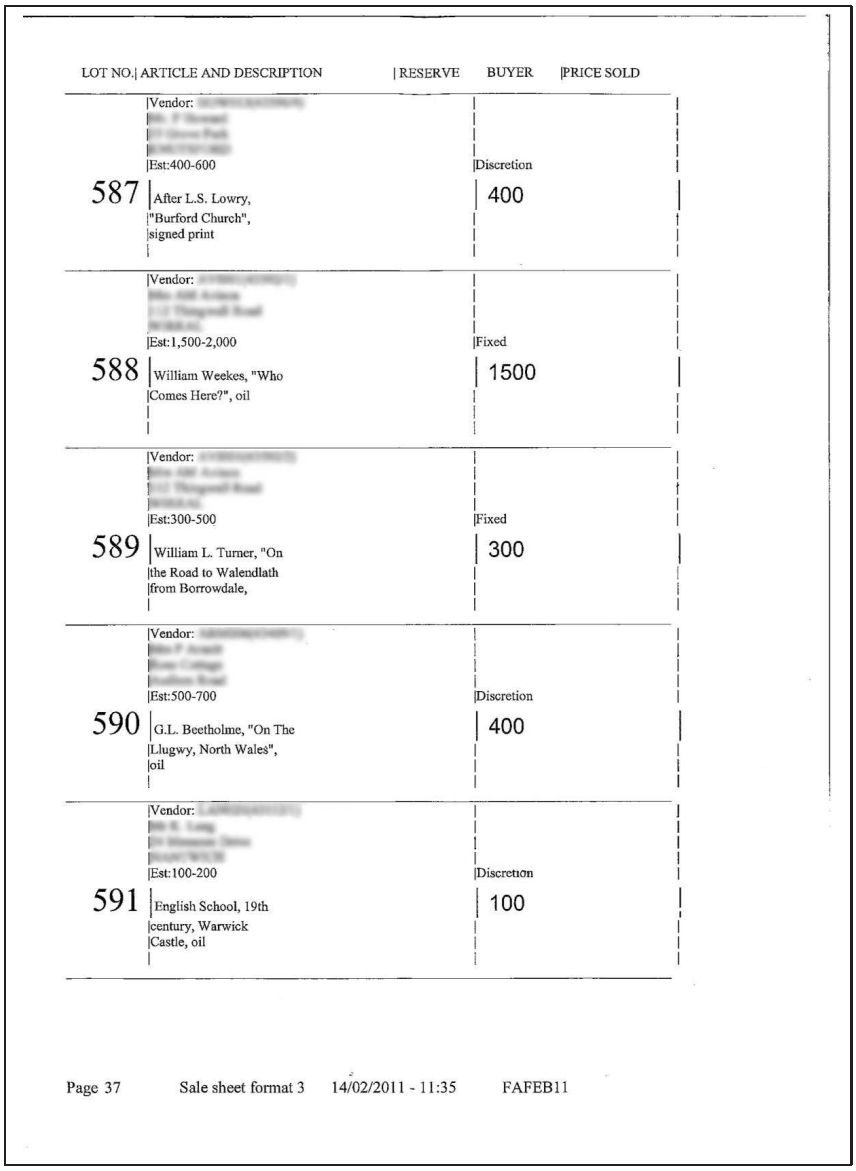
For this purpose, a number of interviews were arranged with the auction staff and a number of domain experts in order to further understand the operations of the auction house. The discussions were aimed at developing a clear understanding of the use of the sales sheet and its different usage contexts and gaining an understanding of how contingencies and exceptions are dealt with. This was necessary for understanding the current sales sheet system user roles, actions and processes of each stakeholder in the sales process of the auction house. Before elaborating on each of these points, we provide an overview of the current sales sheet system.

The Current Sales Sheet System

The sales sheet system used at the auction house took the form of a paper document (Figure 6.2, page 136) which had a number of usage contexts depending on the member of staff using it.

The sales sheet held all the information about lots that were going to be be sold at the auction. This information included a list of each lot, containing details such as: lot number, lot description, price estimate and vendor information. The sales sheet served a number of uses to the auctioneer. It was used (i) as a guide for the auctioneer to use to commence bidding (ii) a reference guide on how to handle bidding (iii) a record of what was sold, and not sold during the auction (iv) a proof of sale. For this reason, it had a number of hand written annotations on it to record extra pieces of information that were necessary for each usage context, which we discuss in subsequent sections.

Figure 6.2 shows an example of a newly printed sales sheet without any handwritten annotations on it. For privacy reasons, the vendor information



LOT NO.	ARTICLE AND DESCRIPTION	RESERVE	BUYER	PRICE SOLD
587	Vendor: [redacted] Est:400-600 After L.S. Lowry, "Burford Church", signed print		Discretion	400
588	Vendor: [redacted] Est:1,500-2,000 William Weekes, "Who Comes Here?", oil		Fixed	1500
589	Vendor: [redacted] Est:300-500 William L. Turner, "On the Road to Walendlath from Borrowdale,		Fixed	300
590	Vendor: [redacted] Est:500-700 G.L. Beetholme, "On The Llugwy, North Wales", oil		Discretion	400
591	Vendor: [redacted] Est:100-200 English School, 19th century, Warwick Castle, oil		Discretion	100

Page 37 Sale sheet format 3 14/02/2011 - 11:35 FAFEB11

Figure 6.2: Current Sales Sheet

has been obscured.

The sales sheet consists of a number of well defined fields:

i) Lot No: represents a unique identifier for a lot being sold in the auction. This identifier is only unique for the current auction.

ii) Article and Description: represents all the information that describes the

lot. Though it is not explicitly labelled on the sales sheet, the data can be broken down into a number of fields:

a) Vendor ID: a unique identifier that is assigned to a registered vendor. It allows the auction house to identify the respective owners of lots.

b) Vendor Name: represents the given name of the registered vendor that is identified by the vendor ID.

c) Vendor Address: the contact address of the vendor that is identified by the vendor ID.

d) Price Estimate: a guide price as to what the lot might sell for. The lower range of the price estimate is the reserve.

e) Description: a visual description of the lot, including any notices regarding the condition of the lot. eg. defects

iii) Reserve: represents the minimum asking price for a lot. It is specified by the vendor of the lots and is used exclusively at the auctioneers discretion. Generally, this information is not shared with market participants. This is an optional field on the sales sheet and can either be 'Fixed' or 'Discretion' as shown above the reserve in the example sales sheet.

iv) Buyer: The buyer field is intended to be used to record the winning bidders ID. This ID is provided to a bidder by the auction house at point of registration prior to the start of the auction.

v) Price Sold: The price field is intended to be used to record the price offered by the maximum bidder. There is a close association between the price and buyer fields. Both fields are required for the sale of an item during the auction and proper processing of the sale post-auction.

Roles, Actions and Processes of the Sales Sheet Users

The sales sheet was used by multiple staff at the auction house for the purposes of recording data and passing on information between staff. In particular staff 'users' included auctioneers, clerks/admin staff and bookkeepers. Usage of the

sales sheet began with the clerk whose task was to create the sales sheet by collating all the data that was relevant to the sale. In particular data on lots being auctioned in the sale, vendor information and relevant reserve prices and commission bid information. Following the creation of the sales sheet, the clerk would proceed to print the sales sheet.

From the point of print, up to the start of the auction, the sales sheet is handled by clerks and admin staff in the back office. Once it is printed, it becomes the *working copy* for that auction. Admin staff are also required to maintain the working copy up to the point of auction. They have the responsibility of keeping the sales sheet up to date and recording any information about the lots that may require any updates before the the start of the auction – for example: the addition or amendment of a reserve price of a lot, or the addition of an extended notice of the condition of a listed lot.

Clerks may also need to record absentee bids (or commission bids) when buyers contact the auction house. Commission Bids are registered via an Absentee Bidding Form. The details collected on the form include the buyers permanent bidding number, the date of the sale and the buyers personal contact details which include: name, address and telephone. The lot number and the maximum bid price are taken. This data is also recorded on the sales sheet against the relevant lots.

Prior to the start of an auction, the auctioneer is required to collect the working copy of the sales sheet for the auction from the admin office. Throughout the auction, the auctioneer is disposed to utilise the sales sheet as his main resource for information on the lots that are being sold in that session. After each sale, the auctioneer must record the price and buyer data into the designated columns on the sales sheet, or a note to indicate that the lot was not sold.

Part of the process of the auction is dealing with post-sale activities which includes recording sales information on to the ledger system, receiving payments from winning bidders and so on. However these activities can only begin once the sales information on the sales sheet reaches the admin office. In effect, this means that the sales sheet needs to be transferred from the salesroom back to the admin office.

In order to reduce delays finalising sales and accepting payments for lots won by bidders, rather than waiting for the entire auction sale to close, the sales sheets are transferred back to the admin office over a number of intervals until the end of the auction. The fact that lots are listed 5-per-page (see Figure 6.2, page 136) allows the auctioneer to release a completed sales sheet for collection by an envoy who delivers it to the book keeper in the admin office. The auctioneer initiates this action by dropping a completed sales sheet behind his seated position on the rostrum. It is only then that an envoy can pick-up the sheet and deliver it to the admin office.

At some point, the book keeper will receive the sales sheet. It is only then that he will begin to enrol the winning bid price and winning bidders id for each lot on to the accounts ledger system. The book keeper also processes any tasks or special instructions from the auctioneer that may have been handwritten as a note on the sales sheet.

Based on the process description given, a number of high level use cases that capture the essential aspects of the working process of the current sales sheet system have been composed. Table 6.2[1] shows an illustration of the use case vocabulary description of the system. Use case names have been assigned to each of the process actions that were described in this section. Also provided, is a summary of the description and the principal actors of each process action. In subsequent sections, we refer to these use cases as HL-Usecases. Figure 6.3[2] is a use case diagram which illustrates the system entities (or actors) of the sales sheet system, as well as the behaviours (use cases) that are associated with the role of each entity. Actors are represented by stick figures with a role name, and use cases are represented by ovals with a use case name. Associations between actors and use cases are indicated by a line drawn between the actor and the use case.

In the next section, we look at the role of the sales sheet in the operation of the auction house. We elaborate on some of its usage contexts and usage contingencies which were elicited from interviews arranged with the auction staff.

[1]Table 6.2: High Level Use Case Description (Page 140)
[2]Figure 6.3: Use case diagram of Auction House High Level Usecases (Page 141)

Action	Use Case Name	Principal Actor	Summary
create	Create Sales Sheet	clerk	The action taken to collate the lots related vendor information of lots that will be auctioned in the sale.
print	Print Sales Sheet	clerk	The action taken to produce the working copy of the sales sheet.
maintain	Update Sales Sheet	clerk	The action taken to keep the information on the sales sheet up to date.
collect	Collect Sales Sheet	auctioneer	Moving the sales sheet from the admin office to the salesroom.
utilise	Use Sales Sheet	auctioneer	Usage actions of the sales sheet during an auction. Eg. reading, scanning.
record	Record Sales Data	auctioneer	The process of recording price and buyer data on the sales sheet after a sale.
release	Release Sales Sheet	auctioneer	Dropping the sales sheet at the collection point.
pick-up	Pick Up Sales Sheet	envoy	Retrieving the sales sheet from the collection point.
deliver	Deliver Sales Sheet	envoy	Dropping the sales sheet in the admin office.
receive	Receive Sales Sheet	bookkeeper	Retrieving the sales sheet from the collection point in the sales office.
enrol	Enrol Sales Data	bookkeeper	Inputting sales data from the sales sheet on to the ledger system.

Table 6.2: High Level Use Case Description

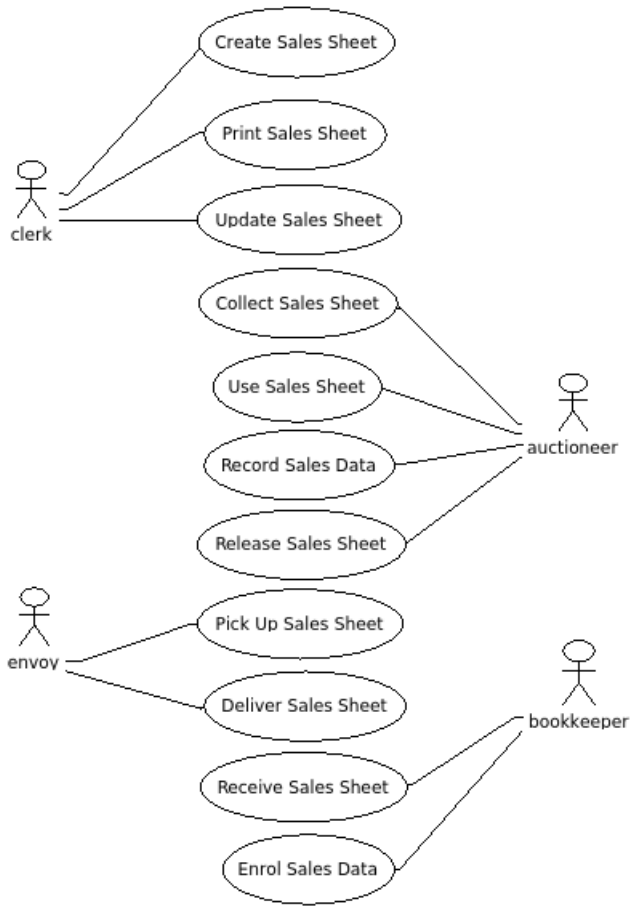


Figure 6.3: Use case diagram of Auction House High Level Usecases

Analysis of the Sales Sheet

The sales sheet plays a crucial role in the working process of an auction sale. Analyses of several examples of both complete and incomplete working copies of sales sheets were instrumental in understanding the behaviour of the different actors in the system. Use of the sales sheet occurred in 3 main usage contexts:

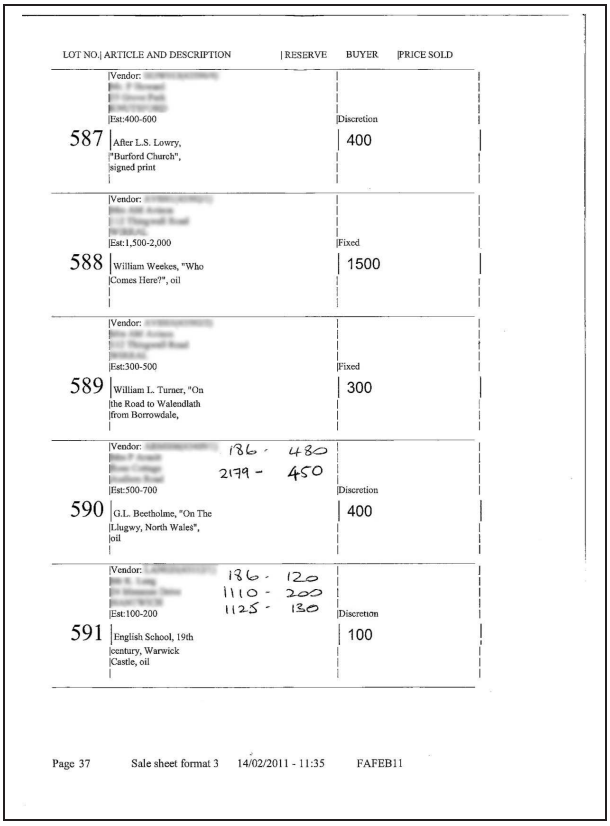
i) Pre-Auction

ii) In-Auction

iii) Post-Auction

Pre-Auction

In contrast to a newly printed sales sheet[3], a Pre-Auction sales sheet has a number of annotations. The sales sheet is handled by the admin clerk and is located in the admin office. Figure 6.4 is an image of a Pre-Auction sales sheet as it might appear right before an auction.



LOT NO.	ARTICLE AND DESCRIPTION	RESERVE	BUYER	PRICE SOLD
	Vendor: [redacted]			
	Est:400-600		Discretion	
587	After L.S. Lowry, "Burford Church", signed print		400	
	Vendor: [redacted]			
	Est:1,500-2,000		Fixed	
588	William Weekes, "Who Comes Here?", oil		1500	
	Vendor: [redacted]			
	Est:300-500		Fixed	
589	William L. Turner, "On the Road to Walendlath from Borrowdale,		300	
	Vendor: [redacted]	186 - 480 2179 - 450		
	Est:500-700		Discretion	
590	G.L. Beetholme, "On The Llugwy, North Wales", oil		400	
	Vendor: [redacted]	186 - 120 1110 - 200 1125 - 130		
	Est:100-200		Discretion	
591	English School, 19th century, Warwick Castle, oil		100	

Page 37 Sale sheet format 3 14/02/2011 - 11:35 FAFEB11

Figure 6.4: Sample Pre-Auction Sales Sheet

The handwritten data refers to commission bids. Commission bids are recorded against the lots that their respective bidders are bidding against. The data is recorded in the article and description column on the sales sheet, on the right side of the vendor information.

[3]Current Sales Sheet - Figure 6.2, Page 136

As is shown in the figure, the commission bids data is not labelled. However, the users of the sales sheet are able to cope with this kind of tacit information as a result of the format being consistent across all auction sheets used in the auction house. Each commission bid entry takes the form of an ordered pair (bidder id, price). The bidder ID of the absentee bidder appears first, followed by a hyphen to separate the maximum bid price which follows after.

Up to the point at which the auction starts, as new commission bids arrive, they are recorded against the lots that their respective bidders have bidded against. Also if any information about the lots requires any alterations (e.g. lot description, reserve etc), the changes will be made directly on the sheet (and duplicated on any back office systems where the data needs to be replicated). Figure 6.5 shows an example of a section of a sales sheet where a note has been written to indicate the withdrawal of a lot from the sale. This is done to ensure that the working copy has the right information at the start of the auction.

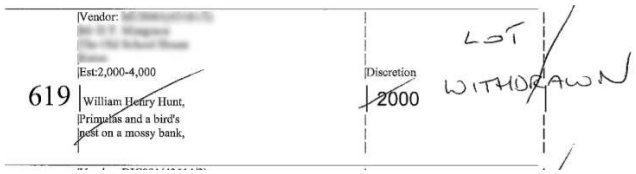


Figure 6.5: Extract from a Pre-Auction Salessheet demonstrating the withdrawal of a lot

Prior to the start of the auction, the auctioneer collects the set of sales sheets for the auction from the admin office and proceeds to take them to the sales room where the auction will take place. This process transitions the Sales Sheet from Pre-Auction to In-Auction.

In-Auction

The sales sheet in the In-Auction environment is handled by the auctioneer. Figure 6.6 shows an image of a sample sales sheet as it might appear during an auction after sales data has been recorded on it by the auctioneer.

The sales sheet shows a number of interesting handwritten fields and annotations. We refer to the way in which sales data is recorded: Lots 587, 588 and

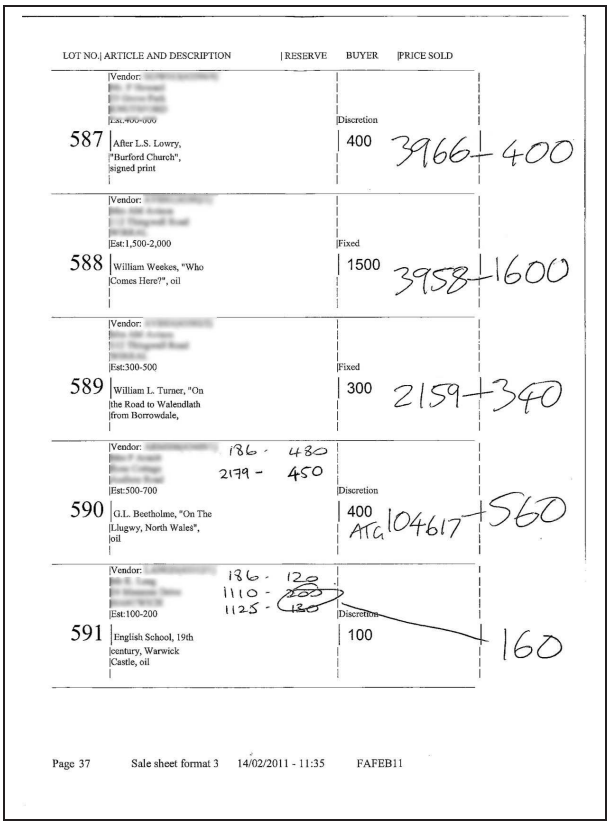


Figure 6.6: Sample In-Auction Sales Sheet

589 are examples of the sale of a lot going to a bidder that is not a commission bidder. As is shown in the figure, the data is recorded as an ordered pair (bidder ID, price). The winning bidders bidder ID is recorded first, followed by a hyphen and then the price the lot was sold at. It is worth noting that this is the same arrangement of fields that is used by the clerk, when recording commission bids.

In the event that the winning bid comes from a commission bid, the winning price is recorded in the price sold column, just as it is done in the standard case (Lots 587, 588, 590). However the buyers id is not recorded in the buyer column. Instead a circle is drawn around the maximum price specified in the commission bid entry of the winning bidder, and a line is drawn between the maximum bid price and the price recorded in the 'price sold' column. Lot 591 in Figure 6.6 demonstrates this example.

In the example shown, a price of 160 is written in the price sold column. A circle is drawn around the maximum price field in the commission bid entry of the winning bidder – in this case it is drawn around the price of 200. A line is drawn between the circled maximum price of 200, and the price of 160 that was recorded in the price sold column. This creates an association between the commission bids entry and the price recorded in the price sold column, indicating that the bidder that issued the commission bid is the winning bidder. We identify this 'association' as tacit. In this particular example, the winning bidder is the bidder identified by bidder ID 1110.

The sales sheet is transitioned from the In-Auction environment to the Post-Auction environment by an envoy who collects the set of completed sales sheets from the sales room at regular intervals, and delivers them to the back office for book keeping.

Post-Auction

In the Post-Auction environment, the sales sheet is handled by the book keeper. Figure 6.7 shows an image of a sales sheet as it might appear following the entry of sales data on to the ledger system.

As is shown in Figure 6.7, the post-auction sales sheet contains a number of additional annotations created by the book keeper. For example, the total price of all the lots on the current sheet is written towards the bottom left region of the sales sheet. At the bottom right region, the running price total of all sheets up to and including the current sheet is recorded. The book keeper uses this information for data validation when inputting data on to the ledger system. As the book keeper types in the price and buyer information for each lot into the ledger system, he ticks off the relevant lot on the sheet. Ticks can be seen for each lot in the example sales sheet shown in Figure 6.7.

During data entry, if the book keeper enters an incorrect price on to the ledger system, there will be a mismatch in the total price recorded on the sales sheet and the total price computed by the ledger system. If a mismatch is found, the book keeper will be able to trace the error more much more easily using the totals recorded on the sales sheet.

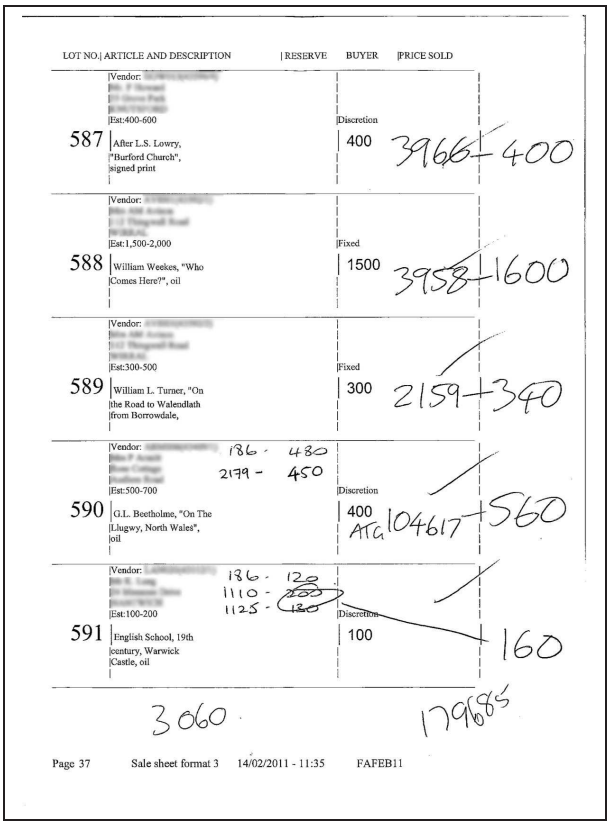


Figure 6.7: Sample Post-Auction Sales Sheet

Preliminary Observations

1. Open Canvas

As a result of the sales sheet being paper based, it is effectively an open canvas which allows its users to annotate freely on it with very little restriction. This may very well have its advantages. For example, notes may be recorded on the sheets in exceptional cases and new 'tacit' data fields may be introduced arbitrarily. The fact that it is open allows the auctioneer for example to swap back and forth between records while writing. Though this may be advantageous in certain circumstances (e.g. a late bid being offered), it has the disadvantage that it is prone to error – the auctioneer is easily able to record the price and buyer information for a sale on the wrong lot.

2. Snowballing Data

Looking at the images side by side (as is shown in Figure 6.8), it shows how how conceptually, the sales sheet transforms between its various usage environments, from pre-auction into in-auction and finally into post-auction.

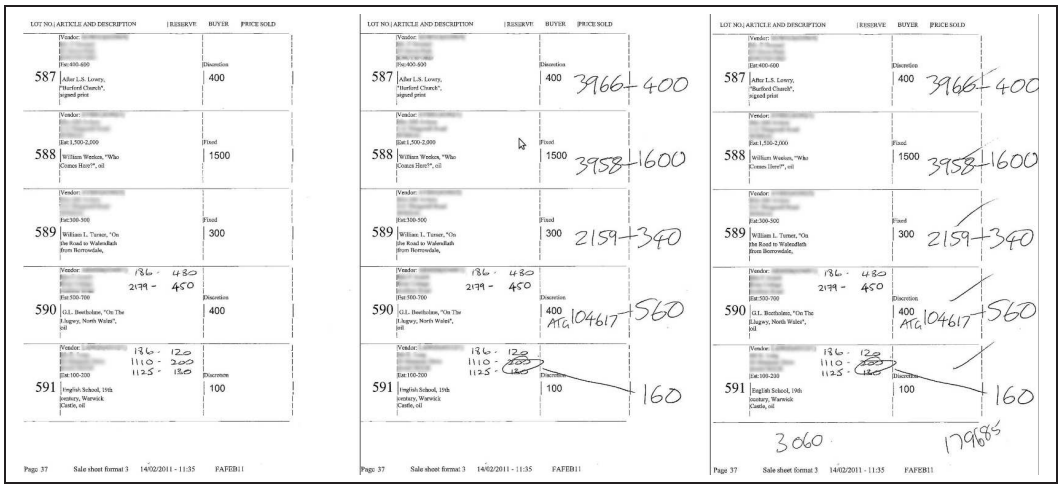


Figure 6.8: Sales Sheets Side-by-Side

The path of the sales data between these environments and how it is snowballed (or transformed) is typical of the movement of data between data boundaries in software systems. For example, moving data between components, data enrichment between processes and so on. For this reason, and due to the social nature and organisation of the setting, care must be taken so that tacit requirements are not lost when drawing up the requirements for the system.

In order to support new fields, rather than redesigning the layout, the open space on the sheet was used to record various items of data regarding particular lots and in some cases, instructions for other users of the sales sheet to action. For example, notes to the auctioneer were seen to be recorded by the admin staff for the auctioneer to use during the auction. On a similar account, notes to the book keeper were seen to be be recorded by the auctioneer. The book keeper also annotated the sales sheet with some useful markers, including 'ticks' to indicate which lots had been processed on the accounts system. To the untrained eye, the sales sheet may be hard to comprehend. However it maintained a useful record of information that was well understood by its

users.

3. A One Handed Sales Sheet System

An important observation of the current salessheet system was the auctioneers' one handed use for form completion. Each auctioneer held the gavel with one hand, and the pen they used to record sales information with the other hand. Occasionally the pen would be dropped on the surface of the rostrum while hand gestures were made to interact with bidders in the room. But what was most common amongst the auctioneers was that the gavel was held in the other hand and hardly placed down. Annotations on the sales sheet were done solely with one hand.

4. Process Bottleneck

A typical auction at Peter Wilson's Auctioneers consists of upwards of 400 lots. This number can increase significantly depending on the type of auction and the expected turn out of auction participants. However in this discussion, we will use 400 lots as the average measure. At 5 lots per sheet (as shown in the example sales sheet in Figure 6.2, page 136), a typical auction will result in 80 sales sheet. The repeated task of transferring sheets from the salesroom to the admin office undoubtedly creates a bottleneck in process. In addition to this, the current practice of handwritten record management creates a bottleneck in the workflow of the paper based sales sheet system. Looking at it from the perspective of bidders, a number of problems are also evident. The fact that the sales sheets are dropped behind the auctioneer in anticipation of collection by the envoy slows down the process of getting sales data on to the ledger system. This results in slower finalising of sales, impacting both on the payment and collection times for winning bidders. Due to this, goods cannot be paid for or redeemed until the relevant sales information has been enrolled on to the ledger system by the book keeper.

Domain Models

In this section, we discuss the design of the system from the perspective of data and the conceptual models of the system. Through the analysis of the

sales process in the auction house and the understanding of what the bidding process entails, we arrived at the Taxonomy of Sales presented below.

Taxonomy of Sales

In order to outline the very specific details of the different possibilities or outcomes of a sale, we present a taxonomy of the categories of sales within the auction house. This taxonomy is presented in Figure 6.9. The auction sales taxonomy begins with the Sale class which is the highest generalisation of possible auction sale categories at the auction house. It specialises to two classes of sales represented by BiddedSale and UnbiddedSale. As the names suggest, an instance of UnbiddedSale relates to auction sales that have not attracted any bids from bidders. On the other hand, an instance of BiddedSale relates to auction sales that have attracted at least one bid.

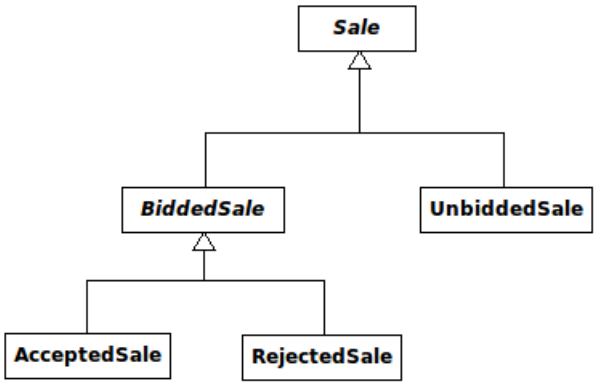


Figure 6.9: Taxonomy for sales in the auction house

However as the description of the problem domain points out, a BiddedSale can both result in the sale of a lot if the reserve is met, and no sale if none of the bids that were submitted met the reserve price of the lot. For this reason, BiddedSale has two specialisation classes: (i) AcceptedSale - this is a legitimate sale according to English Auctions. This is where there is a maximum bidder, and the bid price has at least met the reserve price of the lot. (ii) RejectedSale - this type of sale occurs when a particular lot has attracted bids from market participants but the highest bid offered for the lot has not met the reserve price of the lot. Given this taxonomy, we note that the only instantiable sales

classes of the system are AcceptedSale, RejectedSale and UnbiddedSale. As such, the system will need to be able to process each of these types of sales. The next section introduces the data model of the sales sheet which is formulated around this taxonomy.

The Sales Sheet Conceptual Model

As with any conceptual data model, the Auction House Salesheet CDM provides a visual case in point for reference of the domain. The model describes the domain "entities" and "attributes", both explicit and tacit, that have been informed through Pre-Implementation Ethnography. The CDM of the sales sheet system is presented via the class diagram shown in Figure 6.10[4]. The CDM is formed of abstract and concrete syntax of the domain, and static semantics. It shows the domain concepts, and relationship between domain concepts via association relationships. It also includes a number of tacit attributes and relationships.

[4]Figure 6.10: Sales Sheet Conceptual Data Model (Page 151)

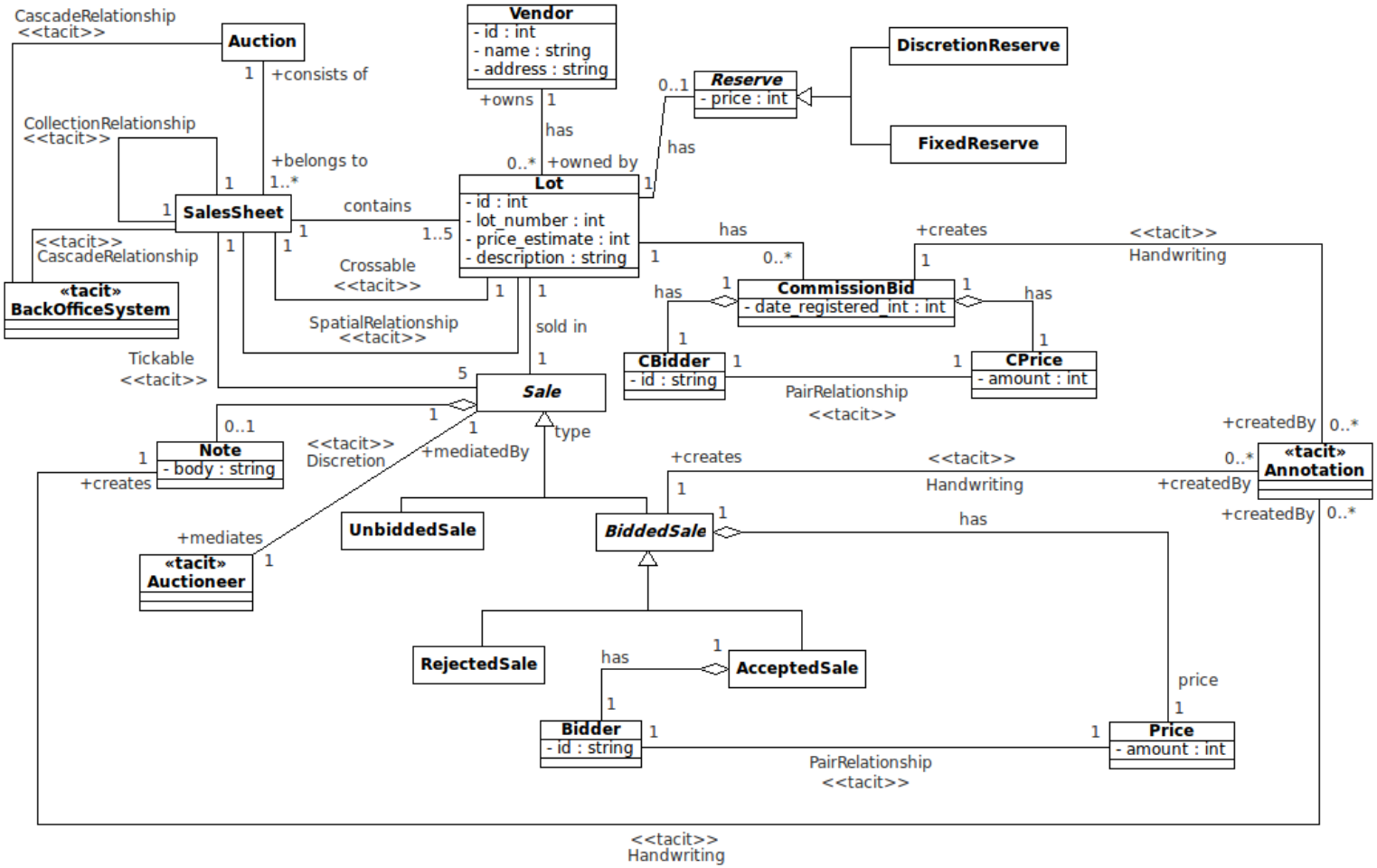


Figure 6.10: Sales Sheet Conceptual Data Model

The CDM begins with the Auction class which represents a physical auction. An instance of Auction *consists of* at least one SalesSheet instance. This is denoted by the 1..* multiplicity on the relationship end of SalesSheet. As Auction instances have multiple SalesSheet instances, we use the CollectionRelationship to define the relationship between all the SalesSheet instances that belong to an auction.

As actions in the back office can affect auction data, in particular sales sheet data, there is CascadeRelationship between Auction and BackOfficeSystem and also between SalesSheet and BackOfficeSystem. CascadeRelationship is a tacit relationship that represents data actions between explicit entities and tacit entities. For instance, deleting a lot on the back office system would require crossing out a lot on the sales sheet, as demonstrated through the analysis of sample sales sheets. BackOfficeSystem is a tacit entity. Though it also has a data model of its own, we are able to represent it as an entity in the model to show its effect on explicit entities in the CDM.

Referring back to the SalesSheet instance, SalesSheet has a 'contains' relationship with Lot. A multiplicity of 1..5 constrains the minimum and maximum number of lots on each sheet within these values. SalesSheet also has a tacit relationship with Lot that is of type Crossable. This relationship models the behaviour of physically crossing out a lot, which in principle is the same as 'deleting the lot. Lot contains a number of attributes id, lot_number, price_estimate, description. Both Vendor and Reserve have been extracted into classes. A Lot instance is owned by 1 Vendor. A Vendor owns none or more Lot instances. Vendors can be individuals or companies. Reserve is a generalisation of the two specialised types of reserves: FixedReserve and DiscretionReserve. A Lot instance has none or 1 Reserve instance.

Referring back to the Lot class, it has an association with the CommissionBid class. As a lot may or may not have commission bids booked against it, there is a multiplicity of 0..* to constrain the number of bid instances. Each CommissionBid instance also has respective associations with CBidder and CPrice which each hold the bidder and price attributes of the commission bid respectively. PairRelationship associates both CBidder and CPrice indicating that they are dependent on each other.

Looking back at the Lot class again, it also has an association with the Sale. A Lot instance is *sold in* a Sale instance. The Sale instance also has a Tickable tacit relationship with the SaleSheet. This relationship has a multiplicity of 5 on the relationship end side of Sale. Tickable defines the action of verifying and validating lots recorded on the sales sheet in the Post-Auction environment. The Sale class assumes the same taxonomical family of entities and relationships shown in Figure 6.9, but with some additional semantics defined over new relationships with entities.

As the auctioneer may optionally append a note to a sale, there is a none or 1 multiplicity on the relationship end of the *has* relationship between Sale and Note. Sale also has a Discretion tacit relationship with the Auctioneer class. This defines the role of the Auctioneer in a sale that falls under the discretionary terms, as discussed in the overview of ethnographic findings. Every sale that attracts a bid requires that the highest price raised from the audience is recorded. If the highest bid falls within the terms for a valid sale, therefore the amount bid is at least the value of the reserve, then the bidder id is recorded. These behaviours are modelled via the relationship between the Price class and BiddedSale class, where the Price instance represents the bid price of the sale. PairRelationship is a tacit relationship which models the dependence between both the Bidder and Price classes.

Annotation is a tacit entity that defines actions of physically writing therefore it has a Handwriting relationship with the explicit data entities in the model that were identified during analysis as handwritten entity types. These entities include: CommissionBid, Note and BiddedSale.

In order to provide a complete model for the auction house sales sheet system, we will provide the syntax that describes the rules and semantics of the tacit information in the model. The next section elaborates on these rules.

System Rules

In this section, we specify a number of rules using our modeling language to define the semantics of the range of tacit information specified in the conceptual model of the auction house that was presented in Figure 6.10. We define

the following predicates which are used in the rules specified in this section.

i) Discretion(a,b): denotes the auctioneers authority to confirm or reject a sale, where a:AcceptedSale and b:Auctioneer.

ii) PairRelationship(x,y): denotes if x and y have a pair relationship, where x is a bidder id and y is a bid price.

iii) Dependent(x,y): denotes the mutual dependence between x and y, where x is a bidder id and y is a bid price.

iv) Crossable(p,q): denotes if q is crossable by p, q:Lot and p:SalesSheet.

AuctionSale Use Case:
A sale is accepted if the bid price is at least the reserve price.

$$\frac{\text{a:AcceptedSale} \quad \text{b:Auctioneer}}{\text{auctionSale(a,b)} : \text{a.price.amount} \geq \text{a.Lot.Reserve.price}} \quad (6.1)$$

AuctionSale with Discretion Use Case:
A sale is accepted if the bid price is less than the discretionary reserve, and the auctioneer applies discretion.

$$\frac{\text{a:AcceptedSale} \quad \text{b:Auctioneer} \quad \text{d:Discretion(a,b)}}{\text{auctionSale(a,b,d)} : (\text{a.Price.amount} < \text{a.Lot.DiscretionReserve.price})} \quad (6.2)$$

Unbidded Sale Use Case:
If a lot has not attracted any bids, skip the lot.

$$\frac{\text{u:UnbiddedSale} \quad \text{b:Auctioneer} \quad \text{l:Lot}}{\text{skipLotUseCase(u,b,l)} : \text{UnbiddedSale(l)}} \quad (6.3)$$

Rejected Sale Use Case:
Reject a sale if the bid price is less than the fixed reserve price.

$$\frac{\text{r:RejectedSale} \quad \text{b:Auctioneer}}{\text{rejectSaleUseCase(r,b)} : \text{r.Price.amount} < \text{r.Lot.FixedReserve.price}} \quad (6.4)$$

Valid Bid Use Case:
A bid is valid if it has a bidder id and a price.

$$\frac{\text{q:AcceptedSale} \quad \text{l:Lot}}{\text{validBidUseCase(q,l):PairRelationship(q.Bidder.id,q.Price.amount)}} \quad (6.5)$$

Accepted Sale Pair Relationship Use Case:
An accepted sale has a pair: bidder and price, which are dependent attributes.

$$\frac{\text{a:AcceptedSale} \quad \text{p} : \text{PairRelationship(a.Bidder,a.Price)}}{\text{requiredPair(c,p)} : \text{dependent(a.Bidder.id,a.Price.amount)}} \quad (6.6)$$

Rejected Sale Annotation Use Case:
A rejected sale is verifiable, valid and tickable.

$$\frac{\text{s:SaleSheet} \quad \text{q:RejectedSale}}{\text{verificationUseCase(s,q):verifiable(q)} \wedge \text{valid(q)} \wedge \text{tickable(q)}} \quad (6.7)$$

Unbidded Sale Annotation Use Case:
An unbidded sale is verifiable, valid and tickable.

$$\frac{\text{s:SaleSheet} \quad \text{q:UnbiddedSale}}{\text{verificationUseCase(s,q):verifiable(q)} \wedge \text{valid(q)} \wedge \text{tickable(q)}} \qquad (6.8)$$

Accepted Sale Annotation Use Case:
An accepted sale is verifiable, valid and tickable.

$$\frac{\text{s:SaleSheet} \quad \text{q:AcceptedSale}}{\text{verificationUseCase(s,q):verifiable(q)} \wedge \text{valid(q)} \wedge \text{tickable(q)}} \qquad (6.9)$$

Sales Sheet Collection Relationship Use Case:
There is a collection relationship over sales sheets that belong to the same auction.

$$\frac{\text{a:Auction} \quad \text{s:SalesSheet}}{\text{collectUseCase(a,s):CollectionRelationship(a,s)}} \qquad (6.10)$$

Sale Sheet Crossable Use Case:
A lot is deletable if a crossable relationship exists between itself and a sales sheet.

$$\frac{\text{s:SalesSheet} \quad \text{l:Lot} \quad \text{c:Crossable(s,l)}}{\text{deletable(s,l,c)} : \neg\, \text{contains(l,s)}} \qquad (6.11)$$

Sales Sheet Referential Integrity Use Case:
There is a referential integrity use case if a lot on a sales sheet has a crossable relationship. This implies a cascading behaviour in which actions (modifications, additions, deletes etc) in the back office system affect the data on the sheet.

$$\frac{\text{s:SalesSheet} \quad \text{l:Lot} \quad \text{c:Crossable(s,l)} \quad \text{q:BackOfficeSystem}}{\text{ReferentialIntegrityUseCase(s,l,c)} : \text{cascade(q)}} \tag{6.12}$$

Note Annotation Use Case:
Represents the handwriting use case of a written note.

$$\frac{\text{a:Annotation} \quad \text{n:Note}}{\text{handwritingUseCase(a,n)} : \text{handwriting(a,n)}} \tag{6.13}$$

Unbidded Sale Annotation Use Case:
Represents the handwriting use case of an unbidded sale.

$$\frac{\text{a:Annotation} \quad \text{u:UnbiddedSale}}{\text{handwritingUseCase(a,u)} : \text{handwriting(a,u)}} \tag{6.14}$$

Accepted Sale Annotation Use Case:
Represents the handwriting use case of an accepted sale.

$$\frac{\text{a:Annotation} \quad \text{b:AcceptedSale}}{\text{handwritingUseCase(a,b)} : \text{handwriting(a,b)}} \tag{6.15}$$

Rejected Sale Annotation Use Case:
Represents the handwriting use case of a rejected sale.

$$\frac{\text{a:Annotation} \quad \text{r:RejectedSale}}{\text{handwritingUseCase(a,r)} : \text{handwriting(a,r)}} \quad (6.16)$$

Commission Bid Annotation Use Case:
Represents the handwriting use case of a commission bid.

$$\frac{\text{a:Annotation} \quad \text{c:CommissionBid}}{\text{handwritingUseCase(a,c)} : \text{handwriting(a,c)}} \quad (6.17)$$

Valid Commission Bid Use Case:
Represents a valid commission bid, which has a pair relationship between bidder and price.

$$\frac{\text{c:CommissionBid} \quad \text{l:Lot}}{\text{validCommissionBidUseCase(c,l)} : \text{PairRelationship(c.CBidder,c.CPrice)}} \quad (6.18)$$

Commission Bid Pair Relationship Use Case:
A commission bid has a pair relationship between bidder and price that are dependent attributes.

$$\frac{\text{c:CommissionBid} \quad \text{p} : \text{PairRelationship(c.CBidder,c.CPrice)}}{\text{requiredPair(c,p)} : \text{dependent(c.CBidder.id,c.CPrice.amount)}} \quad (6.19)$$

Rationalising Design Decisions

For a designer to understand the direction of a potential implementation solution, the designer must first fully understand the requirements of the system, and be able to visualise an implementation solution that meets those requirements. As such, selecting implementation choices that match the needs of a client is an important aspect of design. We therefore consider a number of important usage contexts.

The original sales sheet system was made of a single salessheet that was shared by a number of users. As was shown in Figure 6.3[5], the sales sheet had a different set of functional use cases for each of its principal users. From the clerks' point of view, the use case was one purely of record management. The salessheet was used to ensure that the right information for each sale was available on the sales sheet in advance of the auction. From the auctioneers point of view, the sales sheet was a system used for recording undertakings in the auction sales room. The envoys role was one of data transfer, in moving the sales sheets from the sales room to the admin office. Finally the book keepers' role was one of record entry into a ledger system.

A number of component features were produced based on the analysis of tacit information in the CDM, and informed by the system rules defined in the previous section. Important observations suggested the following key features:

i) Data Validation: Ticks on the salessheet informed by the tickable tacit feature in the CDM suggest the need for verification and validation of data input by users of the salessheet. Informed by Key Rules: [6.1], [6.2], [6.3], [6.4], [6.5], [6.6], [6.16], [6.14], [6.15], [6.18], [6.19]

ii) Referential Integrity: The characteristic of synchronising data across between two sources (for example database tables), such that if records are modified or deleted in one source, changes are reflected in the second source. This is implied by the need to delete a lot on the back office system, and then explicitly cross out the lot on the sales sheet. (This

[5]Figure 6.3: Use case diagram of Auction House High Level Usecases (Page 141)

is suggested by the tacit relationship between crossable and SalesSheet. Informed by Key Rules: [6.11], [6.12]

iii) Record Management (Storing): The ability to store new content on the salessheet. For example, writing commission bids. This requirement is informed by the tacit relationship of type Handwriting which exists between CommissionBid and Annotation. Informed by Key Rules: [6.1], [6.2], [6.3], [6.4], [6.5]

iv) Record Management (Updating): Crossed out lot entries informed by the crossable tacit feature in the CDM suggest the need for being able to update salessheet data i.e modifying or deleting stored data. Informed by Key Rules: [6.1], [6.2], [6.3], [6.4], [6.5]

v) Flexibility: The various relationships with Annotation suggest that free and unrestricted forms of data can be created. This means that new attributes can be created 'on the fly'. For example, if the auctioneer has a special instruction for the bookkeeper, it can be writen on the salessheet and read by the bookkeeper when it is picked up. Informed by Key Rules: [6.13], [6.14], [6.15], [6.16], [6.17]

vi) Movable: The movable nature of the paper based system implies transferability of data between locations. This is implied by the CascadeRelationship between SalesSheet and BackOfficeSystem, as data is modelled to flow between both data locations. This would imply the need for a solution that supports the same. For example in a digital solution, a means for data to be transferred electronically between physical mediums. Informed by Key Rules: [6.10], [6.11], [6.12]

Given all these usages, a number of implementation approaches were considered based on the structure of the CDM and the needs of the auction house. We elaborate on these approaches in the next section.

Possible Implementation Approaches

There is the question of what types of benefits an improved CDM can provide over its equivalent naive counterpart model, when determining what types of

implementation choices to choose from. To pursue this question, while fulfilling the requirement of delivering an improved salessheet system, a rationalisation of implementation approaches was carried out systematically to accommodate the inquisition of what conceptual data models can tell us about the solutions domain. In doing this, two solutions were selected amongst a number of possible choices, that conceptually did not deviate substantially from the original system to support the forms of behaviour and interaction that are currently prominent in the domain. By simply looking at the CDM it is easily apparent that handwritten annotations are a prominent feature of the existing system. Approaches that supported handwritten based interactions were therefore considered:

i) Anoto Pen and Paper Technology (Anoto):
The anoto technology uses a unique printed pattern that is printable on paper to enable a compatible pen equipped with a camera to locate its position on the the pattern [6]. Used in conjunction with a computer, it would allow the seamless interaction and transfer of data between a specially printed sheet with digital media on a computer. For example, a typical salessheets printed on anoto pattern could allow the auctioneers handwritten entries to be transferred to a nearby computer. As such, this solution would require a customised application on the computer that is able to interprete the pattern data that is transmitted to it from the pen.

ii) Touchscreen Tablet PC and Stylus (Tablet):
Touchscreen stylus-enabled device that enables intrinsic digital interactions. This would require a customized application written to support the requirements of the salessheet system. This solution would require an electronic salessheet that would enable the auctioneer to write directly on it to store salesroom data eg. winning bid prices and bidder id's etc.

Each solution reveals a gradual progression from the original system towards a completely electronic solution. The spectrum of choices could have of course been much larger: e.g from the original pen and paper system, which does not do much, up to a fully video and voice based recognition system that is able to automatically detect and transcribe sales and bidding data from live bids in

the salesroom. However, for analysis purposes, the Anoto and Tablet options provided reasonably clear and workable solutions that were sufficient for our objective to interrogate the CDM. In addition, practicalities and constraints of our research were also a consideration: (i) With the aim of advancing the system, it was important to assess the impact of making a small technology shift first and possibly a bigger shift in future empirical exercises. (ii) Time constraints required short development sprints which demanded solutions that could be scoped within the time period of this research (iii) Development expertise with both anoto and tablet platforms made the choices feasible for the sample selection.

Selecting Implementation Approaches

When confronted with a wide array of possible implementation choices to satisfy a set of requirements, it can be a challenging decision making process when planning designs for implementation. In this work, we argue that tacit requirements in a requirements level conceptual data model facilitate a better understanding of possible directions for implementation. Accordingly, tacit contracts directly inform of the need of various characteristics and behaviours of the subsequent system. One approach is to select implementation solutions by applying design affordances and tacit knowledge from the antecedent system to the subsequent system - moving from the old to the new.

Our approach looks at the solutions domain and questions how well tacit contracts and formal rules of the domain feed into the understanding of how each implementation choice addresses the requirements of the problem domain. By doing this, we are able to utilise the 'best' judgement of choices as the basis for the selection. Therefore, for this work, we propose a taxonomic feature based matrix for selecting implementation approaches for design based on a predefined set of component features. Component features can be use cases, rules, characteristics, behaviours or actions of the system. For the purpose of this work, we demonstrate this selection approach based on the two implementation choices that were proposed for the salessheet system in the previous section. In practice, there may very well be a more significant number of choices available to make a selection from.

A Feature based Taxonomy of Sales Sheet Approaches

Looking at the CDM and formal rules of the salesheet system, the following taxonomy of collective support for features and use cases of both systems can be projected on to the feature matrix presented in Table 6.3. The original paper based interface is also included for illustration purposes. Considerations were made following the preliminary observations to find solutions that specifically addressed (i) single handed use, (ii) handwritten notations (iii) freeform text. This fact is reflected in Table 6.3 as both Anoto and Tablet solutions fully support these features.

Component Feature	Paper	Anoto	Tablet
Single Hand Use	✓	✓	✓
Handwritten Annotations	✓	✓	✓
Freeform Text	✓	✓	✓
Data Validation	×	◇	✓
Referential Integrity	×	◇	✓
Record Management (Storing)	×	◇	✓
Record Management (Updating)	×	×	✓
Flexibility (Create new attributes)	✓	✓	◇
Movable	✓	✓	◇

Table 6.3: Taxonomy of Salessheet Solution Domain

✓ Full support for component feature
◇ Partial-Full support for component feature
× Limited-No support for component feature

The taxonomy provides a concise representation of how each of the implementation choices satisfy the component feature requirements of the salessheet system. This is done through the analysis and assessment of each of the component features to assess how well each implementation choice satisfies the set of system requirements, a process which is referred to as feature analysis [76].

While feature analysis is useful in being able to analyse implementation choices, the issue of weighting subjectivity must be recognised. For example, in the taxonomy of the sales sheet solutions, it is debatable whether the assessment of each of the feature scores is subjective, as a different evaluator may

very well assess the features of each implementation choice differently. One method of overcoming this issue, is to use multiple evaluators and utilise an average of the individual assessments provided by all the evaluators.

For this work, feature analysis is used to give an indication of how well each component feature is supported by the solution choices: Paper, Anoto and Tablet. Each component feature in this case carried the same level of importance. However, it is acceptable that some features and use cases will be more important than others, so considerations should be made for such instances. In addition, certain compromises may be made which might constrain decision making. For example cost of implementation, expertise, preference and so on. We look at some notable features of the CDM and their mappings to behaviours in the solution space.

Using the taxonomy as a point of reference and observing how each implementation choice captures the set of defined component features, there is much to be said in favour of the tablet solution in comparison to the anoto based approach. As the anoto based solution is also paper based, it carries some of the drawbacks of the previous system albeit with slight improvements over the original paper system. However in comparison to the tablet approach, the anoto based solution does not provide the features to enforce data validation of handwritten to data on the printed sheet.

The anoto based approach did not provide the features to support referential integrity. If the record of a lot was deleted from the main inventory system, it would have to be crossed out from the sheet which does not represent an improvement from the old system. With the tablet PC approach, a record deleted in the main inventory system would have the deletion reflected on the tablet system. In addition storing and updating records via the anoto based approach would require interacting with the sales sheet directly which presents the same difficulties the original system presented. For example, if a new commission bid is submitted while the auction is in session, the sheet would require manual handwritten updates while on a tablet pc based approach, the new bid can be submitted to the tablet electronically and displayed on the screen for the auctioneer. Based on the merits of the tablet approach, it was selected as the implementation approach for the system.

In the next section, we describe the implemented system and reflect on some of the design decisions made during the development process.

6.4.3 Implementation

The functionality of the system was encapsulated over a range of front end applications for the various users in the system. It was intended that the system should be able to present relevant information to the different types of users in the system. Separate applications were implemented for the auctioneer and the book keeper as their individuals needs were different. In this section, we describe the implemented applications used by both the auctioneer and book keeper. The applications are named as follows:

i) The Auctioneer's Sales Sheet Application (E-SalesSheet)

ii) The Book Keeper's Sales Sheet Application

The Auctioneer's Sales Sheet Application

We refer to Figure 6.11 which shows an interface screenshot of the auctioneers tablet application. The screenshot is shown as it might appear in the In-Auction usage context environment, after the auctioneer has recorded the winning bidders id, and the winning price of a lot. In comparison with the original paper based sales sheet[6], there are a number of key similarities and differences.

The application took a minimalist design approach, without sacrificing core functionality or impeding usability. Just as with the original sales sheet, the E-SalesSheet only presents data on the categories of importance: lot number, article and description, reserve, buyer, price sold, commission bids. The interface is composed of a number of visual elements. The description of the visual elements is given with respect to the lot currently in view in the screenshot:

i) an image of the lot.

[6]Figure 6.6: Sample In-Auction Sales Sheet (Page 144)

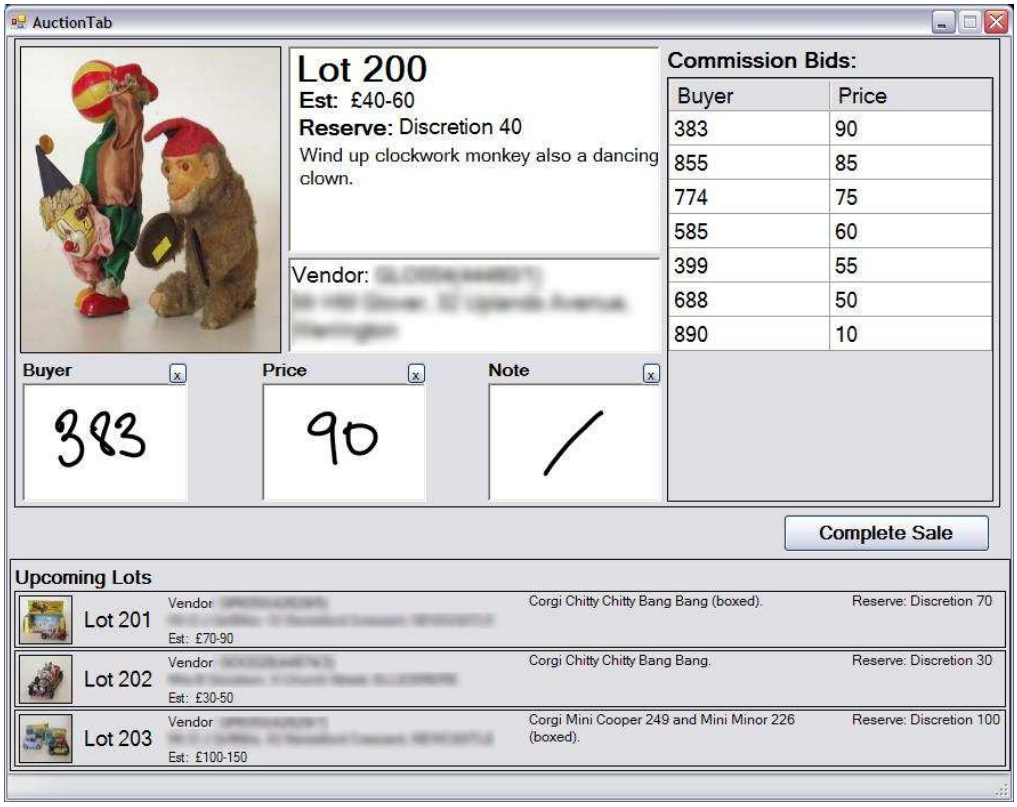


Figure 6.11: The Auctioneer's Sales Sheet

ii) a panel containing the lot details – lot number, price estimate, reserve, description.

iii) a panel containing the vendor details – Vendor ID, name, address.

iv) a panel that presents the commission bids for the lot shown in the main display – this is shown via a table presenting the bidder ID and price for each commission bid entry.

v) a set of ink panels for recording In-Auction sales data – bidder ID, price sold and an optional note. These fields are shown in the application as Buyer, Price and Note respectively.

vi) a complete sale button which must be clicked after a sale is complete, after all sales data has been entered.

vii) a panel displaying a view of the next three 'upcoming' lots in the sale.

The idea is that the auctioneer will treat the data entry process the same way as the original paper based system. During a sale the buyer and price details would be recorded in the relevant ink panels. Instructions may be added in the 'note' ink panel if necessary, and the lot and vendor detail panels will be used in conjunction with the commission bids panel to facilitate the sales process. Finally, at the bottom of the screen the next three upcoming lots are displayed to give the auctioneer upfront notice on lots that might gather elevated interest.

Overview of design choices

In contrast to the paper based sales sheet, the image of the lot was introduced. It was introduced as a need to present the lot data in a single view on the screen. On the existing system, the textual data was available to the auctioneer on the sales sheet, while the image of the lot was available to the auctioneer via a micro-monitor on the rostrum. Consolidating the data in one screen would allow the auctioneer to glance at the lot image, while in view of the lot description adjacent to it rather than having to glance at the sales sheet and look up at the micro monitor.

Another interesting difference between the new system and the existing sales sheet in terms of layout is the presentation of the article and description data. In the E-SalesSheet system, the lot data and the vendor data was separated and placed in different panels, but in close proximity. The vendor data was deemed to be optional but 'needed' information which the auctioneer may choose to refer to. However it was not a critical field that would prevent a lot from being sold. The reserve field was placed in close proximity to the price estimate field of the lot. This was done in anticipation of enabling the auctioneer to make an easy comparison between both fields, as was indicated by Pre-Implementation Ethnography observations.

Furthermore, the new system provided a better presentation of commission bids. The pair-relationship between the bidder id and price was preserved with bidder ID occurring on the left side of its associated price attribute. The placement of the commission bids table was also an informed decision. It

was spatially placed in a position that would enable the auctioneer to make a quick comparison between the best commission price and the reserve. In addition to this, the commission bids table implemented a smart ordering of the commission bid entries. The ordering was done both by price in descending order and time of booking in ascending order. This meant that the auctioneer could immediately have a view of the most competitive commission bid by looking at the entry at the top of the table. In the paper based system, it would have been necessary to scan all the commission bid entries on the sales sheet that corresponded with the lot, to identify the bid with the highest price.

Ink panels were provided to record handwritten data on (i) price (ii) bidder ID and (iii) an optional note for occasions where the auctioneer may want to record additional data against a sale. The sizes of the inkpanels for each field was just big enough to record the amount of data that was typical for each field. It was necessary to add the complete sale button to enable the auctioneer to explicitly regulate the flow of sales information stored by the application. At the point of clicking complete sale there was an electronic submission of the data recorded in the ink panes to a centralised database. Located at the bottom of the application was a panel that showed up to the 'next three' upcoming lots in the auction. During Pre-Implementation Ethnography the auctioneer was observed glancing at the next few lots several times during normal use of the paper based system. This data was purposeful in keeping the auctioneer informed of any lots of significance that may be following up shortly. This is the type of data that typical requirements analysis methods will not capture, and therefore will not be included as part of the design of the proposed system.

The Book Keepers' Sales Sheet Application

The design of the book keepers' application broadly shared most of the same characteristics with the original paper based sales sheet. It was designed as minimalistic as possible to meet the main usage requirements of the book keeper which was primarily for data capture. The function of the salessheet was to present post-auction data to the book keeper so that it could be trans-

ferred on to the ledger system. To reflect these usage needs, the data presentation features on the book keepers sales sheet was designed to be read-only. Figure 6.12 shows an interface screenshot with sales data, as it may appear in the Post-Auction usage context environment. The image shares a close resemblance with the paper based version[7].

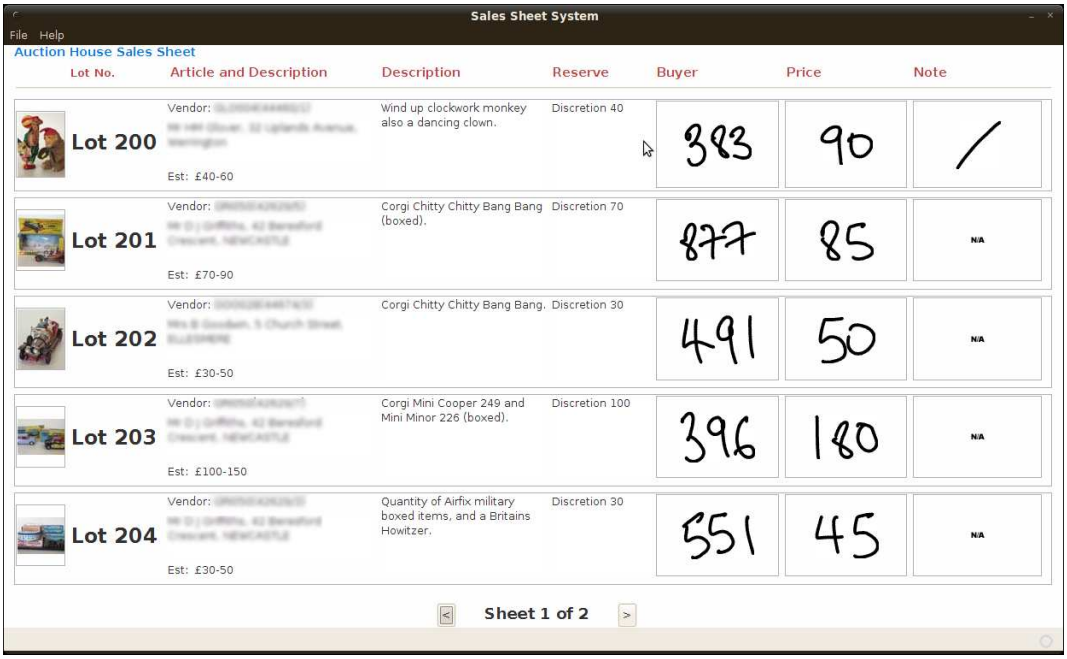


Figure 6.12: Interface Screenshot

As the image demonstrates, the application preserves a number of visual characteristics from the existing sales sheet system. There is much resemblance in the arrangement of fields in comparison to the paper based sales sheet. The application also presented 5 lots on-screen which corresponded with the number of lots that the paper based system presented. A pager with left and right arrows was available at the bottom of the screen to allow the book keeper to browse between sheets to display any of the sheets within the current auction.

[7]Figure 6.7. Sample Post-Auction Sales Sheet (Page 146)

6.4.4 Post-Implementation

Part of the criteria for the evaluation of the system was to assess whether there will be any barriers to its full adoption. Furthermore, it was necessary to explore whether further requirements had remained unseen in the domain and whether any further changes needed to be made to the system. Video recordings were used to capture the trial system in use by both the auctioneer and the book keeper. Using video recordings allowed for the possibility of further analysis of subjects away from the field. The next section elaborates on the data collection and evaluation process.

Subjects for Post-Implementation Ethnography Data Collection

As the primary focus of the PoIE evaluative study was to scrutinize the newly introduced system, the two subjects for analysis were: (i) the auctioneer (ii) the bookkeeper.

i) The Auctioneer: Given that the auctioneers tablet was of central importance to the workflow of the system, it was important to collect relevant and rich enough information to analyse and appraise his use of the new technology. This required two viewpoints for videography. One of the aims of the evaluation was to assess the auctioneer's interaction with the system and the quality of use of the application. For this reason, a close-up camera view of the auctioneer using the tablet was necessary. In this view, the auctioneers interaction with the tablet and hand gestures could be captured. Secondly, it was important to understand if there was any impact on the auctioneers general behaviour when using the tablet system. Thus, a second camera view was focused on the auctioneer. This camera view was taken from the view of the market participants in the sales room.

ii) The Bookkeeper: As the bookkeeper's responsibility was solely to validate and transfer sales information from the new system to the existing accounts system, one viewpoint of the book keeper was used.

Note

Video is one of the richest persistent forms of data collection. Unlike other forms of data collection methods, video provides both visual and audio features which enables the analysis from both dimensions. It also has the advantage of collaborative review away from the field, and the ability to be recalled several times if needed.

Therefore in total, 3 fixed cameras were used for video recording during the auction trial:

i) 1 camera trained to the auctioneers tablet: useful for gaining insights on the usage of the device.

ii) 1 camera trained to the auctioneer from a distance: useful for analysing any change in behaviour of the auctioneer.

iii) 1 camera trained to the bookkeeper in the admin office in full view of the bookkeeper's use of the new salessheet system: important for evaluation purposes.

The decision on the placement of each of the cameras was made in consultation with the participants involved in the study. All three cameras were microphone-enabled for the purpose of capturing audio for analysis. To avoid any complications with obtaining informed consent from market participants, all cameras used in the data collection process, were focused solely on key staff that were part of the trial.

Procedure

The trial was setup to take place during a live auction on a sample selection of lots. The auction consisted of a catalogue of 245 lots for sale in the fine-art category. The catalogue was split into two for the purpose of the trial. The first 200 lots were setup to be sold using the existing paper based salessheet system, and the final 45 lots were setup to be sold using the new sales sheet system. To add redundancy to the trial and limit the potential for data loss due to technology failure, a backup auctioneer was asked to stand in and record

the sales data on the existing paper sales sheet alongside the main auctioneer who used the new sales sheet system. Both the auctioneer and book keeper were given training that familiarised them with the new system.

Evaluation

The evaluation of the system consisted of two perspectives which enabled structured analyses on all key parties during and after the trial. We adopt Bhola et. al's Formative and Summative Evaluation techniques [13] for this process:

i) Formative Evaluation: carries a specific focus on the process of an activity while it is in session.

ii) Summative Evaluation: focuses on the value of an activity after it has concluded.

As part of the Formative evaluation of the system, it was necessary to assess the conditions of the trial environment during the sale of the first 200 in the auction. Therefore the first phase of the evaluation process was to observe the existing system in use directly before the trial. This phase was carried out by multiple observers during the first part of the trial. It was beneficial for the purpose of analysis to see how the existing system worked under the direct conditions of the trial.

The second phase was important for evaluating the use of the new system. This presented the opportunity to assess the extent to which the requirements of the system had been met. It also provided the opportunity to determine if the new system created new tacit behaviours. Video recordings were used in this phase because it allowed for repeated review for analysis work away from the field. Figure 6.13 shows the camera angles that were used to record the auctioneer in practice. The left side of the image shows the camera angle of the auctioneer which was filmed from a distance. The right side of the image shows a close up view of the technology. As both cameras were microphone enabled, analysis could be done on the auctioneers' commentary during the auction.

Figure 6.13: Images from Ethnographic Film – The Auctioneer

In the admin office, a camera was trained on the bookkeeper and the computer screens in use. Figure 6.14 shows an image of the bookkeeper viewing information from the salessheet system. Following the sale of each new lot in the auction room, the sale information which included the buyer and price information would appear on the screen to the right of the bookkeepers' desk. The bookkeeper would then transfer this information to ledger system. This differs from the bookkeeper having to wait for the paper based sales sheet to be delivered from the sales room by other members of staff.

Figure 6.14: Images from Ethnographic Film – Admin Office

The final phase of evaluation was the analysis of the video data collected from the trial. The data collected included video recordings, field notes and

feedback from the staff in the auction house that engaged in the trial. Before we provide a summative evaluation of the study, we first outline a number of limitations of the trial that a fully developed version of the system would not have outside the constraints of this work.

Trial Limitations and Caveats

Executing the trial required a number of practical workarounds and compromises to manage some of the constraints of trial particularly (i) lack of full integration with the back office systems (ii) limited development time (iii) untestable scenarios (eg. the withdrawal of a bid).

Lack of full integration with the existing information systems at the auction house which included the accounts, inventory and ledger systems, and full flexibility to install the necessary hardware in real-world deployable format, created a number of issues which are worth highlighting. As the system was not fully integrated with the existing auction house infrastructure, the auctioneer was required to use both the new sales sheet system, and the micro monitor switch to display the image of the lot on the public display in the sales room. A fully integrated sales sheet system would also control the public display image to keep the image shown in synchronisation with the current lot that is up for sale on the tablet screen. Secondly the book keeper was confined to using two different systems for input. Requiring to view the sales information that came in on the screen of the new system, and typing the sales data into the accounts and ledger system, on another screen. A fully integrated system would send the sales data directly to the accounts and ledger system.

Development time was constrained, therefore not all features that would be expected to be implemented in the final deployed system were completed. For instance, the auctioneers salessheet provided the ability to record sales data and submit the sales information by clicking the 'complete sale', however once the button was clicked there was no way to go back to previously submitted lots. Similarly, there was no way to look ahead at upcoming lots beyond the next 3 upcoming lots. Although these feature were not directly needed during the trial, they were highlighted as key features for the system.

Finally, some conditions were not testable under the trial as they did not arise during the trial. For instance (i) lot withdrawals while the auction was in session (ii) submission of commission bids while the auction was in session. Testing these scenarios would have been important to cover all behavioural possibilities of the system.

Field Observations and Video Ethnography Evaluation

In this section, we summarise the highlights of the observations and analysis of the auctioneer's sales sheet system obtained through field observations and video ethnography.

From the outset of the trial, there were no visible problems or reports from the auctioneer with regards to the technology itself. There were no problems with writing with the stylus on the ink panes, and no specific issues with retrieving or submitting lot data. The trial demonstrated a number of improvements over the existing sales process. Utilising the new system enabled auction sales information to be immediately available in the accounts office after the sale was complete and the data was submitted by the auctioneer. This eliminated the need for an envoy, and enabled buyers to immediately pay for lots in the admin office once the sale of the lot was complete. This was of course in contrast to buyers having to wait for sales sheets to be to be completed by the auctioneer and picked up and transferred to the book keeper to enrol on the ledger system. The new system also had the potential to enable vendors to be notified instantly when lots that they owned had been sold.

Looking at the new system from the perspective of the auctioneer, there were a number of interesting observations. Though the auctioneer started off apprehensively when the auction started with the new system, once the layout and category positions where correctly identified and familiarised, there was a noticeable increase in speed and confidence in the use of the system. In comparison to the paper based system, the auctioneer's sales sheet incorporated a number of additional elements to improve the sales process. For instance the introduction of lot images on the auctioneer's sales sheet eliminated the need for the auctioneer to glance at a separate screen located on the rostrum, which

was the case with the previous system.

From a design perspective, the sales sheet borrowed several visual details from the paper based sales sheet. Much of these 're-used' visual elements were feasible design choices to carry over to the new system. This is similar to the concepts of bridge attributes which are design affordances that exist in both the old and new experience of a system [16]. Cass [16] discusses how highlighting so called "bridge" attributes can be used to facilitate an understanding of how technology can be applied to perform tasks. They are used to orient a user to a new system by allowing them to recognise and interpret a new experience through recognisable affordances or metaphors of the old system. Conceptual metaphors such as bridge attributes (also referred to as *skeumorphs*) widely reduce the barrier of entry of users to new systems because of their familiarity of the old system, and the already recognisable features in the new experience. Field notes acquired during the field observation exercise provided insights into which attributes could be leveraged. For instance handwritten fields, layout positioning and so on. This drove part of the decision making when considering the sample of implementation choices of the system.

While bridge attributes were included in the project, there were some additional feature improvements and design choices that demonstrated benefits through video analysis. The separation of the upcoming lots from the current lot being sold, allowed the auctioneer to easily return focus on the details of the current lot if he glanced away. The fact that the auctioneer's sales sheet had only one lot in view at any one time also prevented the problem of being able to record the sales information against the wrong lot, which was the case with the paper based sales sheet.

In addition to what was expected, there were a number of exceptional conditions which the system was presented with. In one example, the winning bidder for a lot was unregistered and thus did not have a bidder ID to present to the auctioneer after the sale was appointed to him. The auctioneer was able to write the buyers surname in the buyer field and instruct the winning bidder to make his way to the admin office to register and pay for the winning lot. The auctioneer also included the note "buyer on his way" in the note ink pane on the sales sheet prior to clicking on complete sale. This information

was presented to the book keeper, who was able to resolve the matter. This would not have been immediately possible with the paper based sales sheet as the sales sheet would have had to remain behind the rostrum up to the point of collection by the envoy.

6.5 Discussion & Conclusions

This section discusses the findings of the case study and outlines a number of research implications.

6.5.1 Overview of Work

We begin by recalling the objectives of the case study. The objective of the auction house study was not just about developing a replacement system for an auction sales sheet system, it was about investigating our proposed development approach, which concerns early phase and final phase SDLC activities when considering the development of a system. Importantly, in Pre-Implementation it was about: (i) looking at the process of developing a tacit conceptual model of a system, (ii) analysing what the model is able to tell us about the domain (iii) investigating what additional informational benefits it provides in comparison to a naive model (iv) observing how the important structures, entities, relationships and behaviours have been represented and preserved in the model (v) and finally seeing how these concerns contribute to the direction of implementation designs. Concerning the latter phase of the SDLC, Post-Implementation concerns were applicable to (i) looking at how the details captured in the CDM resulted in the final system (ii) analysing facets of information that may have been 'lost' in the conceptual model if a naive model was used (iii) and finally seeing what worked well and what did not.

6.5.2 The Role of Tacit Information in the SDLC

Conceptually the activities of requirements analysis bear much consideration for the concerns of the problem domain and should overlook that of the solutions domain. It is conceivable that a deep understanding of tacit information

in the problem domain has the potential to improve the design process. Accordingly, concerns of the design phase of the SDLC may form part of the requirements analysis effort. This end-to-end approach is one of several perspectives of the concept of bridging the gap. In this context, bridging the gap refers to adjoining the gathering of client requirements and directing the gathering process based on what information might be important at the design level. At the conceptual data modeling level, this presents a need for tacit information alongside explicit information.

From a data modeling perspective, this requires much understanding of: (i) tacit information in the problem domain: this facilitates the creation of rich conceptual data models that reflect the conceptual structure of the domain (ii) how tacit information carries over to the solutions domain: this drives the requirements gathering process to elicit requirements that can tell us more about design directions when considering choices in the solutions domain.

To address the first point, ethnography provides a deep level of understanding of the domain to enable the construction of detailed requirements models. On the second point, there is much to be understood about how upfront insight of the solutions domain impacts the requirements and design process and vice versa. It is however curious to observe whether a conceptual data model can tell us something modest, or something more radical about design. In this work, the case study enabled us to look at each of these areas. Our work has presented a different perspective on how tacit information can be used in the SDLC. We have therefore demonstrated two notions:

i) Modeling Ethnographic Requirements

ii) Mapping Ethnographic Requirements to Design Models

Modeling Ethnographic Requirements

The case study demonstrated a number of advantages of using ethnography for elicitation. The characteristic of ethnography in supporting awareness of context, enabled a deep understanding of the different usage scenarios of the sales sheet. This led to a number of requirements which required representation as conceptual structures.

Current requirements models would suggest the construction of explicit data concerns of a system using for example, structural diagrams such as Class Diagrams, and implicit data concerns of a system using behavioural diagrams such as Use Case diagrams. For instance, the auctioneer's interaction with the sales sheet to record the price and bidder information of a sale would be recorded in a behavioural diagram like a Use Case diagram rather than in a structural diagram like a class diagram.

However there are drawbacks with completely separating behavioural and structural information in requirements models when there are modes of interaction between both types of information. In addition, utilising a naive conceptual model and a use case diagram would also require an additional step of having to reconcile a gap in the understanding of how explicit elements relate to behavioural and tacit concerns of the system. This problem of having to reconcile the usage contexts of data across both behavioural and structural models increases the level of complexity of analysis in design. To this point, both models may elaborate on different levels of information. For instance, if we utilised the auction house Use Case diagram presented in Figure 6.3[8] and a 'naive conceptual model' of the sales sheet system which did not embody tacit information, there would be no detail to show tacit behaviours and relationships, for example CascadeRelationship between BackOfficeSystem and SalesSheet, and Discretion between Auctioneer and Sale, as shown in the tacit conceptual model[9]. This information is of course present if there are use case descriptions to support the Use Case diagram. However this approach is not convenient or suitable for formal analysis or model driven development. If we even assumed that written use cases descriptions and other behavioural diagrams descriptions were sufficient, there is the challenge of ambiguity, due to written texts not being good for clarity and consistency.

If we consider that data has structure and behaviour that *is* tacit, then projecting both concerns in the same model can provide a better understanding of the system simply from direct observation of the model. It would also lead to processes for model driven development, as tacit elements can be treated as

[8]Figure 6.3: Use case diagram of Auction House High Level Usecases (Page 141)
[9]Figure 6.10: Sales Sheet Conceptual Data Model (Page 151)

first class entities of the system. Ultimately, the problem we are addressing in this work is not so much on the separation of structural and behavioural diagrams. It is about representing the right level of detail in a tacit requirements model that shows the interactions between elements of an explicit and implicit kind. Thus we have, a tacit requirements metamodel that enables us to create models that inform us on both structural, behavioural and tacit concerns of a system.

We do not propose that behavioural modeling using use cases for example, goes away. We are proposing that elements that are thought of as purely behavioural correspond with relationships and with data entities, and this information should be encoded as tacit information. Whilst, we are bringing behavioural concerns into the data model. These concerns are part of the forms of data described, and they are still static relationships. They are simply data relationships and attributes. Similar to how Entity Relationship Diagrams compose relationships between entities, our model does the same but with tacit and explicit elements. Notably, tacit elements will not appear in the physical model, and therefore will not be part of the implementation data schema. However, at the conceptual level, they still qualify as things that we will model in the expanded notion of a data model.

In modeling the sales sheet system, our tacit requirements metamodel demonstrated the flexibility required to model these types of concerns. We were able to preserve tacit information. This is unusual for standard class diagrams and Entity Relationship (E.R) Diagrams. Looking at traditional E.R diagrams, they will only show explicit information and not relationships like those highlighted in the auction house conceptual model, like the pair relationship between bidder and price. This is the same result with traditional class diagrams. They both share the drawback of only describing data that is needed – explicit data.

With a tacit requirements model, the model shows some understanding of the usage functions and relationships of data entities. Importantly, the outcome of this is a deliverable to the design phase that conveys this level of information. This is peculiar of our approach which utilises ethnography to 'gather the details' and the combination of an improved CDM and a modeling

language to 'record the details'. In moving from Conceptual Data Models to Logical Data Models, we find that complex data requirements need to be modelled outside the standard mapping to data schemas. We therefore need an additional type of design model that enables us to specify and implement choices.

Mapping Ethnographic Requirements to Design Models

In this work, we have argued that tacit information has an impact on the implementation choices made at the design level. The auction house study allowed us to look at the modeling process of creating a conceptual data model of a system, and how it subsequently turns into a physical system. We make two important observations about the role of the conceptual data model:

i) Diverging design level models

ii) Mapping Conceptual Data Models to Implementation Choices

Looking at the conceptual model of the sales sheet system, it not only informs the construction of the logical data model, it provides details necessary to identify possible implementation choices which may include architectures, devices, and so on. This characteristic is created by the ability of the model to communicate use cases and behaviours. For example, a model which predominantly carries 'annotation' attributes and relationships, might inform the use of technologies that support annotations when considering possible implementation approaches.

The case study also showed how one data model was used to define the data requirements required to support applications in a number of different usage contexts. The CDM provided the requirements detail to communicate usage behaviours concerning the auctioneer's salessheet system and the book keepers salesheet system, with each application specialising on its own specific use cases. The approach is not constraining as it allows the designer to decide what aspects of the design process adhere to the CDM, and whether they are influenced by it or not. The fact that implementation choices are dependent on

the data model and ethnography indirectly informs us of what design concerns to focus on, presents a different way of thinking of development.

Concerning mapping CDMs to implementation choices, we investigated an approach that enabled us to represent implementation choices as a taxonomy of component features represented in the CDM. We utilised a method that allowed us to indicate whether each implementation choice had full, limited or no support for a particular component feature. This information was useful in selecting which implementation choice was an appropriate approach for the system. It might however be useful to scale this approach further. What might be more interesting is to devise a way of appropriately indicating the extent to which each device fulfils a specified component feature using a wider measurement range, and working out a selection approach.

If we think about the relationship between requirements analysis, design and implementation more holistically, our choices of implementation architectures, devices and platforms are all informed by usage profiles. Tacit contracts in our CDM allow us to communicate usage profiles, and usage profiles allow us to select implementation choices. This is how requirements elicited using ethnography are modelled at the requirements level, and feed into various design level concerns.

Our work looked at how tacit information benefits the design process of a new system. Consequently this has set up the need to formally investigate a heuristic for mapping CDM's to implementation choices. This motivates us to propose the notion of an Infrastructure Ecology which summatively represents implementation level architectures, or so called implementation choices. As a first introduction to the concept, the next chapter will elaborate further on the point and specifically discuss the notion of mapping conceptual data models to infrastructure ecologies.

7

From Conceptual Data Models to Ecologies and Logical Data Models

> "A problem-implementation gap exists when a developer implements software solutions to problems using abstractions that are at a lower level than those used to express the problem." *France et. al* [29].

We are yet to see any work that specifies at a high level of abstraction, how system requirements map to feasible implementable solutions. The previous chapter demonstrated how conceptual data models can suggest directions for implementation. In this chapter, we introduce a reference methodology for moving from conceptual data models to infrastructure ecologies and Logical Data Models. We consider possible data views in the Software Development Life Cycle (SDLC) that may be affected by our approach. Furthermore, we present how the selection of infrastructure ecologies can be achieved given a particular set of requirements modelled in the Conceptual Data Model (CDM). Finally, we propose a metamodel for describing infrastructure ecologies and a CDM-LDM transformation. The metamodel is a reference architecture that describes the components that need to be assembled when considering ecologies.

7.1 What is an Ecology

Previous notions of ecologies refer to cooperative devices and digital ecosystems as "social" devices. Under this view, social devices have awareness of each other, and are capable of communicating with one another. It is also said that there is an understanding between devices. This includes governance on device coordination and collaboration, and also how inter-device interaction should be managed. In the literature, this type of ecology has been described as a *device ecology* by Loke [61]. In this thesis, we present a different perspective on ecologies known as 'infrastructure ecologies' (IE), hereafter we will use this term interchangeably with 'ecologies'.

An infrastructure ecology is a solution-instance or a possible implementation of a systems requirements: one of possibly many solutions. Infrastructure Ecologies bring together devices which could be more generally understood as ecology *instruments*. These instruments may be hardware, software, external entities and so on. The assemblage of these instruments creates an encapsulated view of how the requirements of the system can be satisfied - the ecology.

The focus of ecologies leans towards the architecture and organisation of the infrastructure required to satisfy requirements rather than on the social and synergistic behaviours which Loke's notion of device ecologies focuses on.

> If e is a way of satisfying a set of requirements R, then e is an ecology of R.

For example, under this notion, a spreadsheet can be regarded as an ecology of a requirement to represent data in an arrangement of rows and columns. This example can be scaled up to include the hardware devices that are required to run the spreadsheet software and so on.

Although ecologies refer to the final implementation choice, they are still design considerations that require some consideration in the design phase. An understanding of which ecologies are good for meeting which requirements can contribute to bridging the gap between the problem domain and the implementation domain. The work done in this chapter attempts to address the problem of moving from requirements models to logical models and ecologies.

7.2 Ecologies: A Model Driven Engineering Approach

The underlying motivation of Model Driven Engineering in software development is to simplify the process of mapping a system specification expressed as a model, to a model that targets the deployment platform of the system. There is an increased level of complexity when considering mapping requirements models to implementation designs and architectures. This problem is more pronounced when looking at multiple possible implementation choices in the absence of guidelines to aid a systems designer in discriminating between each possible choice - one problem with many possible solutions.

The problem statement is defined as follows:

Presented with a set of stakeholder requirements $\mathsf{R} = \{\mathsf{r}_1, \mathsf{r}_2, ..., \mathsf{r}_N\}$, and a set of possible ecologies (or implementation designs) $\mathsf{E} = \{\mathsf{e}_1, \mathsf{e}_2, ..., \mathsf{e}_T\}$ that each satisfy the requirements expressed in R, identify the ecology e_α that is the optimal ecology for R. As R is expressed as a CDM, the simplified problem statement is thus to find a model transformation for translating the CDM to e_α. The assumption is that the set of ecologies that satisfy the stakeholders requirements is known. As a result, the transformation is purely to facilitate the selection of the ecology.

With a rich CDM, there is enough requirements detail to enable comparisons between ecologies when investigating possible implementation choices. Later in this chapter, we show how we are able to specify what features each ecology supports and the selection method. In order to understand this approach, one must first understand the concept of data views in the SDLC and where CDM's, LDM's and ecologies fit into the larger picture. The next section elaborates on these points.

7.3 Data Views in the SDLC

In MDA, a view is a group of models that represent a particular concern of a system. In [35], a view is defined as a representation of a system from the perspective of a related set of concerns.

A *Data View* is a broad representation of the wider concerns of a system with respect to forms of data. Data Views can be seen as projections of the activities and development responsibilities within various levels of the SDLC.

Our assumption is that data is treated in 3 different forms during the software development cycle. Each form is confined to a Data View with a related set of concerns as shown in Figure 7.1. The broadest view is a series of transformations between the SDLC phases. In each of the individual stages of the tranformation, there are a number of models and activities.

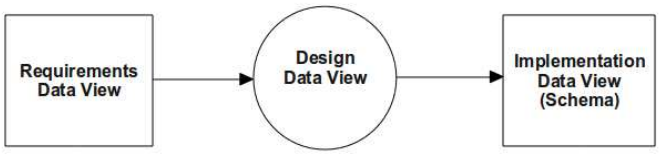


Figure 7.1: Data Views in the SDLC

It is important to note that these data views form part of a larger system. However we confine the discussion to activities and responsibilities in each data view that concern data. With this approach, there is some correspondence with model engineering concepts. We see correspondence between the Computation Independent Model (CIM) and Requirements Data View (RDV), Platform Independent Model (PIM) and Design Data View (DDV), and finally the Platform Specific Model (PSM) and Implementation Data View (IDV). Therefore:

$$\boldsymbol{RDV(CIM) \Rightarrow DDV(PIM) \Rightarrow IDV(PSM)}$$

These three views can be seen as contributing to the three forms of modeling in MDA and can thus be related to the CIM, PIM and PSM - from requirements to design to implementation. We describe these below:

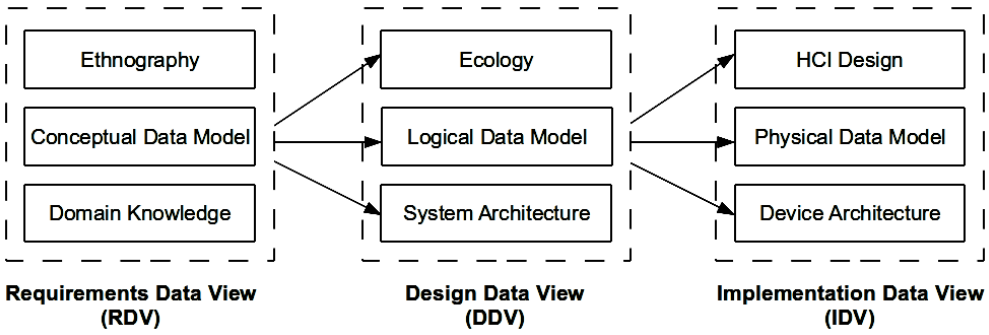


Figure 7.2: Detailed View of Data Views in the SDLC

Each view adds a layer of understanding of the wider concerns of the system for each of the SDLC phases:

i) Requirements Data View (RDV): This is the presentation of requirements centric client facing information which includes both functional and non-functional requirements, and a representation of tacit information pertaining to the information attributes and data relationships of the system. The RDV does not specifically describe system structure or hardware and software features and constructs. Therefore the CIM should not take into account devices, platforms and middleware. What is more important at this stage of system modeling, is the depiction of a detailed hybrid model that incorporates elements that derive from domain knowledge, standard requirements analysis and ethnography.

ii) Design Data View (DDV): refers to all design models of the system. It involves developing a schema that can be implemented for representation and storage of all the data members of the system, elicited from the RDV. Data concerns in the DDV include ecologies, logical data models and system architectures.

iii) Implementation Data View (IDV): This refers to all implementation-ready design specifications such as the physical data model, device architectures, and HCI Design. The implementation data model implements all explicit data members and relationships that have been conveyed from the design data model.

looking at the requirements data view, it maps to a range of models. The logical data model represents pure schemas without implementation considerations. From a storage / data representation point of view, the LDM is not optimised for storage. It also does not have tacit information, as this information is shifted into abstract system architectures and ecologies. The Physical Data Model (PDM) on the other hand is optimised for data, and thus relates directly to the implementation of storage schemas.

7.4 Components of Ecologies

We have defined ecologies as implementation designs that satisfy stakeholder requirements. As with device ecologies which we introduced previously, humans play an important role in an ecology model as they are the users of the system. This is because, amongst other reasons, the choice of software architectures is largely dependent on factors such as the volume of users, types of users and tacit user requirements etc that contribute to the decisions on what types of architectures and ecologies to use.

For the purpose of expressing the definition of infrastructure ecologies, we use the notions of *Instruments* and *Instrument Containers*. The ecology model defines a set of relationships and interactions between *instruments* and *instrument containers*. These relationships and interactions are governed by a number of key elements which we discuss later in this chapter. To begin with, we look at these concepts in a little bit more detail.

Instrument containers are physical containments made up of ecology *Instruments* such as hardware, software and/or concepts. These define the physical limitations of the embodied instruments and any constraints and rules that the ecology may hold. An instrument represents hardware and software within an ecology. It may also relate to architectural concepts within an Instrument Container or anything that can be 'part-of' the configuration of the architecture of a system. We refer to the following components of ecologies:

i) Instruments: represents the elements of the ecology

ii) Instrument Containers: represents the physical boundaries of the ecology constrained by a set of containment requirements.

iii) Associations: Represents relationships between instruments and instrument containers

iv) Features: Represents the capabilities of instruments and instrument containers

The next section proposes a metamodel that captures each of the components listed above.

7.5 An Ecology Metamodel

In this section we propose an ecology reference metamodel, which is an integral part of the CDM transformation. The metaclasses that comprise the ecology reference metamodel are depicted in Figure 7.3. The metamodel begins with the Instrument metaclass as the root. Instrument objects are named instances within the ecology and thus hold a name attribute. An Instrument object may define additional attributes relating its characteristics. The metamodel supports this via a one-to-many relationship between the Instrument class and the Attribute class.

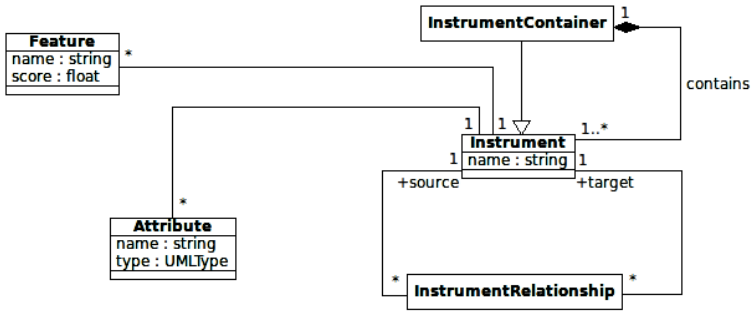


Figure 7.3: Metamodel for defining Ecologies in the CDM

Alongside Attribute objects, an Instrument object can specify a number of features (or characteristics) relating to its capabilities within the ecology. This

is enabled via a one-to-many relationship between the Instrument and Feature metaclass. Feature instances are an important part of the metamodel as they relate to tacit attributes and relationships in the CDM.

InstrumentContainer is a composition of Instrument objects to permit containment of ecology Instrument objects. The InstrumentRelationship models relationships that may exist between Instrument objects contained in an InstrumentContainer or between several InstrumentContainer objects contained in a larger InstrumentContainer object.

7.6 A Heuristic for Mapping Conceptual Data Models to Ecologies

The transformation is a mapping from the CDM in the RDV to the LDM and Ecology in the DDV. The process relies on a detailed CDM that contains tacit and explicit information. The main heuristic rule allows for a mapping between tacit information in the CDM and the features that characterise an ecology. This illustrates the importance and impact of the CDM in the overall picture, as the presence of tacit information in the CDM may entail the selection of a particular kind of ecology.

We propose some guidelines on how this can be done in a model driven approach. A CDM to LDM model transformation will do two things:

a) In the first instance, it will map the CDM data schema to a traditional LDM data schema, generally omitting tacit information. The LDM data schema will be used as the basis upon which to determine storage schemas in the Physical Data Model.

b) Secondly, it will map the CDM data schema to a larger architectural model, that takes into account the tacit information to develop an ecology solution that is suited to maintaining data in a way that preserves tacit requirements.

The first aspect of the transformation is a straightforward lossy mapping. The second aspect of the transformation involves a fitness function that computes a

score that summarises the extent to which an ecology satisfies the specification of the CDM. This enables a system designer to assess as a general heuristic guide, what ecology might be the best implementation approach to meet a set of requirements. The problem statement is defined as follows:

Presented with:

- a conceptual data model Q
- the set of tacit elements in Q:
  where T is the set of tacit elements and $\mathsf{T} = \{\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3, ..., \mathsf{t}_r\}$
- an Ecology E
- the set of instruments in E:
  where I is the set of ecology instruments and $\mathsf{I} = \{\mathsf{i}_1, \mathsf{i}_2, \mathsf{i}_3, ..., \mathsf{i}_s\}$

Define a fitness function f that computes a 'fitness' score of the Ecology E, that gives an indication of the extent to which E captures the set of tacit requirements T, defined in model Q.

CDM Ecology Fitness Function:
Given a CDM Q, we compute the fitness score of an Ecology E as follows:

$$\mathbf{f(E)} = \sum_{\mathbf{i \in I}} \sum_{\mathbf{t \in T}} \frac{\mathbf{Val(i,t)}}{\mathbf{p} * \mathbf{|i|}} * \frac{\mathbf{FeatureSum(Q,t)}}{\mathbf{q} * \mathbf{FeatureTotal(Q)}} \tag{7.1}$$

where:

- FeatureSum(Q,t) is the number of occurrences of feature t in model Q.
- FeatureTotal(Q) is the number of occurrences of features in model Q.
- Val(i,t) is the *feature score* of instrument i, given t as the feature
- |i| is the number of instruments in E.
- p and q are both constants with a value of 1, and can be changed depending on what part of the formula needs to be emphasised.

The feature score, Val(i,t), is the capability of for instrument i at feature t. The function gives weighting to an instrument according to the tacit feature expressed by t. Consider the comparison matrix of instruments against feature shown in Table 7.1.

Instrument	Feature t_1	Feature t_2	...	Feature t_r
i_1	i_1t_1	i_1t_2	...	i_1t_r
i_2	i_2t_1	i_2t_2	...	i_2t_r
...	...	...	...	
i_s	i_st_1	i_st_2	...	i_st_r

Table 7.1: Instrument / Feature Score Framework

We can proceed with the function Val(i,t) on any instrument i and feature t eg. $Val(i_1,t_2) \equiv i_1t_2$. The instrument / feature score framework functions as a feature comparison chart that can be used to compare ecologies based on tacit features highlighted in the CDM.

7.7 Summary

In this chapter, we introduced the notion of implementation design choices known singularly as an Ecology. An early reference approach to a model driven software development was proposed, for moving from conceptual data models in the requirements phases of the SDLC, to design level logical data models and ecologies. Also presented was a metamodel for constructing ecologies, and a heuristic guide for mapping CDM's to ecology instances. We envisage this work developing along a number of lines, to extended to a fully model driven approach to navigate tacit CDM's into LDM's and Architectures.

8

Conclusions

Given the increasing complexity of software systems, it is becoming more and more challenging to effectively capture and communicate stakeholder requirements throughout the software development process. With the problem of ineffective requirements being regarded as the main antecedent to software failure, it is necessary to investigate improved approaches for capturing, communicating and representing requirements, which aim to address the problem at the root cause. In requirements analysis, when considering models at a conceptual level, it has been noted that there is a divide between what knowledge is represented in the domain and the representation of domain knowledge at the conceptual level. In this regard, this thesis has investigated the proposition that tacit information (or implicit domain knowledge) is beneficial at the conceptual level of requirements analysis, to the passage of important contextual information to design. This chapter presents a summing-up of work done in this thesis beginning with a recap of the thesis objectives and an outline of the research directions. Following this, we discuss the thesis contributions and we propose some directions for extensions of this work.

8.1 Overview

Interdisciplinary research in requirements analysis and ethnography has greatly improved the understanding of analysis methods for elicitation and evaluation. The same body of research proposes that when you consider data from an

ethnographic perspective, there will always be tacit information essential to the nature of the data, but which is not part of its explicit or literal definition. With the increasing presence of tacit information in the business domain, this raises the importance of identifying knowledge of this dimension during the requirements analysis phase of software development.

It has been posited that when we follow traditional methods for requirements analysis, tacit information is inevitably lost or omitted in the elicitation phase. This is because traditional elicitation methods focus on explicit entities, relationships and usage behaviours in the system, rather than implicit kinds of information. Consequently, this can lead to problems like incomplete requirements, lack of understanding of the domain and so on. Despite this recognition, there has not been much movement of work in this area of recent. We have therefore pursued this hypothesis as the overarching theme of this thesis, hence the thesis title 'Architecting Tacit Information in Conceptual Data Models for Requirements Process Improvement'.

8.2 Research Goals

In this section, we reassert the objectives of this thesis. This research began with the objective to investigate the proposition that tacit information is beneficial at the requirements level. This objective incorporated a number of goals which directed the thesis to present a renewed effort to:

i) Bridging the gap between ethnography and requirements analysis.

ii) Investigate an approach to carrying ethnographic insights through the requirements analysis stage of software development into design.

iii) Devise an approach to navigating implementation choices at the design level, based on tacit information that has emerged through elicitation.

The next section presents the contributions of the thesis, structured in agreement with the set of proposed objectives.

8.3 Thesis Contributions

This work advances the field of requirements engineering by proposing a method that aims to address problems in elicitation, analysis, representation and communication of requirements in the SDLC. The following questions guided the investigation of methods for performing effective requirements: (i) How do we conduct 'good' elicitation? (ii) What constitutes effective analysis? (iii) What methods of representation are sufficient for structuring requirements? (iv) What are the right methods for communicating requirements between interested parties of the software development process?

We have investigated ethnographic methods for elicitation and analysis, and Model Driven Engineering methods for representation and communication to develop an approach for requirements engineering that aims to address each of the research goals of the thesis. In the process, we make important contributions to the field, and identify possible directions for future work which we elaborate on in the subsections that follow.

8.3.1 Bridging the Gap

The notion of "bridging the gap" relates to adjoining communication activities in the early phases of the SDLC. This encompasses gaining an understanding of the clients needs and effectively communicating the clients needs into the design phase of the SDLC.

An approach to ethnographic elicitation and evaluation

In chapter 3, we investigated the role of ethnography in conceptual data modeling, to submit the idea that ethnography is beneficial for requirements elicitation. The chapter thus sets the scene for how ethnographic studies contribute to a method for data analysis in RE. Contributing to the body of research on interdisciplinary methods, the chapter outlines an approach that incorporates ethnography during the early and final phases of the SDLC under the respective notions of Pre-Implementation Ethnography and Post-Implementation Ethnography. Pre-Implementation Ethnography is aimed at elicitation activi-

ties at the beginning of development, while Post-Implementation Ethnography is aimed at evaluation activities at the end of development.

This feeds into the proposition that tacit information is beneficial at the design level. The aim was to understand what ethnography offers from a requirements level. And what types of information it provides. Chapter 4 elaborated on this theme by investigating the potential benefits of incorporating tacit information in conceptual data models. The chapter demonstrated through a case study, how Pre-Implementation Ethnography works during the elicitation phase, and what information might get lost when not considering the ethnographic perspective. It also demonstrated how a data context can be scrutinized when trying to work out tacit knowledge.

Summary

In our approach we have bridged the gap by outlining an approach that (i) guides the elicitation of requirements during the early stages of the SDLC, and (ii) evaluates the final system in the final stages of the SDLC. However the approach would not be complete without a method for carrying ethnographic insights into design and a formalisation of the same. This forms the foundation for the next contribution of the thesis.

8.3.2 Investigate an approach to carrying ethnographic insights

Tacit Contracts

On the importance of bridging the gap, it was necessary to devise a method for carrying ethnographic insights into design. Chapter 5 led with the introduction of Tacit Contracts in Requirements Analysis aimed at formalising communication between requirements analysis and design in the SDLC. Tacit Contracts are design level obligations that ensure the passage of tacit information identified during Pre-Implementation Ethnography, through into design. The aim of tacit contracts is to make implicit knowledge in the problem domain visible in the design phase. Meanwhile, it is important that tacit information is distin-

guishable from knowledge of the explicit kind. In this way, the quality of the requirements model will be enhanced with domain knowledge which may lead to enhanced design and implementation choices in subsequent SDLC phases.

An Improved Metamodel for Defining Conceptual Data Models

To formalise the notion of tacit contracts, in Chapter 5 we proposed a formal definition for defining semantically rich conceptual data models – a tacit requirements metamodel. The metamodel was proposed to permit the construction of CDM's that incorporated both tacit information and explicit information: a multidimensional model for data. To support the requirement for distinguishable information types in the model, a UML stereotype was created to identify tacit information in the model. Models elements annotated with the $<<$ ***tacit*** $>>$ stereotype enable tacit / implicit elements to be distinguished from explicit elements.

A Modeling Language for Describing Tacit Requirements

The formal semantics of the tacit requirements metamodel was defined using Constructive Type Theory (CTT). CTT provides the benefits of an open extensible formal system. It allows the freedom of adjusting, adding or removing rules as needed. To this end, we developed the formalism for creating rules and constraints over elements in a CDM. The language is intended to be used in the requirements analysis phase to model observations derived from Pre-Implementation Ethnography, leading to a set of rules of the system.

Summary

In our investigation of approaches to carrying ethnographic insights into design, we introduced the notion of tacit contracts that create design obligations between requirements and design. Furthermore, we introduced a rich CDM metamodel that enables both implicit and explicit information to be constructed on conceptual models. This paves the way for work aimed at improving implementation ecology selection at the design level.

8.3.3 Devise an approach to navigating implementation choices

With the development of a metamodel that permits the construction of enhanced Conceptual Data Models, more detailed analysis can be performed as part of the design phase to improve the design process. Such analysis may include more specialised examinations of tacit and explicit information in the CDM, which may lead to better decision making during the selection of possible implementation choices. The idea is that with the introduction of tacit information in the CDM can delineate possible directions for implementations.

Infrastructure Ecologies

In chapter 7 we introduced the concept of Ecologies to describe the implementation choices of a system that satisfy a set of requirements. A metamodel was proposed as the formal definition of Ecologies which enables the creation of design models that concern the solution domain. Promising results suggested possible mappings from conceptual models to ecologies and logical data models. To this end, an early reference architecture to a model driven development approach was proposed to map CDM's to ecology instances. This provides a necessary first step for effectively linking CDM's with models at the design level.

Summary

In our aim to devise an approach to navigating implementation choices at the design level, we introduced the concept of Ecologies aimed at describing instances of the solution domain that meet a particular set of requirements in the problem domain. A formal definition of ecologies was defined via the means of a UML metamodel. Furthermore a method of ecology selection was proposed. Suggested directions for future work are to extend the work to a fully model driven approach to navigate Tacit CDM's into LDM's and Architectures.

8.4 Future Work

This thesis has made a number of contributions aimed at advancing methods in Requirements Analysis. Given the merits of the contributions, there are some limitations that necessitate future extensions. These extensions are suggested below.

i) Futher refinement of our approach to conceptual modeling. At the conceptual modeling phase, it is unfeasible to include all bodies of tacit information in the problem domain as prospective requirements level candidates to aid the design process. It would be of interest to establish further understanding of how to identify what types of tacit information are 'important' and 'relevant' for the the design phase, so as not to overwhelm the conceptual model with a colossal amount of information from the problem domain.

ii) The work done in this thesis has taken a view on requirements process improvement for data. However the approach is agnostic and can be applied in other domains. Further work is suggested to carry this approach into other application domains outside data modeling.

iii) The insight obtained from this work shows that there is potential in the work on mapping Conceptual Data Models to Ecologies. An initial description of a method is presented in this work which generally defines the approach. Further work is suggested to expand this approach especially in the unexplored area of investigating the effect of ecology selection on the logical data model and subsequent concerns of the physical data model.

iv) Finally, it would be of interest to further investigate this approach as a fully automated model driven method that takes in a conceptual data model and subsequently transforms it into its corresponding logical and physical concerns.

8.5 Concluding Remarks

As software systems become increasingly complex, there will continue to be a need to advance methods for understanding the problem domain and effective methods to communicate our understanding of the problem domain to the solution domain. In this thesis, we have demonstrated the concept of architecting tacit information in conceptual data models for requirements process improvement. To approach this concept, we investigated the role of ethnography in conceptual modeling[1]. This enabled us to pursue the hypothesis of the advantages to be gained from incorporating tacit information within conceptual data models[2]. Promising results from preliminary empirical work directed the creation of a tacit requirements metamodel[3] which paved the way for a larger evaluation of our work via a case study in auction house systems design[4]. Results suggested possible mappings of design concerns from conceptual models to ecologies and logical data models[5]. Our approach is successful in that it has considered persistent problems rooted in RE, and while we do not claim to solve them all, our work presents a first step in a seemingly alternative, but actually relevant and complementary direction for requirements process improvement.

[1]Chapter 3: Role of Ethnography in Conceptual Data Modeling, 51–76
[2]Chapter 4: Incorporating Tacit Information within Conceptual Data Models, 77–93
[3]Chapter 5: A Tacit Requirements Metamodel, 94–123
[4]Chapter 6: A Case Study in Auction House Systems Design, 125–182
[5]Chapter 7: From Conceptual Data Models to Ecologies and Logical Data Models, 183–192

References

[1] *1st international enterprise distributed object computing conference (edoc '97), 24-26 october 1997, gold coast, australia, proceedings*, in EDOC, IEEE Computer Society, 1997.

[2] M. Adato, *Integrating survey and ethnographic methods to evaluate conditional cash transfer programs:*, IFPRI discussion papers 810, International Food Policy Research Institute (IFPRI), 2008.

[3] M. I. Aguirre-Urreta and G. M. Marakas, *Comparing conceptual modeling techniques: a critical review of the EER vs. OO empirical literature*, SIGMIS Database, 39 (2008), pp. 9–32.

[4] A. Al-Rawas and S. Easterbrook, *Communication problems in requirements engineering: A field study*, in Proc. of Conf. on Prof. on Awareness in Software Engineering, 1996, pp. 47–60.

[5] R. J. Anderson, *Representations and requirements: the value of ethnography in system design*, Hum.-Comput. Interact., 9 (1994), pp. 151–182.

[6] Anoto Group AB, *Anoto Digital Pen Technology*, `http://www.anoto.com`, 2012. Last Accesssed September 2012.

[7] M. Aoyama, *Persona-and-scenario based requirements engineering for software embedded in digital consumer products*, in Proceedings of the 13th IEEE International Conference on Requirements Engineering, RE '05, Washington, DC, USA, 2005, IEEE Computer Society, pp. 85–94.

[8] Y. Asnar, P. Giorgini, and J. Mylopoulos, *Goal-driven risk assessment in requirements engineering*, Requir. Eng., 16 (2011), pp. 101–116.

[9] L. J. Ball and T. C. Ormerod, *Applying ethnography in the analysis and support of expertise in engineering design*, Design Studies, 21 (2000), pp. 403 – 421.

[10] B. Baudry, C. Nebut, and Y. Le Traon, *Model-driven engineering for requirements analysis*, in Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International, Oct. 2007, p. 459.

[11] R. Bentley, J. A. Hughes, D. Randall, T. Rodden, P. Sawyer, D. Shapiro, and I. Sommerville, *Ethnographically-informed systems design for air traffic control*, in Proceedings of the 1992 ACM conference on Computer-supported cooperative work, CSCW '92, New York, NY, USA, 1992, ACM, pp. 123–129.

[12] P. Beynon-Davies, *Ethnography and information systems development: Ethnography of, for and within is development*, Inf. Softw. Technol., 39 (1997), pp. 531–540.

[13] H. S. Bhola, *Evaluating "literacy for development" projects, programs and campaigns: Evaluation planning, design and implementation, and utilization of evaluation results*, tech. rep., UNESCO Institute for Education; DSE [German Foundation for International Developement], Hamburg, Germany, 1990.

[14] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language user guide*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1999.

[15] A. Brady, M. Seigel, T. Vosecky, and C. Wallace, *Addressing communication issues in software development: A case study approach*, in Software Engineering Education Training, 2007. CSEET '07. 20th Conference on, 2007, pp. 301–308.

[16] K. Cass, *A bridge to the third age: fashioning technological concepts for novice over-seventy home-system users*, in Proceedings of the 2006 ACM SIGMIS CPR conference on computer personnel research: Forty four years of computer personnel research: achievements, challenges & the future, SIGMIS CPR '06, New York, NY, USA, 2006, ACM, pp. 20–24.

[17] J. Castro, S. Acua, and N. Juristo, *Integrating the personas technique into the requirements analysis activity*, in Computer Science, 2008. ENC '08. Mexican International Conference on, oct. 2008, pp. 104 –112.

[18] P. P.-S. Chen, *The entity-relationship modeltoward a unified view of data*, ACM Trans. Database Syst., 1 (1976), pp. 9–36.

[19] B. H. C. Cheng and J. M. Atlee, *Research directions in requirements engineering*, in FOSE '07: 2007 Future of Software Engineering, Washington, DC, USA, 2007, IEEE Computer Society, pp. 285–303.

[20] E. Coakes and T. Elliman, *Focus issue on legacy information systems and business process engineering: the role of stakeholders in managing change*, Commun. AIS, 2 (1999).

[21] S. A. Conger, *The New Software Engineering*, Course Technology Press, Boston, MA, United States, 1st ed., 1993.

[22] A. Cooper, *The Inmates Are Running the Asylum*, Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 1999.

[23] R. Damaeviius, *On the human, organizational, and technical aspects of software development and analysis*, in Information Systems Development, Springer US, 2009, pp. 11–19.

[24] P. Dourish, *Implications for design*, in Proc. ACM Conf. Human Factors in Computing Systems CHI 2006, ACM Press, 2006, pp. 541–550.

[25] H. El-Ghalayini, M. Odeh, and R. McClatchey, *Developing ontology-driven conceptual data models*, in Proceedings of the 1st International Conference on Intelligent Semantic Web-Services and Applications, ISWSA '10, New York, NY, USA, 2010, ACM, pp. 4:1–4:6.

[26] E. Eriksson, Å. Cajander, and J. Gulliksen, *Hello world! - experiencing usability methods without usability expertise*, in INTERACT (2), 2009, pp. 550–565.

[27] D. E. Forsythe, *It's Just a Matter of Common Sense: Ethnography as Invisible Work*, Computer Supported Cooperative Work (CSCW), 8 (1999), pp. 127–145.

[28] B. E. Foucault, *Techniques for researching and designing global products in an unstable world: A case study. human factors and computer systems. conference (4th 2004). proceedings of the chi conference on human factors in computing systems*, in University of Greenwich, ACM Press, 2004, pp. 1481–1484.

[29] R. France and B. Rumpe, *Model-driven development of complex software: A research roadmap*, in 2007 Future of Software Engineering, FOSE '07, Washington, DC, USA, 2007, IEEE Computer Society, pp. 37–54.

[30] R. Fuentes-Fernndez, J. Gmez-Sanz, and J. Pavn, *Understanding the human context in requirements elicitation*, Requirements Engineering, 15 (2010), pp. 267–283. 10.1007/s00766-009-0087-7.

[31] H. Garfinkel, *Studies in Ethnomethodology*, Prentice Hall, Englewood Cliffs, NJ, 1967.

[32] J. Goguen, *Social issues in requirements engineering*, in Requirements Engineering, 1993., Proceedings of IEEE International Symposium on, Jan 1993, pp. 194–195.

[33] J. Goguen and C. Linde, *Techniques for requirements elicitation*, in IEEE International Symposium on Requirements Engineering San Diego, CA, Academic Press, 1993, pp. 152–164.

[34] I. A. W. Group, *Ieee std 1471-2000, recommended practice for architectural description of software-intensive systems*, tech. rep., IEEE, 2000.

[35] I. A. W. Group, *Ieee std 1471-2000, recommended practice for architectural description of software-intensive systems*, tech. rep., IEEE, 2000.

[36] O. M. Group, *Corba component model 4.0 specification*, Specification Version 4.0, Object Management Group, April 2006.

[37] O. M. Group, *Software Process Engineering Metamodel (SPEM)*, Jan. 2007.

[38] M. Hammersley and P. Atkinson, *Ethnography. Principles in practice*, Tavistock Publications, London, 1983.

[39] R. R. Harper, J. A. Hughes, and D. Z. Shapiro, *Harmonious working and CSCW: computer technology and air traffic control*, North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 1991, pp. 225–234.

[40] C. Heath, J. Hindmarsh, and P. Luff, *Video Analysis and Qualitative Research*, SAGE, 2010.

[41] C. Heath, M. Jirotka, P. Luff, and J. Hindmarsh, *Unpacking collaboration: the interactional organisation of trading in a city dealing room*, Computer Supported Cooperative Work (CSCW), 3 (1994), pp. 147–165. 10.1007/BF00773445.

[42] C. Heath and P. Luff, *Documents and professional practice: b̈adörganisational reasons for g̈oodc̈linical records*, in Proceedings of the 1996 ACM conference on Computer supported cooperative work, CSCW '96, New York, NY, USA, 1996, ACM, pp. 354–363.

[43] C. Heath, P. Luff, and G. Cambridge, *Collaboration and control: Crisis management and multimedia technology in london underground line control rooms*, Computer Supported Cooperative Work, 1 (1992), pp. 69–94.

[44] A. Herrmann and B. Paech, *Practical challenges of requirements prioritization based on risk estimation*, Empirical Softw. Engg., 14 (2009), pp. 644–684.

[45] J. Hughes, V. King, T. Rodden, and H. Andersen, *Moving out from the control room: ethnography in system design*, in CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work, New York, NY, USA, 1994, ACM, pp. 429–439.

[46] J. Hughes, J. O'Brien, T. Rodden, M. Rouncefield, and I. Sommerville, *Presenting ethnography in the requirements process*, in Proceedings of the Second IEEE International Symposium on Requirements Engineering, RE '95, Washington, DC, USA, 1995, IEEE Computer Society, pp. 27–.

[47] J. A. Hughes, D. Randall, and D. Shapiro, *Faltering from ethnography to design*, in Proceedings of the 1992 ACM conference on Computer-supported cooperative work, CSCW '92, New York, NY, USA, 1992, ACM, pp. 115–122.

[48] R. Iqbal, R. Gatward, and A. James, *A general approach to ethnographic analysis for systems design*, in Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information, SIGDOC '05, New York, NY, USA, 2005, ACM, pp. 34–40.

[49] M. Jackson, *Problem frames: analyzing and structuring software development problems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[50] B. Jeffrey, P. Bradshaw, P. Twining, and C. Walsh, *Ethnography, education and on-line research*, in European Conference of Educational Research, August 2010.

[51] M. Jirotka, N. Gilbert, and P. Luff, *On the social organisation of organisations*, Computer Supported Cooperative Work (CSCW), 1 (1992), pp. 95–118. 10.1007/BF00752452.

[52] M. Jirotka, P. Luff, and C. Heath, *Requirements for technology in complex environments: tasks and interaction in a city dealing room*, ACM SIGOIS Bulletin, 14 (1993), pp. 17–23.

[53] B. Jordan, *Ethnographic workplace studies and computer supported cooperative work*, in in Interdisciplinary Workshop on Informatics and Psychology, D. Shapiro, M. Tauber, and R. Traunmueller, eds., The design of computer-supported cooperative work and groupware systems, Amsterdam: North Holland/Elsevier, 1993, pp. 17–42.

[54] E. Kindler, *Model-based software engineering: the challenges of modelling behaviour*, in Proceedings of the Second International Workshop on Behaviour Modelling: Foundation and Applications, BM-FA '10, New York, NY, USA, 2010, ACM, pp. 4:1–4:8.

[55] G. Kotonya and I. Sommerville, *Viewpoints for requirements definition*, Software Engineering Journal, 7 (1992), pp. 375–387.

[56] D. Leffingwell and D. Widrig, *Managing software requirements: a unified approach*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[57] T. C. Lethbridge, S. E. Sim, and J. Singer, *Software anthropology: Performing field*, in Lingua, 2004, pp. 281–319.

[58] S. L. Lim and A. Finkelstein, *Stakerare: Using social networks and collaborative filtering for large-scale requirements elicitation*, IEEE Transactions on Software Engineering, 38 (2012), pp. 707–735.

[59] O. Lindeberg, *What are software practice studies good for?*, in in Empirical Software Research, workshop at the 22 nd ICSE conference, 2000.

[60] P. M. Löf, *Constructive Mathematics and Computer Programming*, in 6-th International Congress for Logic, Methodology and Philosophy of Science, 1979, North–Holland, 1982, pp. 153–175.

[61] S. Loke, *Service-oriented device ecology workflows*, in Service-Oriented Computing - ICSOC 2003, M. Orlowska, S. Weerawarana, M. Papazoglou, and J. Yang, eds., vol. 2910 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2003, pp. 559–574.

[62] N. Maiden and A. Gizikis, *Where do requirements come from?*, Software, IEEE, 18 (2001), pp. 10 –12.

[63] N. A. M. Maiden and S. Robertson, *Integrating creativity into requirements processes: Experiences with an air traffic management system*, in RE, 2005, pp. 105–116.

[64] D. Martin, J. Rooksby, M. Rouncefield, and I. Sommerville, *'good' organisational reasons for 'bad' software testing: An ethnographic study of testing in a small software company*, in Proceedings of the 29th international conference on Software Engineering, ICSE '07, Washington, DC, USA, 2007, IEEE Computer Society, pp. 602–611.

[65] P. Martin-Löf, *Intuitionistic Type Theory. Notes by Giovanni Sambin of a series of lectures given in Padua, June 1980*, Bibliopolis, Napoli, 1984.

[66] L. D. Miles, *Techniques of value analysis*, (1951).

[67] D. R. Millen, *Rapid ethnography: time deepening strategies for hci field research*, Proceedings of the conference on Designing interactive systems processes practices methods and techniques DIS 00, 280-286htt (2000), pp. 280–286.

[68] J. Miller and J. Mukerji, *Mda guide version 1.0.1*, Tech. Rep. omg/03-06-01, Object Management Group (OMG), June 2003.

[69] R. K. Mitchell, B. R. Agle, and D. J. Wood, *Toward a theory of stakeholder identification and salience: Defining the principle of who and what really counts*, Oct. 1997.

[70] I. Nonaka, *A Dynamic Theory of Organizational Knowledge Creation*, Organization Science, 5 (1994), pp. 14–37.

[71] I. Nonaka, R. Toyama, and P. Byosire, *A theory of organizational knowledge creation: understanding the dynamic process of creating knowledge*, Oxford Univ. Press, Oxford, 2001, p. 491517.

[72] B. Nuseibeh and S. Easterbrook, *Requirements engineering: a roadmap*, in ICSE '00: Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, 2000, ACM, pp. 35–46.

[73] Object Management Group, *Meta Object Facility (MOF) Specification — Version 1.4*, April 2002.

[74] Oxford Dictionaries, *tacit. Oxford Dictionaries*, `http://oxforddictionaries.com/definition/tacit`. Last Accesssed April 2011.

[75] C. Pettinari, C. C. Heath, P. Luff, and J. M., *Notes toward an applied ethnography*, tech. rep., Work Interaction Technology, 1998.

[76] S. Pfleeger and J. Atlee, *Software Engineering: Theory and Practice*, Pearson Prentice Hall, 2006.

[77] L. Plowman, Y. Rogers, and M. Ramage, *What are workplace studies for?*, in Proceedings of the fourth conference on European Conference on Computer-Supported Cooperative Work, Norwell, MA, USA, 1995, Kluwer Academic Publishers, pp. 309–324.

[78] K. Pohl, *Requirements Engineering - Fundamentals, Principles, and Techniques*, Springer, 2010.

[79] M. Polanyi, *The tacit dimension*, Anchor Books, Garden City, 1967.

[80] J. Poole and D. Mellor, *Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration*, John Wiley & Sons, Inc., New York, NY, USA, 2001.

[81] C. Potts, *Metaphors of intent*, in Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on, 2001, pp. 31 –38.

[82] O. Preiss and A. Wegmann, *Stakeholder discovery and classification based on systems science principles*, in Proceedings of the Second Asia-Pacific Conference on Quality Software, APAQS '01, Washington, DC, USA, 2001, IEEE Computer Society, pp. 194–.

[83] K. Rönkkö, O. Lindeberg, and Y. Dittrich, *'bad practice' or 'bad methods' " are software engineering and ethnographic discourses incompatible?*, in ISESE '02: Proceedings of the 2002 International Symposium on Empirical Software Engineering, Washington, DC, USA, 2002, IEEE Computer Society, p. 204.

[84] T. Salvador, G. Bell, and K. Anderson, *Design ethnography*, Design Management Journal (Former Series), 10 (1999), pp. 35–41.

[85] E. Sanders, *Ethnography in npd research-how applied ethnography can improve your npd research process*, 2002.

[86] D. Schwartz, *Visual ethnography: Using photography in qualitative research*, Qualitative Sociology, 12 (1989), pp. 119–154. 10.1007/BF00988995.

[87] H. Sharp, A. Finkelstein, and G. Galal, *Stakeholder identification in the requirements engineering process*, Database and Expert Systems Applications, International Workshop on, 0 (1999), p. 387.

[88] B. Solemon, S. Sahibuddin, and A. A. A. Ghani, *Requirements engineering problems and practices in software companies: An industrial survey*, in Advances in Software Engineering, D. lzak, T.-h. Kim, A. Kiumi, T. Jiang, J. Verner, and S. Abraho, eds., vol. 59 of Communications in Computer and Information Science, Springer Berlin Heidelberg, 2009, pp. 70–77.

[89] I. Sommerville, T. Rodden, P. Sawyer, R. Bentley, and M. Twidale, *Integrating ethnography into the requirements engineering process*, 1993.

[90] A. G. Sutcliffe and N. A. M. Maiden, *Bridging the requirements gap: policies, goals and domains*, in Proceedings of the 7th international workshop on Software specification and design, IWSSD '93, Los Alamitos, CA, USA, 1993, IEEE Computer Society Press, pp. 52–55.

[91] A. Taylor, *IT projects sink or swim*, British Computer Society, `http://archive.bcs.org/bulletin/jan00/article1.htm`, 2000. Last Accesssed December 2010.

[92] A. Van Lamsweerde, *Goal-oriented requirements engineering: A guided tour*, in Proceedings of the Fifth IEEE International Symposium on Requirements Engineering, RE '01, Washington, DC, USA, 2001, IEEE Computer Society, pp. 249–.

[93] J. Verner, K. Cox, S. Bleistein, and N. Cerpa, *Requirements engineering and software project success: An industrial survey in australia and the u.s.*, Australia Journal of Information Systems, 13 (2004), pp. 225–238.

[94] S. Viller and I. Sommerville, *Social analysis in the requirements engineering process: From ethnography to method*, Requirements Engineering, IEEE International Conference on, 0 (1999), p. 6.

[95] S. Viller and I. Sommerville, *Ethnographically informed analysis for software engineers*, International Journal of Human-Computer Studies, 53 (2000), pp. 169 – 196.

[96] A. Voss, R. Procter, R. Slack, M. Hartswood, and M. Rouncefield, *Design as and for collaboration: Making sense of and supporting practical action*, in Configuring User-Designer Relations, M. s, R. Slack, M. Rouncefield, R. Procter, M. Hartswood, and A. Voss, eds., Computer Supported Cooperative Work, Springer London, 2009, pp. 1–28.

[97] Y. Wand and R. Weber, *Research commentary: Information systems and conceptual modeling–a research agenda*, Info. Sys. Research, 13 (2002), pp. 363–376.

[98] J. Warmer, A. Kleppe, and W. Bast, *MDA Explained: The Model Driven Architecture–Practice and Promise*, Object Technology Series, Addison Wesley, 1st edition ed., 2003.

[99] R. J. Wieringa, *Requirements engineering: frameworks for understanding*, John Wiley & Sons, Inc., New York, NY, USA, 1996.

[100] G. K. Williams and I. Poernomo, *Social Computing Theory and Practice: Interdisciplinary Approaches*, IGI Global, Information Science Reference, USA/UK, 2011, ch. Social Contexts in an Information Rich Environment, pp. 68–84.

[101] G. K. Williams, I. Poernomo, and P. Luff, *Modelling ethnographic analyses for records via tacit contracts*, in RCIS, 2011.

[102] R. R. Young, *The requirements engineering handbook*, Artech House technology management and professional development library, Artech House, Norwood, MA, 2004.