A CRITICAL EVALUATION OF OBJECT-ORIENTED ANALYSIS AND DESIGN METHODS WITH A SPECIAL FOCUS ON OBJECT FORMULATION

by

Helen Campbell, Cert Ed, BSc (Hons)

A thesis submitted in partial fulfilment for the requirements of the award of Master of Philosophy

University of Central Lancashire
May 2001

Contents

Chapter One: Introduction **11**
1.1 Research Question 11
1.2 The importance of this area of research. 14
1.3 What else has been done? 17
1.4 Research Method 24
1.5 Benefits of the research 24
1.6 Structure of the Thesis 24
1.7 Summary 25
Chapter Two: Research Methods **26**
2.1 Introduction 26
2.2 Research Methods 26
2.2.1 Conceptual Study 28
2.2.2 Laboratory experiments 28
2.2.3 Field experiments 29
2.2.4 Surveys 29
2.2.5 Case Studies 30
2.2.6 Phenomenology / Hermeneutics 32
2.2.7 Action Research 34
2.3 Choice of action research and the implications. 36
2.4 Summary 39
Chapter Three: Object Orientation and Object-Oriented Systems Development Methods **40**
3.1 Terminology 40
3.1.1 Methodology and Method 41
3.1.2 Method and Operation 42
3.2 Development of object-oriented methods 43
3.3 Object-Orientation 43
3.3.1 Object-oriented thinking 45
3.3.2 Object system 46
3.4 Themes in object orientation 46
3.4.1 Abstraction 46
3.4.2 Encapsulation 47
3.4.3 Inheritance 47
3.5 Development of object-oriented analysis and design methods 50
3.6 Brief overview of object-oriented analysis methods. 52
3.6.1 Event driven approach to object-oriented analysis 52
3.6.1.1 Shlaer and Mellor's Object Oriented Systems Analysis (OOSA) 52
3.6.1.2 Coad and Yourdon's Object-Oriented Analysis (OOA) 53
3.6.2 Model-driven approach 54
3.6.2.1 Rumbaugh et al's Object Modelling Technique (OMT) 54
3.6.2.2 Embley's Object-oriented systems analysis (OSA) 55
3.6.2.3 Martin and Odell Ptech 55
3.6.3 Use case driven approach 56
3.6.3.1 Jacobson et al's Object-Oriented Software Engineering (OOSE) 56
3.6.4 Role driven approach 57
3.6.5 Responsibility driven approach 58

3.6.6 Unified Modeling Language (UML) 59
3.7 Justification of choice of methods to be evaluated 59
3.7.1 OOA (Coad and Yourdon 1991) 60
3.7.2 Object-Oriented Modelling and Design (Rumbaugh et al 1991) 60
3.7.3 Object Oriented Software Engineering (Jacobson et al 1998) 61
3.8 Summary 61
Chapter Four: Evaluation Framework 63
4.1 Background to Methodologies 63
4.1.1 Comparison of Methodologies 63
4.2 Strategies for comparing methodologies 64
4.2.1 Taxonomies 64
4.2.2 Visual Comparisons 65
4.2.3 Descriptive comparisons 65
4.2.4 Structured thinking 66
4.2.5 Frameworks 68
4.3 A Framework for Comparing Methodologies - Avison and Fitzgerald 68
4.3.1 Philosophy 69
4.3.2 Model 70
4.3.3 Techniques and tools 70
4.3.4 Scope 71
4.3.5 Outputs 71
4.3.6 Practice 71
4.3.7 Product 71
4.4 MULTIVIEW 71
4.5 A Framework for Method Evaluation – NIMSAD 74
4.5.1 Introduction to the NIMSAD framework 74
4.5.2 The 'problem situation' 76
4.5.3 The intended problem solvers and their 'mental constructs'. 77
4.5.4 The Problem Solving Process 79
4.5.5.1 Stage 1: Understanding the 'situation of concern' 80
4.5.5.2 Stage 2: Performing the diagnosis 81
4.5.5.3 Stage 3: Defining the prognosis outline 82
4.5.5.4 Stage 4: Defining problems 83
4.5.5.5 Stage 5: Deriving notional systems 83
4.5.6 Phase 2 - solution design 84
4.5.6.1 Stage 6: Performing the conceptual/logical design. 84
4.5.6.2 Stage 7: Performing the physical design. 84
4.6 Phase 3: Design implementation. 85
4.6.1 Stage 8: Implementing the design. 85
4.7 Methodology evaluation 86
4.8 Justification of choice of Evaluation Framework 89
4.9 Summary 90
Chapter Five: Critical Evaluation of Object Identification in Object Oriented Analysis (OOA). 91
5.1 Introduction to Object Oriented Analysis (OOA) 91
5.2 Evaluation of OOA focused on Object identification 92
5.2.1 Element 1:The 'problem situation' 92
5.2.2 Element 2: The intended problem solver 94
5.2.3 Element 3: The problem solving process 98
5.2.3.1 Stage (i) Understanding the 'Situation of concern' 99

5.2.3.1.1 Investigation models and techniques 99
5.2.3.2 Stage (ii) Performing the Diagnosis 102
5.2.3.3 Stage (iii) Defining the prognosis outline 104
5.2.3.4 Stage (iv) Defining problems 105
5.2.3.5 Stage (v) Deriving Notional Systems 105
5.2.3.6 Stage (vi) Performing Conceptual / Logical Design 106
5.2.3.7 Stage (vii) Performing the Physical Design 107
5.2.3.8 Stage (viii) Implementing the Design 108
5.2.4 Element 4: Evaluation 108
5.3 Summary 109
Chapter Six: Critical Evaluation of Object Identification in Object Modelling Technique 111
6.1 Introduction to Object Modelling Technique (OMT) 111
6.2 Evaluation of OMT focused on Object Identification 113
6.2.1 Element 1: The 'problem situation' 113
6.2.2 Element 2: The intended problem solver 117
6.2.3 Element 3: The problem solving process 120
6.2.3.1 Stage (i) Understanding the 'Situation of concern' 120
6.2.3.1.1 Investigation models and techniques 122
6.2.3.2 Stage (ii) Performing the Diagnosis 122
6.2.3.3 Stage (iii) Defining the Prognosis Outline 129
6.2.3.4 Stage (iv) Defining problems 129
6.2.3.5 Stage (v) Deriving Notional Systems 130
6.2.3.6 Stage (vi) Performing Conceptual / Logical Design 131
6.2.3.7 Stage (vii) Performing the Physical Design 131
6.2.3.8 Stage (viii) Implementing the Design 132
6.2.4 Element 4: Evaluation 132
6.4 Summary 133
Chapter Seven: Critical Evaluation of Object-Oriented Software Engineering 136
7.1 Introduction to Object Oriented Software Engineering (OOSE) 136
7.1.1 Use case driven 137
7.2 Evaluation of OOSE focused on Object Identification 139
7.2.1 Element 1: The 'problem situation' 139
7.2.2 Element 2: The problem solver 142
7.2.3 Element 3: The problem solving process 145
7.2.3.1 Stage (i) Understanding the 'Situation of concern' 145
7.2.3.1.1 Investigation models and techniques 147
7.2.3.2 Stage (ii) Performing the Diagnosis 148
7.2.3.3 Stage (iii) Defining the prognosis outline 156
7.2.3.4 Stage (iv) Defining problems 156
7.2.3.5 Stage (v) Deriving Notional Systems 157
7.2.3.6 Stage (vi) Performing Conceptual / Logical Design 157
7.2.3.7 Stage (vii) Performing the Physical Design 158
7.2.3.8 Stage (viii) Implementing the Design 158
7.2.4 Element 4: Evaluation 159
7.3 Summary 159
Chapter Eight: Finding Objects in Practice 163
8.1 Sampling 164
8.2 Investigation Method 164
8.3 Raw Data 165
8.4 Analysis 165

8.4.1 Problem Situation 166
8.4.2 Problem Solver 166
8.4.3 Problem Solving Process 167
8.5 Lessons Learned 170
Chapter Nine: Conclusions 171
9.1 Major Findings 171
9.2 Conclusions on object formulation from a methodological perspective 175
9.3 Conclusions drawn from evaluated methodologies 180
9.4 Conclusions drawn from the use of NIMSAD 182
9.4.1 Problem Formulation 184
9.4.2 Framework as a basis for learning 185
9.5 Summary 186
9.6 Future Work 188
References 190

Tables and Figures

Figure 2.1 A Framework to aid choice in information systems research approaches ..27
Figure 2.2 Organised use of rational thought 37
Table 2.1 A Comparison of Positivist Science and Action Research 39
Figure 3.1 Multiple Inheritance 49
Figure 4.1 Ontological notion of a 'system' 67
Figure 4.2 Epistemological notion of 'systems' 68
Figure 4.3 The Multiview Framework 72
Table 4.1 Methodology Outputs 73
Figure 4.5 NIMSAD elements and stages. 75
Figure 4.6 Deriving notional systems 84
Figure 4.7 The Problem Solving Process 86
Table 4.1 Evaluation issues. 88
Table 6.1 Criteria used to refine candidate objects 124
Figure 6.1 Identification of relevant classes 125
Table 6.2 Criteria used to refine associations 126
Figure 6.2 Associations from ATM problem statement 127
Table 6.3 Criteria used to refine attributes 127
Table 6.4 Criteria used to identify missing objects 128
Figure 7.1 Systems development viewed as a process of model building 137
Figure 7.2 A Process changes one model into another 137
Table 7.1 Problem Solving process in OOSE 138
Figure 7.3 Can, bottle and crate have common properties inherited from Deposit Item 152
Figure 7.4 Interface objects in the recycling machine, customer panel, operator panel, receipt printer and alarm device 153
Figure 7.5 The objects supporting the use case Returning Item 156
Table 8.1 Population sampling 164
Table 8.2 Raw Data 165
Table 9.1 Issues raised during evaluation 187

Acknowledgements

I wish to thank my colleagues and friends within the Department of Computing for their support while I have been involved in this research. Jean Richards, Head of Department, for her support, forbearance and encouragement without which I could not have continued. Barbara McManus, Pietro Murano, Linda Snape, Peggy Gregory, Janet Read, Martin Birch, Christopher Roberts and Tony Nichol for their support and assistance as fellow research students. Norman Lee, Stuart MacFarlane and Simon Monk for reading the draft copies and offering a guiding hand.

Professor Nimal Jayaratna and my supervisory team, thank you for your help, support and enlightening discussions, without your faith in my abilities I would have given up long ago. Professor Trevor Wood-Harper for the guidance and help in the later stages of the research. Many thanks to Peter Clare for being there when I needed additional support and guidance; it was greatly appreciated.

To my husband Alan, thank you for being there, supporting me and helping me through the bad times. David and Gillian, my children, I appreciate your support. To Samantha, a blessing in disguise, now I have time to be a 'proper' grandmother.

Abbreviations

AI	Artificial Intelligence
CAD	Computer Aided Design
CASE	Computer Aided Software Engineering
CLOS	Common LISP Object System
CRC	Classes, responsibilities, collaboration
DFD	Data Flow Diagram
DM	Dynamic Model
DML	Data Manipulation Language
ER	Entity Relationship
ERA	Extended Relational Analysis
FM	Functional Model
IE	Information Engineering
IS	Information Systems
LISP	LISt Processing
NIMSAD	Normative Information Model-based Systems Analysis and Design
OBM	Object-behaviour model
ODMG	Object Database Management Group
OID	Object Identifier
OIM	Object interaction model
OM	Object Model
OMG	Object Management Group
OMT	Object Modelling Technique
ORM	Object-relationship model
OSA	Object system Analysis
OO	Object Oriented
OOA	Object Oriented Analysis
OOD	Object Oriented Design
OODB	Object Oriented Database
OOPL	Object Oriented Programming Language
OOram	Object-Oriented Role Analysis and Modelling
OOSA	Object Oriented Systems Analysis

OOSE	Object Oriented Software Engineering
RDB	Relational Database
RDMS	Relational Database Management System
SSM	Soft Systems Methodology
STD	State Transition Diagrams
UI	User Interface
UML	Unified Modelling Language
WIMP	Windows, Icons, Mice and Pointers

Chapter One: Introduction

This chapter identifies the research question under investigation. The research is set in context and then the importance, the implications and the benefits of carrying out the research are discussed.

1.1 Research Question

Object-oriented methods have developed around object-oriented technology, such as, object-oriented thinking and themes, systems development process from analysis through design to implementation. An object-oriented method of systems development predominantly focuses on software development. A number of texts on system development e.g. Olle et al. (1991) and Rumbaugh et al. (1991) can give inexperienced method users the impression that the solution to many business problems can be achieved by implementing new systems.

Graham (1995) identifies many benefits of the object-oriented approach. Many of these relate specifically to the benefits of object-oriented programming and reuse of code, classes or modules. Those that are relevant to object-oriented analysis include:

- ' well-designed objects in object-oriented systems are the basis for systems to be assembled from large reusable modules.
- The data-centred design approach of object-oriented analysis, because it subsumes process as well, and is able to capture more of the semantics of a model in an implementable form.
- Object-oriented systems are potentially capable of capturing far more of the meaning of the application: its semantics. Since object-oriented methods are mainly concerned with modelling systems they can be used to carry out scenario modelling and facilitate changes within the business. This property leads to greater reversibility in the end product, and enhances the possibility of reverse engineering systems and tracing features back to requirements.'

This clearly indicates that identification and formulation of objects is central to the object-oriented paradigm.

For any object-oriented method the single most crucial aspect of the analysis and design phase is the determination of which objects and classes to include in the model. Object is a very fundamental concept in object-orientation. In the 'action world', an

object could be a real thing that everyone can see such as a car, a book, a bike or a concept that is a relevant to the 'problem area'. For example, the object 'church' could be identified as a physical building where people meet for specific services; it could also be viewed as a conceptual object that includes the religious practices, the people and the notion of community involved in the church activities.

In object-oriented analysis and design, an object models some unity that exists in physical or conceptual space, or some new unity that could be realised in the physical space because someone has thought it out. Users of object-oriented systems development methods describe object systems of their clients' systems; such systems are often said to give a 'logical view' of an object system (Booch 1991). For example, patients and their visits to a clinic, doctors and their work schedules in a clinic can be partially described as objects of a patient administrative system.

Objects are at the core of any object-oriented system and it requires a clear vision of which objects belong in the system to produce a relevant object-oriented analysis model. However there is currently no universally agreed approach to object-oriented analysis, although recently Rational Rose's Universal Modelling Language (UML) (Booch et al. 1996; Mellor and Lang 1997) has claimed to be the de facto standard. This raises the question:

- How do we identify objects?

Inspection of the current literature reveals that the methods for identifying and formulating objects are based on natural language descriptions of the required system. This introduces many inherent ambiguities as it also depends on the writer's linguistic ability rather than the designer's knowledge and experience in a problem domain (Ramachandran and Taylor 1994).

Few texts discuss how the initial identification of entities or objects may be carried out, and some suggest that there is no problem.

> 'The world being modelled is made up of objects... objects are just there for the picking!' (Meyer 1988)

or that

'identifying objects is pretty easy to do. Start out by focussing on the problem at hand and ask yourself 'what are the things in this problem?''(Shlaer and Mellor 1988).

These views are difficult to reconcile with observations of novice and inexperienced data analysts who frequently find great difficulty in identifying a set of 'relevant' entities or objects (Lewis 1994).

Coad and Yourdon (1991) recommend that the analyst identify those 'things' that 'are of relevance and importance to the organisation'. This however only poses further problems. What do we mean by relevant? How do we identify what is and what is not relevant? Who are they relevant to and in what context?

Object-oriented technology brings a new paradigm for developing information systems. Nielson (1990) sees an information system methodology as a coherent set of guidelines. The guidelines prescribe actions both in terms of 'what to do' and 'how to do'. The elements of 'how' are often referred to as techniques that are precise and contain concrete actions to be taken to achieve an objective. Techniques are closely related to tools and models and provide a way of developing systems. An information systems development methodology is an abstraction of structured thinking and actions to be taken in the situation of information systems development. Its application in practice is the concrete action taken.

This raises a related question:

- Do object-oriented analysis methods help us to identify objects?

In order to consider this question, a framework is required within which we can evaluate the methods under consideration. Many frameworks used to evaluate methods, (Olle et al. (1991); Jayaratna (1994) and Avison and Fitzgerald (1995)) have discussed the issues of information systems development and system development methodologies. Nielson (1990) calls these issues 'knowledge about methodology and information systems development'.

These issues led me to question what principles; criteria, models and concepts were being used in object-oriented analysis and design methods. The methods under consideration Coad and Yourdon (1991); Rumbaugh et al. (1991) and Jacobson et al.

(1998) share commonalities rather than differences. They offer a number of procedures (layers of abstraction) and notations. They all provide support for the essential principles of object-orientation – classification, inheritance and encapsulation. However, they all fail to address the critical issues involved in the analysis process: -

- How do we identify objects?
- How do we derive objects from the requirements determination?

NIMSAD[1] (Jayaratna 1994), a formal framework for evaluating methodologies, is applied to evaluate the selected object-oriented methods; Coad and Yourdon (1991), Rumbaugh et al. (1991) and Jacobson et al. (1998) (see Chapter 3). The formal evaluation framework is used to examine object identification from a methodological perspective.

1.2 The importance of this area of research.

Data analysis and modelling are widely assumed to be 'neutral' or value-free. This arises from the claim that data analysis merely uncovers an independently existing, objectively true, structure of the data in a problem domain, recording this structure through some form of conceptual data model. Whether this model is a relational model (Codd 1970), entity-relationship model (Chen 1976) or object-oriented (Shlaer and Mellor (1988), Kim (1990), Coad and Yourdon (1991), Hughes (1991), Rumbaugh et al. (1991) and Shlaer and Mellor (1991) it is usually presented as descriptive of what Cutts (1987) describes as the generic underlying structure of the organisation's data.

Identification of objects serves two purposes: to promote structuring of our understanding of the 'real world' and to provide a practical basis for developing models for implementation for potential users. The decomposition of a problem into objects depends on judgement and the nature of the problem. The lack of attention given to the question of how to identify entities and objects may be explained by the original use of data analysis in the design stages of development, where it is relatively straightforward to identify the 'things' about which data are required to be held. It was

[1] NIMSAD – Normative Information Model-based Systems Analysis and Design Jayaratna, N. (1994). Understanding and Evaluating Methodologies NIMSAD A Systemic Framework, McGraw Hill.
. An introduction to the NIMSAD framework is given in chapter 4

in this context that entities were first represented as generalised descriptions of 'real-world things', and authors such as Martin (1975) explicitly describe an entity as something that exists in the 'real-world'; entity occurrences are merely particular identifiable instances of the same 'real-world' thing.

As the use of data analysis is shifting to earlier stages of development where the problem is less well defined, one would have hoped that one would understand exactly what they meant by terms such as 'entity' and 'object' and to declare whether they are used as ontological labels for parts of the real world, or epistemological devices through which the nature of the 'real world' may be explored. Unfortunately this has not occurred and there has been a continued reliance on pragmatic 'common sense' example led definitions that may be inconsistent (Lewis 1994). In much of the object-oriented literature the concept of 'object' is described, in that objects have identity, state, behaviour and properties and can be characterised by the set of operations that can be performed on it or by it and its possible states.

The 'common sense' of our tradition is that in order to perceive and relate to things, we must have some context in our minds that corresponds to our knowledge of them. We cannot, therefore, answer questions from a neutral or objective standpoint - we subconsciously need to consider tradition - that is, 'way of being'. It is a way of understanding, a background, with which we interpret and act. Tradition emphasises historicity of our ways of thinking, the fact that we always exist with a pre-understanding determined by the history of our interactions with others who share the tradition. People from substantially different traditions have strange and apparently arbitrary 'worldviews', which in a multi-cultural society may lead to some interesting debates on the relevance of particular 'things'.

It is claimed (Coad and Yourdon (1991), Rumbaugh et al. (1991), Embley et al. (1992), Booch (1994), Conger (1994), Elmasri and Navanthe (1994), Graham (1995), Kroenke (1995) and Norman (1996) that objects are a more natural representation of the 'real world', allowing for duplication. However on closer inspection of object-oriented analysis methods, (Shlaer and Mellor (1988), Coad and Yourdon (1991), Rumbaugh et al. (1991), Embley et al. (1992), Booch (1994), Booch et al. (1996), Jacobson et al. (1998)) there is little, in the texts, to differentiate between entities and

objects and very little to help the methodology user to determine appropriate objects. Entities and Objects are variously described as:

Entity: a thing that can be distinctly identified (Chen 1976).

Entity is a 'thing' in the real world with an independent existence (Paton et al. 1996).

An entity is a distinct object (a person, a place or thing, concept or event) in the organisation that is to be represented in the database (Connolly et al. 1995).

An entity is an object that can be distinctly identified and is important to the information system of the organisation (Maciaszek 1990).

Object: a concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand (Rumbaugh et al. 1991).

Object is a uniquely identifiable entity that contains both the attributes that describe the state of a 'real-world' object and the actions that are associated with a 'real-world' object (Connolly et al. 1995).

Objects are the basic unit of construction, be it for conceptualisation, design or programming. These are based, as far as possible, on the real-world entities and concepts of the application or domain (Graham 1995).

In many cases objects are perceived as being the same as the entities identified by traditional analysis and design methods. This raises an immediate question as to the definition of entities and objects.

When the object-oriented literature does offer suggestions, on how to identify objects and entities, they are drawn from the 'craft knowledge' of experienced practitioners rather than being derived from any theory of how human beings make sense of their world or inquire into problem situations. These suggestions include discussions with probable users, investigating the content of current data stores, studying written problem statements and / or requirement specifications (how these have been prepared and by whom is not clear) and intuition and experience. These may be useful but ultimately the data analyst is forced to make subjective choices on what are relevant entities or objects. This is particularly evident where data analysis is being used early in the development; when its role is to investigate exactly what 'things' we need to store information about. This indicates that the identification of object is recognised as a key bottleneck in applying both object-oriented analysis and design.

Traditional structured methods focus on the business functions; data analysis methods focus on the flows of data and the structure of the data files within the organisation context. These methods identify entities using textual analysis: nouns and verbs. The normalisation process removes dependencies and data duplication so that re-creation of an entity can involve joining multiple tables. The person responsible for writing the queries needs to have an understanding of the structure of the underlying tables and the purpose for which the queries are being written. Consequently it can be difficult to write ad hoc queries and obtain the expected results.

The methods evaluated (Coad and Yourdon (1991); Rumbaugh et al. (1991) and Jacobson et al (1998) rely heavily on 'craft knowledge and experience in order to identify and formulate objects'. This is unsatisfactory from a personal and an academic perspective. How is experience gained? How do you know if it is relevant experience? What is meant by craft knowledge? There is very little work on the role of methods in the identification of objects, and an emphasis on 'experience' and 'craft knowledge' when identifying objects with little justification. This is based on a pragmatic approach that does not appear to be well argued. Answers to these questions and related questions arising from the research process are important in order to clarify our understanding and to assist in formulating a realistic set of criteria by which objects can be identified and formulated. If identification and formulation of objects are not considered in this context then we are left with reliance on 'experience', be that of the problem domain or the method and the use of textual analysis, (Abbott 1983), Kelly grids and other techniques offered for identifying entities.

1.3 What else has been done?

Literature relating to identification and formulation of objects is very sparse although sections purporting to cover this are found in textbooks relating to either specific object-oriented methods or to object-orientation in general be that from a software engineering perspective, an analysis and design perspective or a database perspective, for example, (Shlaer and Mellor (1988), Goodman (1989), Khoshafian and Abnous (1990), Booch (1991), Coad and Yourdon (1991), Rumbaugh et al. (1991), Bertino and Martino (1993), Conger (1994), Graham (1995) Jacobson et al. (1998)). How objects are identified depends very much on the authors' perspective and can vary

from using entity and object interchangeably to the implication that identification of objects is not a problem as everyone implicitly knows what objects are.

Lewis (1994) gives a number of quotes from the data modelling and object-oriented literature showing that many methods assume an objective ontology. Lewis shows that there is a potential danger with object-orientation in the insistence that objects are a more natural representation of the real world. When combined with an ontological use of the concept of 'system' this leads to the representation of data analysis as a non-complex, though complicated, process concerned with the creation of models that describe precisely the 'real world'. The commitment to creating data models that reflect the 'real world' is extremely important, as it suggests a particular philosophical location for data analysis as an organised means of rational enquiry, where object modelling is seen as an objective exercise, and classes and associations can be said to exist independently of people. Problems with object models are associated with a failure to capture and specify the real requirements. The solution is to improve the engineering process such that the requirements specification is accurate, complete consistent, and unambiguous. The assumption that there is agreement on means and ends leads to a single object model that is suitable for and acceptable to all stakeholders (Vidgen and Wood 1995).

The emphasis on particular styles of rationalised thought and action means that 'rationalistic tradition' can be distinguished by its narrow focus on certain aspects of rationality. This often leads to attitudes and activities that are not rational when viewed in a broader perspective. Rationalistic orientation pervades much of the computer science, linguistics, management theory and cognitive sciences. A rationalistic orientation to object identification can be depicted in a series of steps:

- Characterise the situation in terms of identifiable objects with well-defined properties.
- Find general rules that apply to situations in terms of those objects and properties.
- Apply the rules logically to the situation of concern, drawing conclusions about what should be done.

It could be argued that many of the developers of object-oriented analysis approaches adopt a rationalistic perspective, as the first step in their approaches is to characterise the situation in terms of identifiable objects.

This view is apparent in the literature written by many developers of object-oriented analysis methods. For example:

> The 'strategy (for identification of objects) comes from practice and experience in the field' (Coad and Yourdon 1991).

> 'Analysts must consider the problem in which they work... first they must understand the problem domain at hand.... If an analyst simply assumes s/he has the subject matter knowledge s/he is likely to indulge in thinking that will lead to fuzzy requirements' (Coad and Yourdon 1991).

As Coad and Yourdon (1991) initially developed structured analysis methods, one assumes that they are relying on this past experience. This would suggest that they use approaches that are familiar to themselves, as authors, without understanding exactly what principles are being applied and why. Similar statements are to be found in other texts, with no explanation to their meaning.

> 'Begin by listing candidate object classes found in the written description of the problem. Write down every class that comes to mind. Classes often correspond to nouns (Rumbaugh 1991).

> ' identifying objects is pretty easy to do. Start by focusing on the problem at hand and ask yourself ' What are the things in this problem''. If your organisation uses database technology, you probably know how to do this already (Shlaer and Mellor 1988).

> 'the world being modelled is made up of objects... objects are just there for the picking!' (Meyer 1988).

An alternative but complementary approach, first suggested by Abbott (1983) is to extract the objects and operations from the textual description of the problem. This relies on the readers having a shared understanding of the problem domain and perceiving the same problems and objects. Objects correspond to nouns and operations to verbs. Verbs and nouns can be further sub-classified. For example, there are proper and improper nouns, and verbs to do with doing, being and having. 'Doing' verbs give rise to operations, 'being' verbs to classification structures and 'having'

verbs to composition structures. Transitive verbs generally correspond to operations, but intransitive ones may refer to exceptions or time-dependent events; in a phrase such as 'the shop closes', for example. This process is a helpful guide but may not be regarded as any sort of formal method. Intuition is still required to develop the best design.

Object-oriented design techniques (such as OMT) organise systems around real world objects or conceptual objects that exist in the user's view of the world (Rumbaugh et al. 1991). Different interpreters will see and talk about different problems. The subjectivist considers reality to be socially constructed and that there are potentially as many realities as there are people involved in the development exercise. Object models are at best agreed inter-subjectively, requiring the participation of many of the parties affected by a proposed Information System. The more people whose interests are reflected in the model then the more likely it is that a shared understanding will be reached with agreement of what classes and associations should be incorporated in the object model and a successful implementation achieved (Vidgen and Wood 1995).

Many of the descriptions of objects include the choice of relevant 'things'. Quinton (1973) suggests that the idea of a 'thing' is general and is derived not from our mastery of the technique of reference to individuals but from the 'thing-predicates', which are the first predicates we learn. The basic elements of discussion are 'thing-indications', statements in the form 'Here is a chair', 'Here is an orange' - even if they are expressed in one word sentences- 'chair', 'orange' are still understandable. We learn to recognise what is meant by the term and can apply it / recognise it in other situations. What these primary objects of attention have in common is that they are all 'things', not stuffs, (i.e. milk, coal, rain,) or qualities (i.e. hot, hard, round).

The reason there can be no substantial idea of a 'thing' in the widest possible sense of the term is that nothing is excluded from its field of application; there are, in the nature of the case, no defining characteristics that can serve to mark it off from anything else. The situation is different in the case of the primary, concrete or perceptible 'things', under consideration. They are in the first place, occupants of space. This means there must be a boundary or limit where the 'thing' finishes and its surroundings begin in physical terms. This implies that the 'thing' will have both a

shape and a size. The characterisation of primary 'things' in general is not very exact but this is because our conception of such 'things' is not very exact either.

Rationalistic tradition regards language as a system of symbols that are composed into patterns that stand for things or explanations in the world. Sentences can represent the world truly or falsely, coherently or incoherently, but their ultimate grounding is in their correspondence with the state of affairs they represent. The content words of a sentence (nouns, verbs and adjectives) can be taken as denoting (in the object-oriented domain world) objects (nouns), properties, relationships (verbs), or sets of these. A body of work has been produced that systematically examines meaning from a formal analytical perspective:

- Semantic correspondence recognises deep ontological problems in deciding just what constitutes a distant object or in what sense a relation or event exists.
- To study meaning is to take for granted that some kind of correspondence exists, without making a commitment to its ontological grounding.

There is an on-going debate between those who place meaning within the text and those who see meaning as grounded in a process of understanding in which the text, its production and its interpretation all play a vital part. The goal of hermeneutics theory (Checkland (1981), Checkland and Scholes (1990) and Checkland and Howell (1998) (theory of interpretation) is to develop methods by which we rid ourselves of all prejudices and produce an objective analysis of what is really there.

Gadamer (1975) takes an opposing approach that takes the act of interpretation as primary, and understanding as an interaction between the horizon provided by the text and the horizon that the interpreter brings to it. Any individual, in understanding his or her own world, is continually involved in activities of interpretation. Interpretation is based on prejudice (that which is accepted without question) that includes the assumptions implicit in the language that the person uses. That language in turn is learned through activities of interpretation. The individual is changed through the use of language, and the language changes through its use by individuals. Prejudice implies that all reasons for a conclusion are an accepted fact and are therefore not open to further reasoning.

Object-oriented analysis methods imply that the analysts can step into any organisation and undertake an analysis exercise. Shlaer and Mellor (1988), Coad and Yourdon (1991) and Rumbaugh et al. (1991) state that reading documents and talking to the client and users will supply the necessary background knowledge and enable the analyst to understand the problem domain. This implies that the analyst is already familiar with the problem domain. This can lead to problems whereby the analyst makes assumptions as to what is required without talking to the users. Having gained an understanding of the problem domain does not mean that the analyst shares the same understanding of the terminology used.

Information analysis is concerned initially with understanding the business itself and business needs. It then considers how the needs can be met by providing an organisational and technical system. Business needs are rarely considered by object-oriented analysis methodologies. There is an assumption (Jacobson et al. 1998) that an in-depth business analysis has been performed prior to the object-oriented analysis. Coad and Yourdon (1991) and Rumbaugh et al. (1991) concentrate on the perceived problem area and do not consider the organisational requirements. Indeed Rumbaugh et al. (1991) state that 'psychological, organisational and political considerations for doing this (analysis) are deemed to be beyond the consideration of the methodology'.

There are indications that the literature concerning object-oriented analysis is inheriting some of the lack of conceptual clarity of its predecessors. Object-oriented analysis methodologies have been compared by a number of authors (de Champeaux and Faure (1992), Fichman and Kemerer (1992) and Iivari (1994)), they concentrate mainly on the notation used and the use of inheritance and other object-oriented features, none of them consider how objects are identified or determined. Brown (1989), Bouzeghoub and Metais (1991) and Kroenke (1995) propose semantic object models, which in some cases are seen as alternatives to object-oriented models, in others as an intermediate step between traditional data models and object-oriented models

Object-oriented analysis methods use some of the concepts of classification and categorisation, although there is rarely any reference to the philosophical underpinnings. Shlaer and Mellor (1988) suggest five categories: tangible entities,

roles, incidents, interactions and specifications. Coad and Yourdon (1991) argue that the techniques and notation of OOA are based on the 'methods of organisation that pervade all human thinking'. They claim that analysts should look for things or remembered events, devices, roles, sites and organisational units, and ways of classifying objects. They rely on classification theory to validate and justify their proposals that people think in terms of objects and that therefore these 'thinking tools' are useful in determination of requirements.

An analyst familiar with an application may formulate objects in many different ways and choosing the 'best' form of representation is to understand the 'real world' phenomena in object-oriented terms. Some conceptual objects that are formulated in the mind may correspond directly to 'real world' objects, such as employees or chairs but others may correspond to invented categories. The objects that are considered to be relevant may depend on the application, but this may compromise the reusability of the object. The more high level our systems become, the more they are regarded as tools for enhancing communication between humans. The more the human aspect is emphasised the more the subjective factor in object identification comes to the fore.

More recently the study of Semiotics (Lui et al. 1998; Lui 1999) considers the identification of objects from semantic, syntactic and pragmatic perspectives and as such could be usefully integrated with object-oriented analysis. From a semiotics perspective a proper information analysis exercise can only be done if the people involved use the same language and have a substantial common understanding of the problem domain. In semiotics the world to be modelled is constructed by the community of agents, for example, problem owners. The agents know the meanings of words in their own world, their interpretations are the only ones justified. It is not allowed, within semiotics, for an analyst to invent artificial terms or introduce new concepts when modelling the agent's actions. The purpose of this is to force the analyst to speak the same language as the problem owners. Any ambiguity in the terms or concepts used in describing the problem should be resolved by putting them into a context of actions, which are already described and understood. When doing so, if the problem-owners are inspired with some new terms, the problem owners and the analyst may use them only after careful consideration.

1.4 Research Method

My study will be of an exploratory nature and will consist of a critical evaluation of selected object-oriented analysis methods, using a selected framework, and concentrating on how the methodologies help the analyst to determine appropriate objects. A pilot study will be conducted using interviews and working through a given scenario by a small number of academics and practitioners in order to determine what methods are used in practice to identify objects.

In evaluating three object-oriented methodologies, my research follows an 'interpretivist' approach (Galliers 1991) using Action Research and consideration of hermeneutics. The object-oriented methodologies will be evaluated using NIMSAD (Jayaratna 1994) framework. Chapter 2 discusses the research methods chosen and justifies the stated choice.

1.5 Benefits of the research

This thesis critically evaluates a number of well-known methods and a small number of industry practices, this underpins future research in this area which could lead to the development of a set of criteria to aid object identification and formulation, it also benefits others researchers in the field.

Although object-oriented analysis and design methodologies have been evaluated previously (de Champeaux and Faure (1992), Fichman and Kemerer (1992) and McGinnes (1993)), the comparisons are based on differences in notation or technical content. They do not consider how the methodologies identify objects. In this respect this thesis contributes to knowledge in this area.

The final benefit of undertaking this research is in improved teaching. The research area feeds directly into the teaching situation and students are therefore being asked to critically evaluate their own methods of identifying objects and to challenge what they read.

1.6 Structure of the Thesis

The main emphasis of this thesis is how objects are identified in object-oriented analysis methodologies and why this is important from methodological and practical perspectives. Three object-oriented analysis and design methodologies are evaluated

using the NIMSAD (Jayaratna 1994) framework; the similarities and differences between the methods are then compared.

The thesis consists of three parts and nine chapters. Part 1 comprises four chapters. Chapter 1 addresses the research issue and Chapter 2 research methods. Chapter 3 introduces object-oriented concepts and terminology in order to ensure a common understanding for the reader. This chapter also includes an overview of current object-oriented analysis and design methods and justifies the selection of Coad and Yourdon (1991), Rumbaugh et al. (1991) and Jacobson et al. (1998) for evaluation. Chapter 4 discusses evaluation frameworks and justifies the selection of NIMSAD (Jayaratna 1994).

Part 2 comprises four chapters. Chapters 5 –7 respectively evaluate the selected object-oriented analysis and design methodologies using NIMSAD as an evaluation framework. Chapter 8 evaluates the methods used by practitioners, both in industry and academia, to identify and formulate objects.

Part 3 comprises one chapter. Chapter 9 identifies the major findings of the research and then expands on these conclusions from

- a methodological perspective,
- the perspective of the evaluated methodologies
- the use of the NIMSAD framework.

1.7 Summary

This chapter identified the research question under investigation, and the importance of the research issues. The research area was set in context and the implications and benefits of undertaking the research were identified. The structure of the thesis was also outlined.

In chapter two I will consider research methods that are relevant for Information Systems research and justify my choice of action research.

Chapter Two: Research Methods

2.1Introduction

The field of Information Systems is seen as multi-disciplinary and as such choosing an appropriate research approach in Information Systems has been the focus of much debate (McFarlan 1984; Mumford et al. 1984; Nissen et al. 1991). These conferences highlighted the existence of 'radically different' approaches within the Information Systems research community. By 'radically different', Klein et al. (1991) means research approaches building on assumptions that are in some way contradictory to each other. For example, three of the most widely used information systems research methods, laboratory experimentation, surveys and case studies, are identified as having a scientific approach, while action research, reviews and descriptive / interpretive research are categorised as interpretivist. However there is much debate as to whether case studies should be categorised as an interpretivist research approach, especially given the particular 'appreciative system' (Vickers 1990) or 'cognitive filter' (Simon 1978) of the researcher.

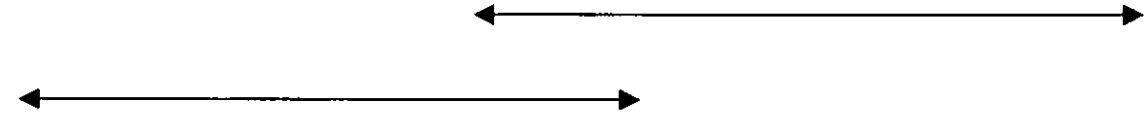
Research into information systems can be conducted in a number of ways. The structure of this chapter is as follows: a brief discussion on typologies of research methods using some ideas from the two colloquia followed by a brief look at each research method; the choice of action research for this study and its implications; and finally a short summary

2.2 Research Methods

According to Galliers (1991) a range of research approaches have been identified for use in the general field of Information Systems. These have been developed into a framework (Galliers 1995), to aid choice of information system research approach (Fig 2.1).

The scientific (objectivist) approach is characterised by realist ontology and a positivist epistemology. Ontology is concerned with the nature of reality. Epistemology is concerned with how we gain legitimate knowledge about the world. Loosely speaking, the objectivist assumes that objects and structures exist

independently of human observers and that the appropriate way of gaining knowledge is by observation and the identification of causal relationships. Scientific approaches are those that have arisen from the scientific tradition, characterised by repeatability, reductionism and refutability (Checkland 1981) and which assume that observations of the phenomena under investigation can be made objectively and rigorously (Klein and Lyytinen 1985). The subjectivist or interpretivist approach holds that scientific method is not appropriate for explaining the social world as different people interpret the world in different ways and any agreement is inter-subjective (Laing 1967; Checkland 1981; Vidgen and Wood 1995).



Scientific Approaches
Modes for traditional, positivist approaches
(based on an observation paradigm)

Interpretevist Approaches
Modes for post-positivist approaches
(based on an interpretative paradigm)

Object	Theorem Proof	Lab Exp	Field Exp	Survey	Case Study	Futures Research	Simulation	Reviews	Action Research
Society	*	*	**	***	**	***	**	***	**
Organisation	*	**	***	***	***	***	***	***	***
Individual	*	***	***	**	**	**	***	***	**
Technology	***	***	***	**	*	***	***	**	*
Methodology	***	*	***	***	***	*	***	***	***

Key: likely relevance: *** potentially good; ** possible; * less likely

Adapted from (Galliers 1995)

Figure 2.1 A Framework to aid choice in information systems research approaches

Interpretivist approaches argue that the scientific ethos is misplaced in social scientific enquiry because of:

'- the possibility of many different interpretations of social phenomena,
- the impact of the social scientist on the social system being studied,
- the problems associated with forecasting future events concerned with human...activity' (Checkland 1981; quoted in Galliers 1991).

The explanation and understanding of the occurrences have therefore to consider the subjectivity of the actors and the potential impact of the researcher upon the situation.

The following sections consider each method, giving a fuller definition and a brief discussion:

Conceptual study

Laboratory experiments

Field experiments

Surveys

Case Studies

Phenomenological Research / Hermeneutics

Action research

2.2.1 Conceptual Study

This method is to the extreme right of the interpretivist approach (for example, more idealistic) and is sometimes referred to as 'armchair research'. The implication is that no actual on-site experimentation is carried out. The development of frameworks for research and representative models may be seen as falling into this category. A related approach is that labelled as philosophical research (Jenkins 1985). The basis for the research may be strictly that of opinion and speculation.

A conceptual study often involves the logical reasoning of causal relations using a standard set of rules. This ensures that anyone may repeat the process and obtain the same results. These methods require only the thinking process and no active experimentation. There is therefore no measurement or observation. The testing of a hypothesis is therefore also irrelevant. The major drawback with this approach is the necessity to link the conceptual environment with a 'real environment'. That type of mapping requires the application of one of the other methodologies in the Information Systems field.

2.2.2 Laboratory experiments

The key features of the laboratory experiments approach is the identification of the precise relationships between variables in a designed, controlled, environment (for example, a laboratory) using quantitative analytical techniques. This is done with a view to making general statements that are applicable to real world situations. The major strength of the approach lies in the ability of the researcher to isolate and control a small number of variables, which may then be studied intensively. The major weakness of the approach is the limited extent to which identified relationships exist

in the real world due to the over-simplification of the experimental situation and the isolation of such situations from most of the variables that exist in the real world (Galliers 1991).

2.2.3 Field experiments

Field experiments are an extension of laboratory experiments into the 'real world' of organisations/society and include field studies, field tests, adaptive experiments or group feedback analysis (Dickson et al. (1977), Jenkins (1985)). The idea is to attempt to construct an experiment in a more realistic environment than is possible in the artificial laboratory situation. Strengths and weaknesses are the same as for laboratory experiments but an additional weakness is the difficulty of finding organisations prepared to be experimented on. In addition replication is problematic, in that it is extremely difficult to achieve sufficient control to enable replication of the experiment with only the study variable being altered.

2.2.4 Surveys

Survey research is characterised by Bryman (1989) as entailing

> 'the collection of data on a number of units and usually as a single junction in time, with the view of collecting systematically a body of quantifiable data with respect to a number of variables, which are then examined to discern patterns of association'.

The systematic collection of data in quantifiable format, primarily through the use of structured questionnaires and interviews, therefore forms the bedrock of the survey research method. A sample, which is representative of the target population, is collected in respect of a number of predefined variables at a particular point in time. This pool of data is subsequently aggregated and examined, using various rules, to discern patterns of association between the variables. These patterns are compared against the hypothesis investigated to determine causality. On the basis that the sample is representative of the target population, statistical generalisation of the findings can be deduced. Statistical sampling error techniques are employed to establish the scientific nature of the findings.

In surveys, involvement on the part of the researcher is limited to composition of questionnaires and a passive collection of the data from the selected group of

respondents. The strength of this research approach is that a large pool of quantitative data can be obtained on a large number of issues under investigation. Analysis of data leading to the conception, and inference, of knowledge is often dependent on the satisfaction of various pre-requisites of the different research approach. One of the pre-requisites involves the degree of subjectivity that affects the findings. In survey research the subjective quality of the respondents and the researcher are often not recognised and considered in the findings. This deficiency can lead to possible misrepresentations and generalisations of knowledge. Similarly, analysis and interpretation of the data is open to the subjective bias of the researcher through the nature of the data collection process.

As object-orientation is increasingly been seen as the preferred development approach within information systems, a survey of participants within large organisations that develop information systems could contribute to the broad evaluation of the use of object-oriented analysis methods within the industry. However, the survey method would not contribute to the resolution of the problem and difficulties in gaining access that may have been encountered (Whyte (1984) and Buchanan et al. (1988)) and acceptance of the participants due to the feeling of suspicion about the impact of further research. Furthermore the results of a survey would be subject to the personal interpretations and speculation of the respondents.

2.2.5 Case Studies

Case study research essentially involves the detailed investigation of real world phenomena. It considers events that take place and relationships that exist, in particular within a single organisation or a number of organisations. These observations provide insights that contribute to the building of theoretical knowledge.

In case study research, detailed examination of one or more cases of similar experiences provides the background for the elucidation of findings and enhancement of theoretical notions. This is often carried out longitudinally, enabling the observation of events unfolding over a period of time within particular organisational settings (Pettigrew 1989). Analysis of the findings, which may take both quantitative and qualitative forms, leads to increased knowledge of the 'action world'. Case study research can also be used to explore new areas of knowledge and to test theoretical

ideas (Benbasat et al. 1987). Benbasat et al. (1987) identify four criteria to assess the appropriateness of the case study method. The first is the question of whether the area of interest can be divorced from its natural setting. The second is whether or not the interest lays in current events. The third is the question of the need for control or manipulation of variables. The fourth is the existence of an established theoretical base.

Identification and control of independent variables is weak in the use of case studies and interpretations are 'highly subjective'. The area of research may not lend itself to a highly quantified and controlled situation. The study of a unique situation, occurring only once, would indicate a case study approach.

Yin (1989) points out that case studies do not necessarily imply a particular type of evidence or particular mode of data collection. The terms qualitative and case study are not necessarily synonymous, though often case studies are used to derive qualitative as opposed to quantitative data. Case studies might utilise questionnaires, interviews, archival records, documentation, physical artefacts or direct observation as their source of evidence (Benbasat et al. 1987).

The involvement of the researcher in a case study is generally passive observation and recording perceptions of the events and interactions that take place. The researcher adopts the stance of an insider, using flexible approaches to gather data over a period of time, but plays no part in the events and interactions. Detailed capturing of data relating to the 'action world' situation is the strength of the approach. However, the restricted scope of the case study research is considered to be a constraint on its capacity for generalisation of knowledge. Focused on the interpretations of findings within the context of the research environment case study research, which often employs covert means of data collection, is open to the possibility of the Hawthorne effect (Roethlislerger and Dickson 1939) upon the participants. This possibility, together with the personal attributes of the researchers which influence their perception of the situation are not overtly considered in the interpretation of the findings leading to distortion of observations.

The previous discussion has described the case study approach, when its use might be appropriate and the various types available. Advantages and disadvantages of the various approaches have emerged throughout this discussion. These can now be summarised:

Benbasat et al. (1987) identify three reasons why the case study is appropriate for information systems research.

1. The study can be conducted in a natural setting, learning about the state of the art and generating theories for practice.
2. The nature and complexity of the process can be studied.
3. Situations that are new and rapidly changing can be studied using the case study approach.

The case study lies on the subjective end of the Galliers (1995) framework. Positioning case studies as a subjective approach to research constitutes the main source of criticism in the United States. It has been argued that the case study is relatively high in cost, and control is minimal or nonexistent. Offsetting these disadvantages, the strength of the case study is in its use for study in the natural situation, including contextual influences, and for provision of deep and comprehensive analysis, which is unavailable through the application of other methods (Jenkins 1985).

2.2.6 Phenomenology / Hermeneutics

Boland and Hirschheim (1987) define phenomenology as:

> 'The methodological study of the consciousness in order to understand the essence of experience... phenomenology is concerned with the structure of meaning that gives sense and significance to our immediate experience.'

In other words, the concern is not in finding out how things work but rather in what things are (Galliers 1991). Boland and Hirschheim (1987) point out that the problem is basically a hermeneutic problem, that is, one involving the interpretation of situations. The impetus of this mode of research is that we cannot really understand communication itself unless we are aware of the context in which that communication was generated.

Hermeneutics has its origins in the interpretation of religious texts. Two schools of thought exist within hermeneutics; the objectivist school, which wishes to 'decontextualise' the text, thereby producing an objective analysis of what is really there. This is described in Winograd and Flores (1986). The other school views interpretation as an essential key allowing us to map from the prejudices of the communicator to our current set of prejudices.

Heidegger (1962 in (Winograd and Flores 1986)) refutes the idea of the objective world, which needs no interpretation and the subjective world comprising our thoughts and feelings only. He argues that the two are not independent, but rather one does not exist without the other. No objective view of reality exists: it is flavoured by our subjective perceptions.

Hermeneutics, Palmer (1969) suggests is a way of understanding textual data. It is primarily concerned with the meaning of a text; that is understanding the text as a whole and the interpretation of its parts, in which descriptions are guided by anticipated explanations (Gadamer 1976). It follows from this that we have an expectation of meaning from the context of what has gone before.

The hermeneutic world-view recognises multiple interpretations of events, based on the different understandings, motives and reasoning of unique individuals (Webb 1990). Each individual has their own construction of the world and creates their own reality and their own learning pathway through it (Zuber-Skerritt 1996).

The discussion given to describe this research method would tend to make 'objective' comparisons with other methodologies difficult. This method may be seen as close to the subjective end of the spectrum. Its application may not be so much as an independent method but an approach to use in conjunction with other methods, for example, action research or case study. Phenomenology / Hermeneutics would not be appropriate where one wishes to use statistical inference as the guarantor of truth.

There is a plethora of information relating to object-orientation per se but there is little specifically written about object identification. What is available depends on the perspective of the author be that a software engineering perspective, a database

perspective or a systems analysis perspective. Phenomenology / hermeneutics can be focused on the literature or past developments, in addition to actual, current, happenings. This approach is frequently adopted in addition to other approaches in order to establish the viability of the research question and to limit the scope of literature search. Critical evaluation of selected texts can lead to significant advances in knowledge, and the ability to develop theory can be made through in-depth review of this kind in a particular aspect of subject matter (Galliers 1991).

A phenomenological research approach was taken in order to discover how 'experts' perform a complex intellectual task; this is described in chapter 8. This was an exploratory investigation, starting with a minimum of preconceptions and mapping the main issues that the participants raised.

2.2.7 Action Research

Action Research is sometimes treated as a subset of the case study approach within the Information Systems research community in North America (Benbasat et al. 1987). However, it would fail the selection as a research method utilising the 'positivistic or natural science' criterion. In other words, this criterion would conform to the assumption that the empirical world is value free, logical and the ultimate reality (Morgan 1983). The justification of action research is based on the acceptance of the subjective processes and the possibility that knowledge is socially constructed.

Action research is one of the research strands in social science. The others are pure basic, which is concerned with a theoretical problem; basic objective, which takes a general practical problem; evaluation, which assesses some aspect of performance and applied, which tries to solve a problem by applying the appropriate knowledge (Clark 1972).

Action research is a method that could be described as a paragon of the post-positivist research methods. It is empirical, yet interpretive. It is experimental, yet multi-variate. It is observational, yet interventionist (Baskerville and Wood-Harper 1996).

Warmington (1980), who attributes the first explicit use of the method applied to social problems to the social psychologist Kurt Lewin in the 1940's, has traced the

history of the development of action research. In Warmington's account, the implicit use of the method by researchers such as F.W. Taylor and E. Mayo in solving practical organisation problems predates Lewin's work. Lewin's motivation and contribution was that the researcher ought to be 'useful' as well as being an observer and that theory needs to be accumulated. Other versions of the method were used in the 1960's by the Tavistock Institute and by Lancaster University. At Lancaster Action Research was used extensively in Soft Systems Methodology (Checkland 1981; Checkland and Scholes 1990; Checkland and Howell 1998) which develops the concept of human activity systems within a system engineering methodology.

These variants will not be discussed further but the essence from the different interpretations can be captured in the following quote from Hult and Lennung (1980):

> 'Action research simultaneously assists in practical problem solving and expands scientific knowledge, as well as enhances the competencies of the respective actors, being performed collaboratively in an immediate situation, using data feedback in a cyclical process aiming at an increased understanding of a given social situation, primarily applicable for the understanding of change processes in social systems and undertaken within a mutually acceptable ethical framework'

Action research therefore attempts to link theory and practice, thinking and doing, achieving both practical and research objectives (Susman 1983). Gaining this knowledge is seen as an active process, so that beliefs may be re-defined in light of the outcomes. A representation of reality is not so much desired, but rather is a means of dealing with reality. Action research is a pragmatic approach which desires to 'come to terms' with the world. This emphasis on the pragmatic is further reflected by Susman (1983) who states that action research builds on a learning circle. His learning circle is diagnosis, action planning, action taking, evaluation and the specification of learning (Susman and Evered 1978). The key point to notice is that the researcher/ actors/ participants/ subjects begin in a real concrete situation and return to it at the end of the leaning cycle.

This type of learning represents the enhanced understanding of a complex problem. Information about a particular situation and a particular environment has been

obtained, which gives a contingent value to the truth learned. The researcher expects, however, to generate knowledge that will further enhance the development of models and theories. The aim is the understanding of the complex human process rather than a universal prescriptive truth.

Finally, in the process of learning, an explicit clear conceptual framework must exist which the researcher imposes on the situation. The conceptual framework must be acceptable to both the researcher and the organisational actors in the action research study (Warmington 1980). This is needed so that the explicit lessons will emerge from the research cycle.

In comparing action research with the other methods, three important distinctions should be brought out.

1. The researcher is actively involved, with expected benefits for both researcher and organisation.
2. Knowledge obtained can be immediately applied. There is not the sense of the detached observer, but that of an active participant wishing to utilise any lessons learned based on an explicit clear conceptual framework.
3. The research is a cyclical process linking theory and practice.

2.3 Choice of action research and the implications.

The discipline of information systems seems to be a very appropriate field for the use of action research methods. Information Systems is a highly applied field, almost vocational in nature (Banville and Landry 1989). Action research methods are cyclical in nature and place Information Systems researchers in a 'helping hand' role within the organisations that are being studied. Action research merges research with praxis thus producing exceedingly relevant research findings (Baskerville and Wood-Harper 1996). Such relevance is an important measure of the significance of Information Systems research.

The above discussion can be seen within Checkland's intellectual context: the 'organised use of rational thought (Checkland and Scholes 1990). This context is

based on a simple model of a cycle of continuous inquiry where theory interacts with practice (Fig 2.2)

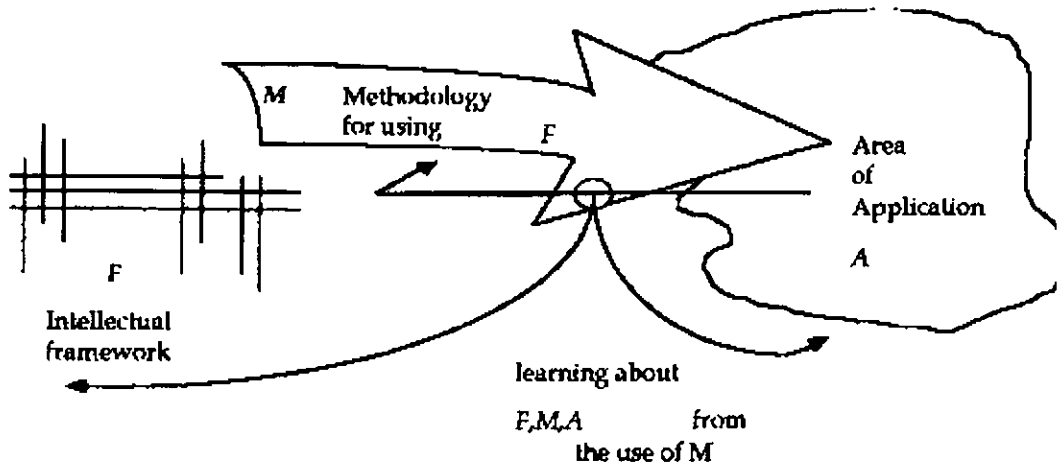


Figure 2.2 Organised use of rational thought

(adapted from Checkland 1981)

Depicted in this diagram is the use of a theory, an intellectual framework of linked ideas F; through a methodology or an intervention process M; to an application area A. In going through a cycle, learning can be generated about F, M and A.

Within this thesis the area of application is the identification of objects in object-oriented analysis methods. There are a plethora of object-oriented analysis methods available, many of which would have us believe that everyone knows what an object is or identifying objects comes with practice. In order to evaluate a selection of object-oriented analysis methods a problem-solving framework is required, NIMSAD framework was selected as an appropriate intellectual problem-solving framework (chapter 4). The question now is which research method is appropriate to learn about the theory, the methodology and the identification of objects in object-oriented analysis?

The need for this learning implies that the practical implications of the three aspects should be explored. To do this the methodology ought to be used as an exploratory tool to search patterns in the practical world. Secondly, the assumptions, practical and otherwise, which are implicit in using the methodology, ought to be identified.

Finally, the conditions in which the methodology is useful and non-useful should be clarified (Weick 1984).

In helping us in the choice of a research method, Galliers (1991) has identified six general application areas in Information Systems: society, organisation, small group, individuals, technology and methodology (Fig 2.1 pg 26). He suggests that from these research methods, mathematical modelling and laboratory experiments are inappropriate for research into methodologies. The case study approach was rejected as it is difficult to negotiate access to suitable organisations and as case studies are longitudinal by nature it was considered too time consuming. The non-involvement of the author in case study research, and the passive collection of quantitative survey data, would not have been appropriate and would have encountered similar problems to those associated with surveys. By reviewing the remaining methods, in order to choose an appropriate one, elements of hermeneutics are appropriate but there is a need to combine this with action research in order to fulfil the requirements for learning in practice mentioned above.

Land (1982) succinctly captures the research requirements, the choice in this study and the implications in using action research:

> ' If you are going to do research in this area, which method of research is valid? We're concerned with a multi-dimensional world in which it is very difficult to analyse cause and effect...some people call it action research... develop techniques, methods and methodologies and thus we go into the world and try them out... find a host who is willing to use these and we try to see as best as possible how things will work out.... It is difficult to say what effect is due to the methodology or to some personal or environmental factors....'

The implications can also be seen in comparison with other methods. As described above the methods should be viewed as a continuum and the comparison of the extremes, Laboratory Experiments (Positivist Science) to Action Research is shown in Table 2.1

Points of Comparison	Positivist Science	Action Research
Value position	Methods are value neutral	Methods develop social systems and release human potential
Time perspective	Observation of the present	Observation of the present plus interpretation of the present from knowledge of the past, conceptualisation of more desirable futures.
Relationship with units	Detached spectator, client system members are objects of study	Client system members are self reflective subjects with whom to collaborate
Treatment of units studied	Cases are of interest only as representatives of populations	Cases can be sufficient sources of knowledge
Language for describing units	Denotative, observational	Connotative, metaphorical
Basis for assuming existence of units	Exists independently of human beings	Human artefacts for human purposes
Epistemological aims	Prediction of events from propositions arranged hierarchically	Development of guides for taking actions that produce desired outcomes
Strategy for growth of knowledge	Induction and deduction	Conjecturing, creating settings for learning and modelling of behaviour
Criteria for confirmation	Logical consistency, prediction and control	Evaluating whether actions produce intended consequences
Basis for generalisation	Broad, universal and free of content	Narrow, situational and bound by context

From Susman and Evered 1978

Table 2.1 A Comparison of Positivist Science and Action Research

The major consequence of the choice of Action Research method is that the research is context-bound as opposed to context-free. This means that this produces narrow learning in its context: understanding object-oriented methods, the NIMSAD framework and the resulting outcomes.

2.4 Summary

The above discussion considered a number of research methods that have been used within Information systems research. The selection of Action Research as an appropriate research method was discussed and justified.

Chapter three considers object-oriented terminology, the development of object-oriented systems methods, and an overview of current object-oriented analysis and design methodologies.

Chapter Three: Object Orientation and Object-Oriented Systems Development Methods

The term object-oriented (OO or O-O) has its origins in OO programming languages. OO concepts are now being applied to many different types of computer based design software i.e. databases, software engineering, knowledge bases and artificial intelligence systems. The principles of object technology can be applied across all aspects of software development. Object technology can be used to construct many types of software, and many types of software can exhibit characteristics of object technology.

This chapter gives an overview of the terminology used in this thesis. The development of object-oriented methods is then considered followed by object-orientation concepts and terminology. Justification for selecting Coad and Yourdon (1991), Rumbaugh et al. (1991) and Jacobson et al. (1998) methodologies to be further evaluated using NIMSAD framework (Jayaratna 1994) follows an overview of the development of object-oriented analysis methods.

3.1 Terminology

This section provides explanations of some of the terms used in the body of this report that may be open to interpretation. Specific object-oriented terminology is considered in section 3.3.

Real World: is taken to consist of both the 'thinking world' and the 'action world' of the intended problem solver (see 'thinking world' and 'action world').

Action World: is the situation in which methodologies are used for bringing about transformations. Many of the scenarios described in object-oriented analysis texts relate to the 'action world'; that is the physical reality of the situation (see 'thinking world' and 'real world').

Thinking World: relates to the conceptual world and the way in which the methodology user thinks about the situation; that is the different

ways in which 'reality' can be conceptualised (see 'real world' and action world').

Paradigm: a way of thinking, object-oriented paradigm means thinking in terms of objects.

3.1.1 Methodology and Method

'Methodology' is a Greek term meaning the 'study of methods'. Within the information systems field the term 'methodology' has come to mean the same as 'method'. Stamper (1988) expresses dissatisfaction with the use of the term, methodologies, in this way. He argues that 'methods 'should be used when referring to specific ways of approaching and solving problems and 'methodologies' should be reserved for comparative and critical study of methods in general.

At a general level, a methodology is considered to be a recommended series of steps and procedures to be followed in the course of developing an information system (Avison and Fitzgerald 1995). The British Computer Society (BCS) Information Systems Analysis and Design Working Group provided one of the most useful definitions in 1983. They defined an information system methodology as a

> 'recommended collection of philosophies, phases, procedures, rules, techniques, tools, documentation, management and training for developers of information systems' (Maddison 1983).

Checkland (1981) has distinguished between the terms 'method' and 'methodology' and says that a methodology

> 'is a set of principles of method, which in any particular situation has to be reduced to a method uniquely suited to that situation'.

According to Graham (1995) a method consists of at least: a modelling technique, a process and a notation. The notation could just be natural language. A method should minimise the amount of notation that must be learnt, minimise the number of mandatory deliverables from the process and minimise the number of foundation concepts in the modelling approach (Graham 1995).

Jayaratna (1994) defines a methodology simply as,

> 'an explicit way of structuring one's thinking and actions. Methodologies contain model(s) and reflect particular perspectives of 'reality' based on a set of philosophical paradigms. A methodology should tell you 'what' steps to take and 'how' to perform those steps, but most importantly the reasons 'why' those steps should be taken in that particular order' (Jayaratna 1994).

This differs from the definition of methodologies given by other authors, for example, Avison and Wood-Harper (1990), Avison and Fitzgerald (1995) and Olle et al. (1991), which concentrate on the 'what' and 'how'. Avison and Fitzgerald (1995) define an information systems development methodology as:

> 'a collection of procedures, techniques, tools, and documentation aids which will help the systems developers in their efforts to implement a new information system. A methodology will consist of phases, themselves consisting of sub-phases, which will guide the systems developers in their choice of techniques that might be appropriate at each stage of the project and also help them plan, manage, control and evaluate information systems projects.'

This definition clearly concentrates on techniques and tools of the methodology - the 'what', and the phases and sub phases of the methodology - the 'how'. 'Why' a particular methodology should be chosen is not considered. Arguably this implies that the problem situation is 'taken as given' and a potential solution has already been identified at the start of the project.

In this thesis I will follow the definition as given by Jayaratna (1994); that is methodology refers to a specific way of approaching and solving problems.

3.1.2 Method and Operation

'Method' in object-oriented terms indicates the behaviours to be carried out by an object class. There are usually two parts to a class structure: data structure (or attributes) and methods (operations in Rumbaugh et al. (1991), and Jacobson et al. (1998) or functions in Coad and Yourdon (1991)). A method is a function applied to or by an object class, for example, dancing and jumping are methods of an object class person. Such a usage of method is quite different from a 'method' in systems

development. To avoid confusion, 'method' in an object class structure in object-oriented contexts will be replaced by 'operation', instead of other expressions mentioned above.

3.2 Development of object-oriented methods

Early object-oriented techniques were developed in Norway during 1960's with the development of Simula (Nygaard and Dahl 1981). The term 'object-oriented' was first used with the advent of Smalltalk (Goldberg and Robson 1983) during the 1970's. During the 1980's there was great interest shown in the User Interface (UI). Object-oriented programming (OOP) supported the development of such interfaces, while the style of object-oriented languages was heavily influenced by the Windows, Icons, Mice and Pointers (WIMP) paradigm. This had the effect of creating vast libraries of objects for interface development, compared to other areas.

From the mid 1970's there was also considerable cross-fertilisation between object-oriented programming and artificial intelligence (AI) research and development, leading to extensions of AI languages, notably List Processing (LISP) and Common LISP Object System (CLOS). These systems were influenced by ideas on knowledge representation based around semantic networks and frames. These representations express knowledge about 'action-world' objects and concepts in the form of networks of stereotypes, objects that can inherit features from more general ones.

As OOP matured, interest shifted, initially, to object-oriented design methods and more recently to object-oriented analysis or specification. Important questions arise in this area; such as whether an object-oriented design must be implemented in an object-oriented language or whether current design methods are in fact tied to specific languages. The current phase in the history of object-orientation is characterised by the shift in emphasis from programming to analysis and design and by an awareness of the issues of open systems and standards.

3.3 Object-Orientation

An object-oriented approach is, according to many texts, for example, Coad and Yourdon (991), Rumbaugh et al. (1991) and Graham (1995) suitable for the whole development life cycle and moves much of the system development effort to the

analysis phase of the life cycle. As an object-oriented approach defines a set of problem oriented objects early in the project and continues to use and extend these objects throughout the development cycle, the separation of development life-cycle phases is much less distinct. The seamlessness of object-oriented development makes it easier to repeat the development steps at progressively finer levels of detail. Each iteration adds or clarifies features that have already been identified, so there is less chance of introducing inconsistencies or errors.

As Rumbaugh et al. (1991), state

> ' The object-oriented approach fosters a pragmatic approach to problem-solving drawing upon the intuitive sense that object-oriented technology captures and by providing a notation and methodology for using it systematically on real problems'.

An object-oriented approach takes objects from the application domain, in the first place, which are then used to understand the problem domain; operations are then fitted around the objects.

As the terminology used in relation to object-orientation has not settled on one consistent set of definitions, referring to the definitions given by a number of authors could lead to conceptual confusion. The following terms will, therefore, be used in the discussions of object-orientation.

Object:	the basic unit of construction for conceptualisation, design or programming. Objects are based on 'action world' entities and concepts of the application or problem domain.
Object Class:	an abstraction of a set of objects that specifies the common static and behavioural characteristics of the objects, including the public and private nature of the state and behaviour. A class is a template from which object instances are created.
Object System:	a description by a system developer of how potential users of an information system perceive an activity they are going to manage, control or analyse supported by an electronic

	information system, i.e. the designer's description of the user's mental model of the activity (see section 3.3.4)
Problem Area:	a description by a system developer of a field of activity, which the potential users of the information system intend to manage, control or analyse.
Area of use:	those parts of an enterprise in which an information system becomes used in order to manage, survey, or control a field of activity.
Information system:	a set of components organised to realise a dynamic model of an activity and an interface between this model and its users.

3.3.1 Object-oriented thinking

A new way of thinking (paradigm) is opened up in object-orientation by making a dynamic concept of 'object' a fundamental building block of descriptions. The concept of 'entity' from database theory offers a static element of description, whereas object in an object-oriented sense presumes some kind(s) of behaviour as a characteristic feature. Nouns like 'persons', 'cars', 'houses' etc orient people about classes of phenomena. Object class membership is more often than not decided on the basis of stable, shared features of the members of the object class. Behavioural properties, generally giving particular members of an object class a limited life span, are seldom considered when distinguishing object class membership in everyday conversations.

In object-oriented technology, an object class should be created for individual objects in two dimensions: data structure modelling and procedure modelling. In addition to the modelling aspects, operations at the object level are one of the novel features of object-orientation. Operations capture the behaviour of objects of object classes. For example, dancing and jumping are operations that can be carried out by or applied to object class Person.

In object-oriented systems development, object abstraction is used for describing similar things and gathering them into object classes, including class hierarchies. An object class is created for describing all instances of this object class at an abstract

level. Thus, object orientation is not only a new way of programming, but also a new way of thinking for most people involved in systems development.

3.3.2 Object system

A problem area and an object system describe what Jayaratna (1994) would call 'a part of an action world' and 'a part of a thinking world' respectively. The latter can be looked on as a designer's description of a small part of a user's understanding based on 'mental construct'. According to Jayaratna (1994) the object system comprises among other things 'models and frameworks' in which the object-oriented thinking should be reflected upon.

Designers base their object systems on descriptions given by some potential users of an information system. The actual users of the system bring their current 'mental constructs' to the task of interpreting output and (potentially) 'transform' these into action. Everyone, from systems users to designers use their 'mental constructs' in tackling different tasks at different times.

3.4 Themes in object orientation

Object-orientation as a new technology depends on the following concepts and their interconnections: object, abstraction, encapsulation and inheritance. These all play a fundamental role within object-oriented technology. The concept of object was discussed in Chapter one.

3.4.1 Abstraction

In object-oriented technology, abstraction is a process and a means for modelling 'reality' in physical and conceptual spaces. According to Wirfs-Brock and Wilkerson (1989) people typically understand the world by constructing mental models of portions of it. People try to understand phenomena in order to interact with them based on their mental models, which are simplified views of the phenomena. This construction process can be considered a type of model building, just as systems development has been viewed as a model building process. Models that people construct aid in the understanding and interaction. Abstraction as a mental process is essential in both cases and is crucial in our understanding of 'the action world'.

Firesmith (1993) regards an abstraction as a model that includes all essential capabilities, properties or aspects of what is being modelled without any extraneous details. Rumbaugh et al (1991) support this view

> 'The goal of abstraction is to isolate those aspects that are important for some purpose and suppress those aspects that are unimportant'.

The power of abstraction in object-orientation lies in selecting relevant objects from a problem area and constructing a whole system vision. Coad et al. (1997) summarise a strategy for selecting objects named 'not this time', which holds

> 'objects that are interesting, ones that you choose to consider, yet for one reason or another, you have decided not to include at this time'.

3.4.2 Encapsulation

The concept of encapsulation, or information hiding, means that the external aspects of an object are separated from its internal details, and hidden from the outside world. In this way the internal details of an object can be changed without affecting the applications that use it, provided that the external details remain the same. This prevents a program from becoming so interdependent that a small change has massive ripple effects. The only way to access an object's state is to send a message that causes one of the operations to execute. Strictly speaking the attributes are values, shorthand for operations that get and put values. This makes object class equivalent to abstract data type in programming.

Encapsulation in programming languages derives from the notion of abstract data types. In this context, an object consists of an interface and an implementation. The interface is the specification of the set of operations that can be invoked on the object and are its only visible part. The implementation contains the data, i.e. the representation or state of the object and the operations that provide, in whatever programming language, the implementation of each operation (Brown (1991), Elmasri and Navanthe (1994) and O'Neil (1994)).

3.4.3 Inheritance

Inheritance or classification hierarchy is a concept used by object-oriented methods when modelling a system structure and the relationships between classes. Inheritance is a mechanism for expressing similarities between classes, simplifying the definition

of classes similar to those previously defined. The ideas of inheritance have their roots in the study of artificial intelligence (AI), where semantic networks (Quillian 1967) and frames (Minsky 1975) are techniques for representing knowledge about stereotypical concepts and objects, and the relationship between more general and more specialised concepts is handled through the inheritance of properties and operations.

There is, however, an important difference between inheritance as discussed under object-oriented programming and that supported by AI. Normally, in object-oriented programming, objects inherit operations and attributes from their superior classes (superclasses), but they do not inherit the values of the attributes. An instance inherits the ability to have certain types of values. In AI and some database applications the ability to inherit values is supported (Graham 1995).

Several inheritance relations occur between classes, for example, simple and multiple inheritance; structural and behavioural inheritance. Simple inheritance means that a subclass is related to only one superclass, whereas multiple inheritance creates a subclass from more than one superclass. Structural inheritance is a mechanism for propagating operation declarations from superclasses to their subclasses. By contrast behavioural inheritance propagates what operations actually do, rather than how they are declared. It implies that behaviour and data associated with subclasses are always an extension of the properties of their superclasses.

With the inclusion of inheritance in object-oriented technology, the process of identifying objects and classes and their characteristics is subtly changed. Inheritance can be used to create new object classes from classes already recognised and defined. The process becomes one of recognition and presentation rather than one of identification – to find characteristics that are largely exhibited by an existing object, and then to define the specialisation necessary to characterise a new set of objects.

Instances (usually) inherit all the features of the class they belong to, but it is also possible in an object-oriented system to allow instances to inherit features from more than one superclass. If we consider a guppy, a typical pet fish, it belongs to both the class Fish and the class Pet. A guppy should, therefore, inherit properties from both of

these classes; multiple inheritance. The problem with multiple inheritance is that on rare occasions the properties from two (or more) parents may be directly or partially contradictory. Referring back to the guppy. A Fish lives in the sea and a Pet lives in the owner's home, where does a guppy live?

Naming conflicts can arise with multiple inheritance; that is several of the supertypes for a given subtype may have a property with the same name; furthermore those properties may or may not be semantically equivalent. If the name of an attribute in a class is the same as that defined in a superclass, the attribute of the superclass is not inherited. Another problem arises when multiple inheritance is itself subject to several different interpretations. For example, some employees are full time and some are part time. There are also full time and part time programmers and full time and part time secretaries. In this example an individual programmer inherits either the properties of full time employees or the properties of part time employees, but not both (Date 1995) (Fig 3.1).

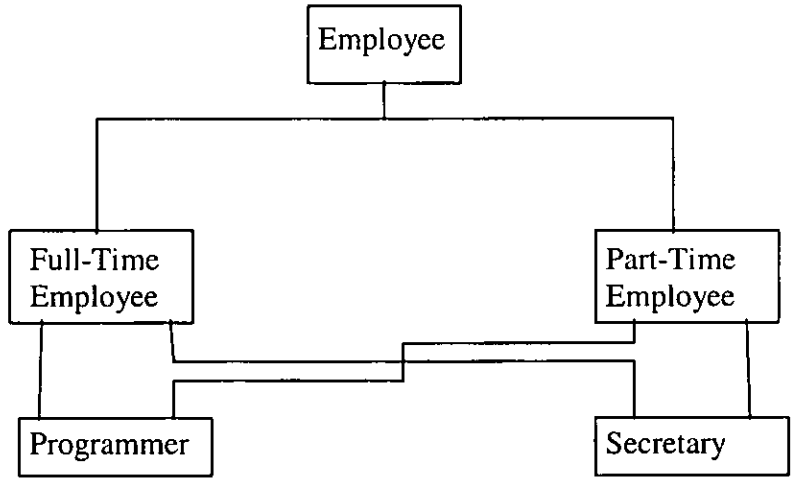


Figure 3.1 Multiple Inheritance

The important aspect of inheritance is the relationship that is established between the classes, as a superclass can be a subclass of other classes (Brown (1991), Bertino and Martino (1993) and Elmasri and Navanthe (1994)). Building an inheritance structure can start by seeking similarities between classes, similar data structure and similar behaviour and then assemble them in a system hierarchy. Inheritance can be seen as a way of increasing the number of classes in a system without changing the entire

system structure. This feature has been called 'local change' (Jacobson et al. 1998). A number of abstract classes are essential in an inheritance structure to allow for these changes.

The concept of class hierarchies and inheritance permits the specification of new types and classes that inherit much of their structure and operations from previously defined types or classes. This makes it easier to develop the data types of a system incrementally and to reuse existing type definitions when creating new types of objects (Bertino and Martino 1993).

3.5 Development of object-oriented analysis and design methods

There is no universally agreed approach to object-oriented analysis and design. In traditional structured 'waterfall' methodologies developers maintain a rigid distinction between analysis and design, which may be due to the fact that analysts and designers frequently occupy different professions. Analysis; when you determined what the application should do and design; when you decided how to do it. By keeping analysis and design separate you keep your options open, and avoid situations in which the system created is hampered by constraints imposed by some unspoken, but assumed system design.

One of the major problems encountered with structured analysis and design methods is the lack of overlap or smooth transition between the two. This often leads to difficulties in tracing the products of design back to the original user requirements or analysis products. The approach adopted in object-oriented analysis and design tends to merge the systems analysis with the process of logical design. However, there is still a distinction between requirements elicitation and analysis and between logical and physical design. Nevertheless, object-oriented analysis, design and programming, by working consistently with a uniform conceptual model of objects throughout the life cycle, at least promises to overcome some of the traceability problems associated with systems development. In these transitions the unit of currency remains the same: it is the object. Analysts, designers and programmers can use the same representation, notation and metaphor rather than having to use data flow diagrams at one stage, structure charts at the next etc.

Analysis is concerned with devising a precise, concise, understandable and correct model of the 'real world'. The purpose of object-oriented analysis is to model the real world system so that it can be understood. To do this the requirements must be examined and the implications analysed and restated. Important real world features must be abstracted first and small details deferred until later. A successful analysis model states what must be done, without restricting how it is done and avoids implementation decisions. The result of analysis should be understanding the problem as a preparation for design (Rumbaugh et al. 1991).

Object-oriented analysis contains an element of synthesis. The stages of abstracting user requirements and identifying key domain objects are followed by the assembly of those objects into structures that will support physical design at some later stage. The synthesis is a result of analysing a system, in other words imposing a structure on the domain.

There are three primary aspects of a system. These are respectively concerned with: a) data, objects or concepts and their structure, b) architecture or atemporal process: and c) dynamics or system behaviour. These dimensions are referred to as data, process and control. Object-orientation combines two of these aspects - data and process - by encapsulating local behaviour with data. Thus, an object-oriented analysis can be regarded as a form of syllogism moving from the particular (classes) through the individual (instances) to the universal (control).

Object-oriented analysis and design methods fall into two basic types, those which mimic existing structured methods by having separate notations for data, dynamics and process, and those which assert that, since objects inherently combine processes (operations) and data, only one notation is required. These approaches have been referred to as ternary (three-pronged) and unary respectively (Graham 1995).

Three pronged approaches have the advantage that existing practitioners of structured methods will be broadly familiar with the philosophy and notations, and proceeding in this way helps to ensure a smooth transition to object-oriented approaches for those

already skilled in traditional methods. Against this, unary approaches are a) more consistent with the metaphor of object-orientation and b) easier to learn from scratch.

OMT (Rumbaugh et al. 1991) the modified Ptech of Martin and Odell (1992) and OSA (Embley et al. 1992) are examples of the ternary approach. OOA (Coad and Yourdon 1991) and CRC (Wirfs-Brock et al. 1990) are examples of unary approaches.

Consideration of the existing object-oriented analysis methods available and published, highlight a number of points. For example; some, such as Coad and Yourdon (1991) are simple but lack support for describing system dynamics; Rumbaugh et al. (1991) OMT and Shlaer and Mellor (1988) are richer but more complex to learn. Methods like OMT offer little guidance to express business rules and constraints. Embley et al. (1992) OSA and Martin and Odell's (1992) synthesis of Information Engineering (IE) with Ptech are slightly simpler three pronged approaches. OSA allows the analyst to write constraints on the diagrams as an afterthought. None of these methods support business rules and there is little guidance on combining the products of the three separate models into a coherent single model, though OMT does provide slightly more help than others in this respect.

3.6 Brief overview of object-oriented analysis methods.

3.6.1 Event driven approach to object-oriented analysis

Shlaer and Mellor (1988), and Coad and Yourdon (1991) can be described as event driven approaches to object-oriented analysis. Event orientation is built on the partition of events and objects, which was introduced by Simula for 'discrete event simulation'. Event driven approaches are based on the idea that systems are stimulus-response mechanisms, the concept was originally created to expedite and standardise the creation of logical process requirements models. One logical process (essential activity) is created for each event type or class. That process represents the instantaneous and complete response to that event.

3.6.1.1 Shlaer and Mellor's Object Oriented Systems Analysis (OOSA)

Shlaer and Mellor (1988), object-oriented analysis method is an early example of object-oriented analysis, but it cannot really be regarded as object-oriented for several reasons, including the complete absence of any notion of inheritance. Their initial text is little more than an object based extension of data modelling. A later book (Object

Lifecycles: Modelling the World in States; Shlaer and Mellor 1991), introduced inheritance (entity subtyping) and the idea that modelling the lifecycles of entities with State Transition Diagrams (STD) could discover methods. Shlaer and Mellor also consider identifiers as sets of attributes or keys and apply normalisation rules to objects, regarding them as little more than relational tables.

This method may be classed as ternary and proceeds by creating an information (or data) model showing objects, attributes and relationships (Graham 1995). A state model is then developed that describes the states of the objects and transitions between states. Finally a Data Flow Diagram (DFD) defines the process model. The method is strongly influenced by relational design: objects are in first normal form and object identity is not a natural feature of designs.

3.6.1.2 Coad and Yourdon's Object-Oriented Analysis (OOA)

The basis for this method is in entity relationship modelling and was the first widely published account of a reasonably complete, practical, object-oriented analysis method and supporting notation, suitable for commercial projects. They shift the emphasis to the analysis phase as opposed to the design phase.

Coad and Yourdon (1991) suggest that analysis proceeds in five stages, which they identify as:

- Subjects: the problem area is decomposed into 'subjects' that correspond to the notion of 'levels' or 'layers' in dataflow diagrams.
- Objects: Objects are identified in detail by searching for business entities in much the same way as data analysis is performed. Little additional guidance is given on how to perform this task.
- Structures: Two completely different structures are identified, classification structures and compositional structures. This is where inheritance is considered, as the classification structures are specialisation/generalisation trees.
- Attributes: As in conventional data analysis, attributes are detailed and modality and multiplicity relationships specified using a version of extended relational analysis (ERA).

- Services: This is Coad and Yourdon's terminology for operations. Each object type must be equipped with operations for creating and deleting instances, getting and putting values and with special operations characterising the object's behaviour.

OOA is described as a unary method as it concentrates on analysis of data (Graham 1995). Design specific issues, such as threads of control, specific processors etc are added to the OOA diagram when the transition is made to Object Oriented Design (OOD). Great emphasis is placed on the similarity of notation for analysis and design and the way this eases the transition so sharply felt in traditional methods. However no real distinction is made between logical and physical design. Coad and Yourdon is weakest when it comes to specifying the dynamics of systems, but a state transition approach is suggested for object dynamics though no notation is defined. This method is critically evaluated using the NIMSAD framework in Chapter Five.

3.6.2 Model-driven approach

This approach is based on proven methods of information modelling and behaviour specification and is characterised by the development of models to represent different aspects of the analysis process. In general methods that are classified as using a model-driven approach include three integrated submodels: an object-relationship model for representing objects and their relationships to other objects; an object-behaviour model for representing object behaviour and an object-interaction model for representing interactions among objects. Building systems analysis models is not a step-by-step process, but a conceptual framework behind the analysis technique. A model-driven approach can be seen as a form of extended entity-relationship diagram and data flow diagrams. Rumbaugh et al. (1991), Embley et al. (1992) and Martin and Odell's (1992) Ptech can all be described as model-driven approaches to object-oriented analysis.

3.6.2.1 Rumbaugh et al's Object Modelling Technique (OMT)

Object Modelling Technique (OMT; Rumbaugh et al. 1991) is widely regarded as one of the most complete object-oriented analysis methods published to date. It is a three-pronged approach with strong roots in traditional structured methods and offers an extremely rich but complicated and detailed notation (Graham 1995).

OMT has three main phases or activities: analysis, system design and object design. Analysis assumes that a requirements specification exists and proceeds by building three separate models using three different notations. The first to be built is the Object Model (OM), which consists of diagrams similar to those of Coad and Yourdon (1991) and a data dictionary. The notation is fundamentally that of entity relationship (ER) modelling with operations and other annotations added to the entity icons. Next, for every object a Dynamic Model (DM) is built consisting of STD's. The third step is the Functional Model (FM), which to all intents and purposes is identical to DFD's. Operations discovered in both the DM and FM are then added to the OM and a walkthrough is performed, after which the analysis report, including the deliverables, is produced. This method is critically evaluated using the NIMSAD framework in Chapter Six.

3.6.2.2 Embley's Object-oriented systems analysis (OSA)

In this method the whole process of analysis is developed around models. OSA (Embley et al. 1992) begins with an object-relationship model (ORM), which permits the description of attributes, classification and aggregation structures and data semantics in the form of associations. More general constraints are written on the diagrams in note form next to the object they relate to, thus constraints have no relation to inheritance. The claimed advantage of this approach is that it does not constrain the analyst.

A state transition notation allows the analyst to describe the behaviour of each object. This object-behaviour model (OBM) defines the operations for the object, but the ORM need not be extended to include them. Message passing is described in a dataflow-like object interaction model (OIM). OSA is 'model-driven' in the sense that a model is constructed rather than asking the analyst to follow a fixed series of steps.

3.6.2.3 Martin and Odell Ptech

The ideas behind Ptech (Martin and Odell 1992) are based on the metaphor of process engineering as the production of systems assembling reusable components, an approach that separates analysis and logical design from implementation. The emphasis is therefore on what is done rather than how it is done. In that sense Ptech is process-oriented rather than strictly object-oriented. Ptech combines this process-driven view with the abstraction features of more data-centred, object-oriented design

and some ideas from set theory and artificial intelligence. Edwards (1989) the author of Ptech, stresses the need for a formal basis for any object-oriented method and criticises what he calls 'naïve' methods of object-oriented analysis such as Coad and Yourdon (1991) or OMT (Rumbaugh et al. 1991).

Ptech consists of three types of diagram: concept diagrams (broadly equivalent to extended entity relationship diagrams) called object schemata or concept schemata; event diagrams, which fulfil the role of state transition diagrams; and activity/function diagrams (with the same purpose as DFD's).

In methodological terms Ptech consists of four main activities: object structure analysis, object behaviour analysis, object structure design and object behaviour design. Object structure analysis involves building concept schemata showing classes, associations, classifications and composition. Object behaviour analysis uses event schemata and shows state transitions, event types, trigger rules, control conditions and operations. Object structure design adds implementation dependent aspects to the object structure analysis and object behaviour design adds the details of the operations. Ptech is described as having a very rich and expressive notation and method, but it is horrendously complicated to apply and 'takes a real expert to use it well' (Graham 1995).

3.6.3 Use case driven approach

The use case driven approach has made contributions to systems development as an industrial process. It stands out as being a truly object-oriented method, in which both the process and the methods are represented as objects. Use case is the core concept providing the linkage between requirements, development, testing, and final customer acceptance. System requirements included in use cases can be formally expressed and systematically tested. The use case model illustrates, in a straightforward way, the function of the business and the business processes they offer to the outside world. OOSE (Jacobson et al. 1998) is a prime example of the use case approach.

3.6.3.1 Jacobson et al's Object-Oriented Software Engineering (OOSE)

Object-Oriented Software Engineering (OOSE; Jacobson et al. 1998) is a method for object-oriented analysis and design derived from Jacobson's Objectory. Many of the ideas of OOSE are similar to those of other object-oriented analysis methods, but one

original idea stands out: the use case. Use cases are descriptions of how users interact with a system. These can be viewed as either process bubbles in a DFD style context diagram or as scripts similar to those found in AI research. Scripts (or use cases) can have subscripts to deal with exceptions. The advantage of this approach is that requirements can be traced right through the lifecycle, and this is what OOSE emphasises. It is also straightforward to generate test scripts from use cases.

Use cases appear in the requirements model along with actors. Actors are external entities with which the system interacts. The actors are used to find the use cases and are instantiated in actual interactions with users. An event can initiate several use cases depending on the state of the system (mode) and the purpose and subsequent behaviour of the user. The use case model is used to generate a domain object model with objects drawn from the entities of the business. This is then converted into an analysis model by classifying the domain objects into three types: interface objects, entity objects and control objects. Entity objects emphasise storage and are stable in the sense that they persist longer than a use case. Control objects emphasise behaviour. The analysis model supports inheritance and instance associations, which are used to record aggregation. The analysis model is then converted to a design model expressed in terms of 'blocks' that are implementations of one or more objects This method is critically evaluated using the NIMSAD framework in Chapter Seven.

3.6.4 Role driven approach

Object-Oriented Role Analysis and Modelling (OOram; Reenskaug et al. 1996) is an object-oriented method that assumes an object can play several roles. Analysis describes subproblems of encapsulating behaviour in the objects of an object model, which is termed a role model. A role model functions as a conceptual model that combines the expressiveness of the object and the class. All information that can be expressed in a class-based model and an object-based model can be expressed in a role-based model. The most important contribution of this method is the 'added leverage' (Reenskaug et al. 1996) provided by role modelling, for example, how to create object-oriented models of interesting phenomena, compared with the conventional class modelling.

OOram consists of some basic concepts: objects, role, types and object class. Every object has its own identity and attributes, and is encapsulated so that the messages it sends and receives constitute all its externally observable properties. The object abstraction entails three kinds of abstraction around recognised roles: why, what and how. The 'why' abstraction is a concept called 'role', which represents the task of the object within the organised structure of objects. All objects that serve the same purpose in a structure of collaborating objects, as viewed from the context of some area of concern, play the same role. The type is the 'what' abstraction. All objects that exhibit the same externally observable properties belong to the same type. The class is the 'how' abstraction. All objects that share a common implementation are considered to belong to the same class.

This method focuses on three critical operations: roles, synthesis and structuring. It models small subproblems initially, and then combines small models into larger ones in a controlled manner using both inheritance (synthesis) and run-time binding (structuring) (Reenskaug et al. 1996). These operations are based on encapsulation, inheritance and dynamic binding.

3.6.5 Responsibility driven approach

Responsibility driven design by Wirfs-Brock and Wilkerson (1989) is a method that models an application in terms of classes, their responsibilities and the collaborations. Responsibilities are a way to apportion work among objects that comprise an application. This approach focuses on what actions must be accomplished and what objects accomplish the actions. How each action is accomplished is deferred until after an object model and its interactions have been created and understood. The 'system's' responsibilities are analysed and allocated to the classes of the 'system' Finally, the collaborations between classes of objects that must occur in order to fulfil the responsibilities are defined. This gives a preliminary design, which can be further explored through hierarchies, subsystems and protocols.

Responsibilities include two key items: the data that an object maintains, and the actions that an object can perform. They are meant to convey a sense of the purpose of an object and its place in the system. The responsibilities of an object are all the operations it provides for all objects that communicate with it. In this manner,

common responsibilities are grouped together. Object classes fulfil their responsibilities in one of two ways: performing the necessary computation themselves and collaborating with other objects classes. Involving superclasses embodies general responsibilities and subclasses that specify operations and hierarchies of object classes.

3.6.6 Unified Modeling Language (UML)

The lack of standardisation for object-oriented modelling was identified as a serious problem during the 1990's, and the fact that every method, tool and practice had its own set of symbols and terminology resulted in confusion. Rumbaugh, Booch and Jacobson's Unified Modelling Language (UML) (Rational 1997) was developed to solve some of the perceived problems of the object-oriented community. UML is based on the methods of Rumbaugh et al. (1991), Booch (1994) and Jacobson et al. (1993) and is aimed at creating a standard modelling language not a methodology. There are important differences between a methodology and a modelling language. A methodology is an explicit way of structuring one's thinking and actions. A modelling language lacks a process or the instructions for what to do, how to do it, when to do it and why it is done (Booch et al. 1996; Rumbaugh 1996; Mellor and Lang 1997; Eriksson and Penker 1998; Harmon and Watson 1998)). UML is described as a modelling language and leaves the initial choice of approach to analysis to the analyst. It would, therefore, be difficult to determine how the modelling language assists the analyst in determining objects when they can generate an initial object model by using Jacobson et al. (1993) use cases, Rumbaugh et al's (1991) object model or Booch's (1994) object 'bubbles'.

3.7 Justification of choice of methods to be evaluated

Given the vast range of methods available a decision had to be made as to which object-oriented analysis methods would be considered for critical evaluation, the decisions taken were subjective. However, the chosen methods had to meet the following criteria:

a) be well-known and dominant
b) be used in teaching and training
c) be used in industry and have software support
d) be available as a text.

Cost and lack of coverage of systems development steps were considered as selection criteria and rejected as not being relevant to the identification and formulation of objects.

3.7.1 OOA (Coad and Yourdon 1991)

Coad and Yourdon's (1991) OOA satisfies the criteria identified above. OOA was the first widely published account of a reasonably complete, practical object-oriented analysis method suitable for commercial projects. OOA is described as taking a unary approach, that is, one model is developed that incorporates information about data, processes and dynamics or system behaviour. This is claimed to be in keeping with the philosophy of object-orientation and makes it easy to learn (Graham 1995).

OOA has been used by a number of authors, de Champeaux and Faure (1992), Fichman and Kemerer (1992), Iivari (1994) and Losavio et al. (1994) who compared object-oriented analysis methodologies using varying sets of criteria. OOA is also described in many textbooks (Rumbaugh et al. 1991; Graham 1995; Jacobson et al. 1998) that consider object-oriented analysis methodologies.

'Object Models, Strategies, Patterns and Applications' (Coad et al. 1997) was considered as being more 'up-to-date' than Object Oriented Analysis (Coad and Yourdon 1991). However, the approach taken in 'Object Models, Strategies, Patterns and Applications' (Coad et al. 1997) is to present a number of scenarios (initially transcripts of interviews) using different problem domains and to apply a number of strategies in order to develop object models. Having worked through the examples, in the text, the reader is deemed to have 'experiential background' and can then select the relevant strategies and patterns to apply to any problem domain. 'Object Models, Strategies, Patterns and Applications' (Coad et al. 1997) uses the notation of OOA (Coad and Yourdon 1991) but has little if any theoretical explanation as to why the specified strategies have been chosen, hence my decision to concentrate on the initial text.

3.7.2 Object-Oriented Modelling and Design (Rumbaugh et al 1991)

Rumbaugh et al's (1991) Object Modelling Technique (OMT) again satisfies each of the criteria listed above. OMT is described as taking a ternary approach, and a model

driven approach (Graham 1995) as three separate models are developed to represent data, process and dynamics. OMT makes use of the principles of entity-relationship modelling, state transition diagrams and data flow diagrams. These are seen to be broadly similar to approaches taken by traditional structured methods, as a result transition from traditional approaches to object-oriented approaches should be easier for those already skilled in traditional methods.

de Champeaux and Faure (1992) and Iivari (1994) used OMT as one of their selected methods when comparing object-oriented analysis methodologies. OMT is also frequently described in textbooks, for example, Graham (1995) and Jacobson et al. (1998).

3.7.3 Object Oriented Software Engineering (Jacobson et al 1998)

Jacobson et al's (1998) OOSE takes a use-case driven approach to object modelling. Use cases are descriptions of how users interact with a system. These can be text based or process 'bubbles' as in a DFD. The use case model is then used to generate a domain object model with objects drawn from the entities of the business. OOSE is a simplified version of Objectory. Objectory was derived from experience in building telephone exchange systems and is one of the oldest object-oriented methods. However, as OOSE is a subset of Objectory it is said to be inadequate for use in production (Jacobson et al 1998) as much of the complexity of Objectory deemed necessary for production is missing. Jacobson et al (1998) recommend that in order to use OOSE for production you need to buy consultancy.

de Champeaux and Faure (1992) and Iivari (1994) consider OOSE in their comparisons of methodologies. Rumbaugh et al. (1991) and Graham (1995) also consider OOSE in their comparisons of object-oriented analysis methodologies.

3.8 Summary

This chapter considered the terminology, both the general terminology and the specific object-oriented terminology used within the thesis. Development of object-oriented systems and object-oriented concepts were then considered. The development of object-oriented analysis methods followed and a small number of object-oriented

analysis methods were described in order to demonstrate the various approaches taken by object-oriented analysis developers. The selection of methodologies to be evaluated was then justified.

Chapter four considers different strategies for comparing methodologies. These include the use of taxonomies, visual comparisons, descriptive comparisons, structured thinking and evaluation frameworks. Mulitview (Avison and Wood-Harper 1990), Avison and Fitzgerald's framework (1995) and NIMSAD (Jayaratna 1994) are considered as evaluation frameworks. The selection of NIMSAD is also justified.

Chapter Four: Evaluation Framework

The previous chapter identified and justified the choice of methodologies to be critically evaluated. In order to critically evaluate methodologies it is necessary to use an evaluation methodology or a framework that will enable us to focus on specific aspects of the methodology under investigation.

This chapter initially considers why it is important to compare methodologies; this is followed by strategies for comparing methodologies. These include the use of taxonomies, visual comparisons, descriptive comparisons, structured thinking and evaluation frameworks. Mulitview (Avison and Wood-Harper 1990), Avison and Fitzgerald's framework (1995) and NIMSAD framework (Jayaratna 1994) are considered as evaluation frameworks. The choice of NIMSAD (Jayaratna 1994) as the evaluation framework, to be used, is then justified.

4.1 Background to Methodologies

In 1994 it was estimated that there were over 1000 brand-name methodologies in use all over the world (Jayaratna 1994). Many of these are similar and are differentiated only for marketing purposes; others are internal to individual companies and have been developed in-house (Avison and Fitzgerald 1995). In order to compare methodologies it is necessary to apply 'a set of criteria' or 'a framework' to ensure that the methodologies under investigation are treated in the same way. Comparisons based on 'a set of criteria' are generally found in methodology books, for example, Wirfs-Brock and Johnson (1990), Page-Jones (1992), Rumbaugh et al. (1991); and journal or research papers, for example, de Champeaux and Faure (1992), Iivari (1994) and Fichman and Kemerer (1992). The methodology books tend to compare their own methodology with those of others, whereas an independent comparison is made among chosen methodologies in journal or research papers.

4.1.1 Comparison of Methodologies

There are two main reasons for comparing methodologies:

- An academic reason: to better understand the nature of methodologies (their features, objectives, philosophies etc) in order to perform classifications and to improve future information systems development

- A practical reason: to choose a methodology, part of a methodology, or a number of methodologies for a particular application, a group of applications or for an organisation as a whole.

The two reasons are not totally separate as academic studies help in the practical choices and the practical reasons influence the criteria applied in the academic studies.

The earliest attempts to compare methodologies were a series of conferences known as CRIS (Comparative Review of Information Systems Design Methodologies). These conferences were organised by the IFIP (International Federation of Information Processing) working group. The first conference in 1982 invited authors of methodologies to describe their methodologies and apply them to a case study. The chosen case study was the organisation of an IFIP conference that was specified to the authors. The most interesting outcome of this was to illustrate the wide differences between methodologies and what they produced from the same case study.

The second conference in 1983 consisted of a series of papers, which analysed and compared the features of various methodologies and the third conference in 1986 addressed the issues of improving the practice of using methodologies. A fourth conference in 1988 examined the ways in which computers could be used during the construction of information systems. Olle et al. (1991) is based on the work of these conferences. However the conferences failed to resolve any of the issues they set out to achieve.

4.2 Strategies for comparing methodologies

4.2.1 Taxonomies

This approach involves the invention of a classification scheme in which each methodology is placed at an appropriate position in the scheme. For instance Davis and Wood-Harper (1990) categorise requirements determination methodologies according to their main information-gathering approach. Their categories are (a) asking, (b) deriving from the existing system, (c) synthesising from the utilising system and (d) discovering from experimentation with an evolving information system (prototyping).

Lyytinen (1987) presents a more abstract taxonomy of the various approaches to information systems development. Lyytinen stresses the importance of assigning equal weighting to technology, language and organisation. For each methodology, he asks which contexts the methodology incorporates and for what types of system the methodology is suitable. Lyytinen is interested in identifying the fundamental assumptions made by each methodology, which are often left unstated. His framework classifies methodologies into five groupings, for example,

Grouping	**Author/Example**
Technical design-oriented	Most well-known and widely used methodologies, such as structured analysis and ISAC
Socio-technical	ETHICS
Decision oriented	DSS (Keen and Scott-Morton 1978) Mason and Motroff's MIS programme
Sense making	Boland
Communication oriented	Lyytinen and Lehtinen

(adapted from (McGinnes 1993))

4.2.2 Visual Comparisons

Another popular way of comparing methodologies is to rate them in various ways and then to plot the ratings as co-ordinates in a two dimensional space. Although it is claimed that 'positioning' methodologies in this way can assist in understanding their relationships to one another, the methodologies sometimes do not fit the classifications neatly. An example of a visual comparison is given by Wood-Harper (1985) who discusses a highly abstract two-dimensional framework that positions methodologies using two dimensions (a) subjective/objective and (b) conflict/regulation.

4.2.3 Descriptive comparisons

Possibly the most common approach for comparison of methodologies is to define a set of criteria and then to describe or evaluate how each methodology addresses each criteria. The set of criteria may be very simple or very complex. Wilson (1984) categorises methodologies in a very simple way, according to whether they (a) focus on verbs or nouns, and (b) consider information or data. Maddison (1984) gives a very comprehensive analysis with many separate points of comparison.

Inherent in any comparison of information systems methodologies is a view of what the methodologies are for and what their priorities ought to be. Many of the comparative frameworks outlined take a predominantly technical view, concentrating

on the construction of information systems as a technical problem solving exercise. Davis and Wood-Harper (1990) point out that ideally, an information system methodology will be able to address both technical and social or political issues. They consider methodologies under four headings which help to bring out these important issues: (a) the methodology's assumptions about reality, (b) assumptions about the information system, (c) objectives in using the methodology, and (d) conceptual models/constructs used by the methodology.

4.2.4 Structured thinking

Jayaratna (1994) argues that a fundamental distinction should be made between two modes of use of the notion of 'systems' when formulating problems. These two modes (ontological and epistemological) guide methodology users to structure thinking from different philosophical assumptions of reality. One takes a 'system' as something existing in an enterprise independent of various actors, for example, designers. The problem solvers or system designers require a correct, but possibly incomplete, description of the system. This view is based on positivistic paradigms that help to structure 'reality' by making the methodology users look for 'factual' data about a single correct state. This data model then becomes a 'true' representation of the situation. Many structured methodologies have adopted this view; for example, DeMarco (1979) and Coad and Yourdon (1991). Although there is an acceptance of multiple views, object-oriented methodologies believe in the definition and creation of one final correct system.

In structured methodologies, the notion of 'systems' is used in an ontological sense, that is systems are taken as given without question. Methodology users search for the system and do not question its boundary. Methodology users are frequently unconsciously conditioned by clients to arrive at a given system's boundary with which they agree Fig 4.1. (Jayaratna 1994)

These 'hard' systems approaches are concerned with finding a solution to a given problem (Lewis 1994). However, in many instances, a potential solution is already seen to exist, in this respect the 'hard' systems approach is similar to an engineering approach. For example, the civil engineers work begins when the need for a river to be crossed has been identified; that is, there is already a solution in mind prior to the design phase. The 'hard' systems approach uses the concept of 'system' ontologically,

that is, as a label for things in the 'action world' and analysis proceeds on the basis that the world is composed of systems and subsystems.

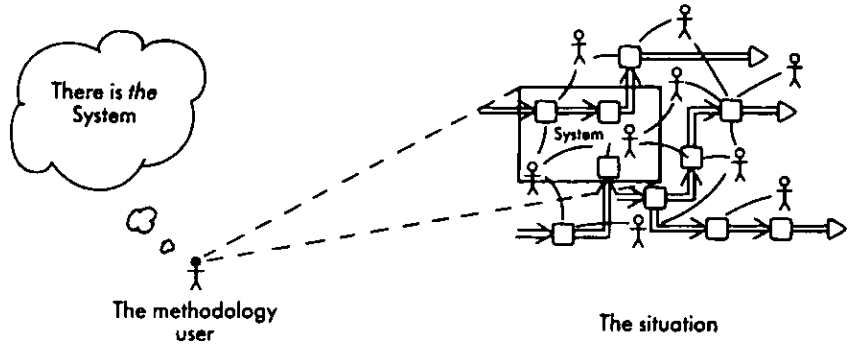


Figure 4.1 Ontological notion of a 'system'.

(Jayaratna 1994)

An epistemological mode takes a 'system' as a way of conceptualising the analysis of the situation of concern, from this perspective no 'system' can be taken 'as given' and several systems could be constructed to support analysis and discussion. This mode of thinking can help to follow hermeneutic paradigms and can make the methodology user search for many states, any or all of which could be argued to have the same truth-value. Checkland (1981), Mumford (1983a and 1983b), Checkland and Scholes (1990) and Checkland and Howell (1998) use this mode in Soft Systems Methodology (SSM). Methodology users employ this notion for conceptualising many system boundaries, any of which can be considered as relevant and useful for gaining an insight into the situation. This means that the methodology user has to justify why he or she considers a particular selection as a system Fig. 4.2. (Jayaratna 1994)

Soft systems' thinking differs from other systems approaches in a number of ways: it abandons the goal-seeking model of human behaviour and rejects the aims of engineering systems that will meet objectives. Soft systems approach considers the concept of 'system' to be an epistemological device for thinking about some part of the 'action world' rather than an ontological description of part of the 'action world' (Lewis 1994). From a soft systems perspective the notion of system and sub-system are only 'mental constructs' through which we may choose to make sense of the external world.

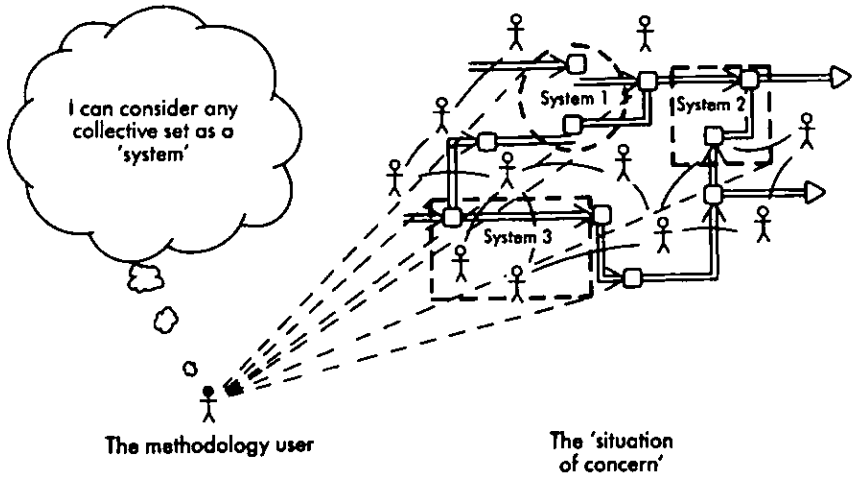


Figure 4.2 Epistemological notion of 'systems'.

(Jayaratna 1994)

4.2.5 Frameworks

The aim of a conceptual framework is to assist the methodology user in understanding the problem situation. The 'real world' used in literature serves to distinguish between academia and industry or between theory and practice. This implies that the latter is complex and full of action and the former is less complex and involves only thinking (see section 3.1).

The clients concern for change in the 'action world' prompts them to seek the help of external problem solvers. However, both clients and problem solvers may have some initial expectations of the possible outcomes. For a framework to be effective it should consider not only the problem situation (the methodology context) but also the 'mental constructs' of the intended problem solver and the methodology under scrutiny (Bessagnet et al. 1994).

4.3 A Framework for Comparing Methodologies - Avison and Fitzgerald

Avison and Fitzgerald's (1995) framework for comparative analysis builds on the earlier work of Wood-Harper and Fitzgerald (1982) and Fitzgerald et al. (1985). The analysis is based on a discussion of the approaches in terms of three criteria: paradigm, model and objective. In the framework the analysis is extended to include a variety of other criteria. There are seven basic elements to the framework:

Philosophy
- Paradigm
- Objectives
- Domain
- Target

Model
Techniques and tools
Scope
Outputs
Practice
- Background
- User base
- Players

Product.

The headings are not mutually exclusive and there are inter-relationships between them. For example, aspects of philosophy are reflected in some sense in all other elements.

4.3.1 Philosophy

The question of philosophy underpins all other aspects of a methodology, and it distinguishes a methodology from a method. The choice of areas covered by the methodology, the systems, data or people orientation, the bias or otherwise towards computerisation, and other aspects are made on the basis of the philosophy of the methodology. Paradigm, objectives, domains and applications are seen as guiding the philosophy.

Wood-Harper and Fitzgerald (1982) identify two paradigms of relevance: the science paradigm, characterised by most of the hard scientific developments in the latter part of the twentieth century and the systems paradigm, characterised by a holistic approach.

The stated objectives offer an obvious clue to the methodology philosophy. Some methodologies state that the objective is to develop a computerised information system. The problem with concentrating solely on the aspects to be computerised is that this is an artificial boundary in terms of the logic of the business. A methodology that concerns itself solely with analysing the need for a computer solution is quite a different methodology from one that does not.

Early methodologies, such as the conventional life cycle approach saw their task as overcoming a particular and sometimes narrow problem. The solution of a number of these kinds of problems on an ad hoc basis at a variety of different points in time can lead to a mish-mash of different physical systems being in operation at the same time. More recent methodologies take a wider view of their starting point and initially are not looking to solve a particular problem.

The last aspect of philosophy is the applicability of the methodology. Some methodologies are specifically targeted at particular types of problem, environment, or type or size of organisation, whilst others are said to be more general.

4.3.2 Model

The model is the basis of the methodology's view of the world; it is an abstraction, and a representation of the important features of the information system or the organisation. The model works at a number of levels: a means of communication; a way of capturing the essence of a problem or a design, it represents an insight into the problem or area of concern. Traditionally abstraction is to a conceptual level, logical level and physical level. The conceptual model is a high level description of the universe of discourse or the object system from a particular perspective. Conceptual models cause confusion as the term is used in different ways in various contexts. Confusion is created due to the language of conceptual modelling being the same as for logical modelling. The logical level is a description of the information system without any references to technology that could be used to implement it. The physical level is a description of the information system including the technology of the particular implementation.

4.3.3 Techniques and tools

A key element of a methodology is the identification of the techniques and tools used in the methodology. These can include: rich pictures, root definitions, entity modelling, normalisation, data flow diagrams, decision trees, decision tables, structured English, Action diagrams, entity life cycle, object orientation, structure diagrams and matrices.

4.3.4 Scope

Scope is the indication of the stages of the life cycle of systems development that the methodology covers. An analysis of the level of detail at which each stage is addressed is useful. The problem with using stages of the life cycle as a basis for the examination of scope is that there is no real agreement on what are, or should be, the stages of the cycle.

4.3.5 Outputs

It is important to know what the methodology is producing in terms of deliverables at each stage and, in particular, the nature of the final deliverable. This can vary from being an analysis specification to a working implementation of the system.

4.3.6 Practice

This is measured according to: the methodological background; the user base; the participants in the methodology; and what skill levels are required. The practice should also include an assessment of difficulties and problems encountered, and perceptions of success and failure. It is also important to assess any differences there appear to be between the practice and the theory of the methodology.

4.3.7 Product

This is what the customers actually receive at the end of the development process. This may consist of software, written documentation, and agreed number of hours training, a telephone help service, consultancy etc.

4.4 MULTIVIEW

The foundations of Multiview (Avison and Wood-Harper 1990) as an enquiring framework for IS development rest on a recognition that the needs of computer artefacts, organisations and individuals must be jointly considered. Multiview perceives information systems development as a hybrid process involving computer specialists, who will build the system, and users, for whom the system is being built. The methodology therefore looks at both the human and technical aspects of information systems development (Vidgen 1998).

The approach adopted has been used on a number of projects, and the methodology itself has been refined using 'action research' methods, that is the application and testing of ideas developed in an academic environment into the 'real world. Avison and Wood-Harper (1990) describe Multiview as an exploration in information systems development. It therefore sets out to be flexible: a particular technique or aspect of the methodology will work in certain situations but is not advised for others.

The stages of the Multiview methodology and the inter-relationships between them are shown in Fig 4.3. The boxes refer to the analysis stages and the circles to the design stages. The arrows between them describe the inter-relationships. Some of the outputs of one stage will be used in a following stage. The dotted lines show other major outputs.

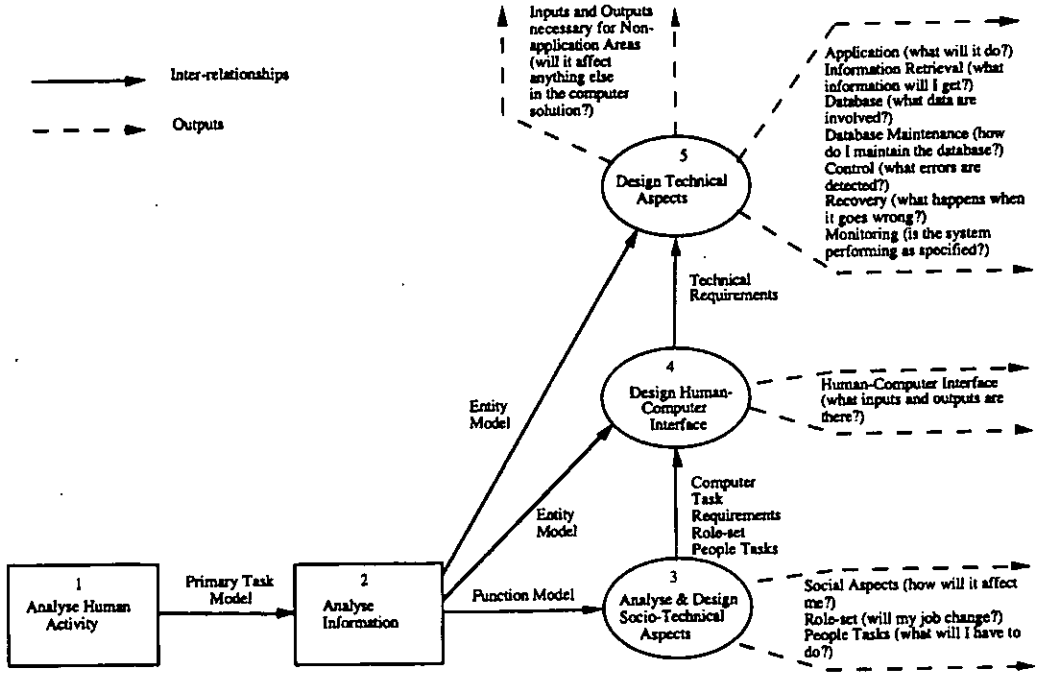


Figure 4.3 The Multiview Framework

(adapted from Avison 1992)

The five stages are:

1. Analysis of human activity
2. Analysis of information (information modelling)
3. Analysis and design of socio-technical aspects
4. Design of the human-computer interface

5. Design of the technical aspects.

They incorporate five different views that are appropriate to the progressive development of an analysis and design project, covering all aspects required to answer the vital questions of the users. These five views are necessary to form a system that is complete in both technical and human terms. The outputs of the methodology, shown as dotted lines in Fig 4.3 are listed in Table 4.1 together with the information they provide and the questions they answer.

Outputs	Information
Social Aspects	How will it affect me?
Role-set	Will my job change? In what way?
People Tasks	What will I have to do?
Human-Computer Interface	How will I work with a computer? What inputs and outputs are there?
Database	What data are involved?
Database Maintenance	How will I maintain the integrity of the data?
Recovery	What happens when it goes wrong?
Monitoring	Is the system performing to specification?
Control	How is security and privacy dealt with? What errors are detected?
Information Retrieval	What information will I get?
Application	What will the system do?
Inputs and Outputs necessary for Non-Application Areas	Will it affect anything else on the computer subsystem?

Table 4.1 Methodology Outputs

(adapted from Avison 1992)

Because it is a multi-view approach, it covers computer related questions and also matters relating to people and business functions. It is part issue-related and part task-related. An issue-related question is 'What do we hope to achieve for the company as a result of installing a computer?' A task-related question is: 'What jobs is the computer going to have to do?'

The first stage looks at the organisation: its main purpose, problem themes and the creation of a statement about what the information system will be and what it will do. The second stage is to analyse the entities and functions of the problem situation described in stage one. This is carried out independently of how the system will be

developed. The philosophy behind the third stage is that people have a basic right to control their own destinies and that if they are allowed to participate in the analysis and design of the systems that they will be using, then implementation, acceptance and operation of the system will be enhanced. This stage emphasises the choice between alternative systems, according to important social and technical considerations. The fourth stage is concerned with the technical requirements of the user interface. The design of specific conversations will depend on the background and experience of the people who are going to use the system, as well as their information needs.

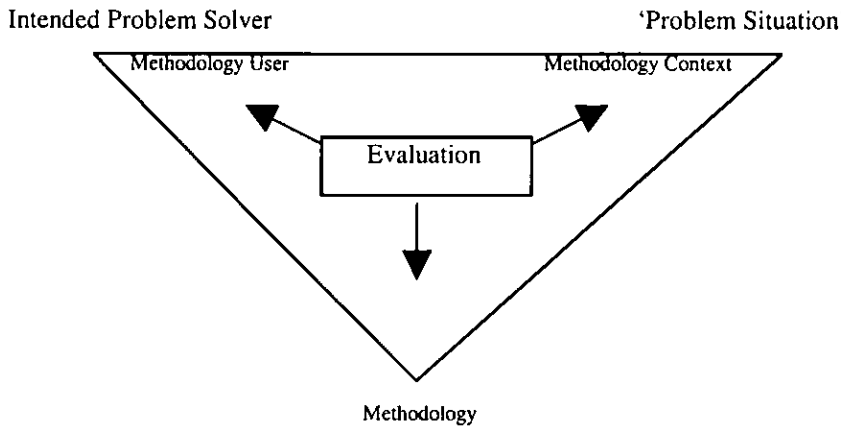
Finally the design of the technical subsystem concerns the specific technical requirements of the system to be designed, and therefore to such aspects as computers, databases, control and maintenance. Although the methodology is concerned with the computer only in the latter stages, it is assumed that a computer system will form at least part of the information system.

4.5 A Framework for Method Evaluation – NIMSAD

4.5.1 Introduction to the NIMSAD framework

The Normative Information Model-based Systems Analysis and Design (NIMSAD) (Jayaratna 1994) framework defines a problem solving methodology as a goal-directed way of solving problems. The NIMSAD framework is a systemic meta-model for understanding and evaluating a methodology, and explains the process of solving problems. NIMSAD has four essential elements to assess the problem-solving capabilities of each methodology. These elements and their relationships are shown in Fig 4.5.

This figure gives a complete picture of the framework, which covers elements, phases and stages for solving a problem. The evaluation as an essential element should be carried out in order to measure how effective the methodology is in solving the problem, although phases and stages can differ from one methodology to another.



Problem -solving Process

Elements	Phases	Stages
The 'problem situation'		
The 'problem solver'		
The problem solving process	Problem Formulation	Understanding the 'situation of concern'
		Performing the diagnosis
		Defining the prognosis outline
		Defining 'problems'
		Deriving notional systems
	Solution design	Performing logical design
		Performing physical design
	Design implementation	Implementing design
Evaluation	The 'problem situation'	
	The problem solving process	
	The problem solver - (before, during and after the intervention)	

Figure 4.5 NIMSAD elements and stages.

(Jayaratna 1994)

The four elements include:

- The 'problem situation' (the methodology context) in which one special methodology is going to be applied, and the information systems will be designed, implemented and operated in this situation;
- The intended problem solvers (the methodology users) and their 'mental constructs', which influence their thinking processes and actions.
- The problem solving process (the methodology), which deals with the concrete phases, stages and steps to identify and resolve problems. This element focuses on the major parts of a methodology;

- The evaluation of the above three elements before, during and after the methodology application.

The NIMSAD framework does not limit itself to situations that are perceived as causing problems, or to well-structured problems, the framework can be equally well applied to badly structured problems. The framework aims to assess systems methodologies that purport to transform a current state into a desired state.

4.5.2 The 'problem situation'

Organisations are perceived as the contexts for information systems. Thus, the 'problem situation' is identified by clients within the context of an organisation, and investigated by the intended problem solvers.

To know the 'problem situation' is to know the organisation where the methodology will be applied. Jayaratna (1994) suggests that a useful way to gain this knowledge is to take the organisation as purposeful systems. The intended problem solvers have to consider organisational issues in the 'problem situation' whether or not they are directly related to information systems. This is because organisational issues may constrain the current requirements; if they are taken into consideration a 'fuller picture' of the problem situation is derived that could affect IS development in the future. By concentrating effort on the area where a problem is perceived to occur the intended problem solvers are determining a boundary that may not be central to the problem situation. The 'current states' should be transferred into 'desired states' by intended problem solvers. The 'desired states' expected from users should be transformed to a desirable situation for the organisation. An important point for discussion is that there is a difference between the two 'states', (i.e. a problem). At a conceptual level this allows us to define any problem as:

> '... the difference between perceived 'reality' and perceived 'expectation' for that 'reality', together with a desire to make the perceived 'expectation' become 'reality'' (Jayaratna 1994).

If the two 'states' were seen to be the same then there would not be a problem and the services of an intended problem solver would not be required.

To make a better transformation, Jayaratna (1994) identifies 'interpersonal relationships with clients and others as an ongoing process' which is as important as the management of the 'problem situation'. Communication between different people involved in the process is seen as an important issue as they facilitate better dialogue. However, they do not guarantee the discovery of 'desired states'.

4.5.3 The intended problem solvers and their 'mental constructs'.

In the discussion of intended problem solvers, Jayaratna (1994) pays much attention to what makes them perceive and resolve problems in particular ways. The framework helps to identify pre-conditions of the 'mental constructs' in an explicit way. The 'intended problem solvers' could be thought of as the 'methodology users', who apply methodologies to solve problems, i.e. analysts, users, or clients.

The NIMSAD framework advocates that effective problem solvers need to acquire the richest possible understanding of the organisation, not merely rely on their clients. The 'mental constructs' of the intended problem solvers reflect their individual characteristics and these together with their understanding of the organisation should result in the success of effective and efficient information processing systems design and development. On this basis, Jayaratna (1994) explores what makes the intended problem solver select some elements of the 'action world' as relevant, significant and useful while dismissing others as being irrelevant, insignificant and useless (Jayaratna 1994). This selection process is shaped by several characteristics that constitute the intended problem solvers 'mental constructs'. A 'mental construct' differs from one person to another. Its importance, expressed explicitly in Jayaratna (1994), is unique in the way that 'mental constructs' of intended problem solvers identify important personal characteristics that affect the way in which the role is carried out.

Methodologies cannot structure all issues within a problem situation, some issues rely on 'shared understanding' etc. The problem solver(s) need to be aware of their pre-conceptions, prejudices, values etc that may have a bearing on how they interpret the problem situation.

According to Jayaratna (1994) 'mental constructs' are dynamic, which means external inputs and critical self-reflection can influence the intended problem solver and

thereby change their future actions. The problem solver could, therefore, derive new meanings from the same situations, consequently the problem solver should be aware of any influencing factors that are likely to modify his/her understanding of the problem situation. Jayaratna (1994) identifies the elements that influence 'mental construct' as:

Perceptual process: One of the most influential characteristics of the intended problem solver's 'mental model'. 'It acts as a filter of information to the 'action world' and determines what information is to be significant' (Jayaratna 1994). The perception of the intended problem solver could be different from that of the client who is defining the problem, awareness of this enables the methodology user to make an effort to understand the filtering process used by others.

Values: The intended problem solver's beliefs of what is considered to be 'good', their principles or standards, these beliefs, standards and principles are accepted without question as they are an intrinsic part of the being and/or the culture to which one belongs.

Ethics: This relates to the standards placed on a person's expected behaviour.

Motives: The needs that people try to satisfy in a given situation, but keep to themselves. This includes the personal motives for being involved in the problem-solving situation that we may or may not be explicitly aware of. While these will provide a major range of 'easy', and 'obvious' solutions, the same models may prevent us from exploring new ideas or becoming effective listeners to others' ideas.

Reasoning: The ability to abstract essential aspects from any situation and to understand the concepts underlying the thought processes, that is, determine what makes us reason in a particular way. 'Systems thinking' can be used as a reasoning process.

Prejudices: These can be defined as persistent opinions that are formed from one's values, experiences or insecurities. They are useful in that they can reduce time spent on information gathering, but can also prevent methodology users from receiving valid information. Feedback from others is very useful for examining one's prejudices. As far as possible prejudices should be acknowledged and effort made to ensure that they do not unduly influence the information gathering process.

Experiences: An invaluable source for developing knowledge and skills for forming implicit models for constructing the understanding of situations. Experience makes the intended problem solver more confident and able to assume 'expert' status.

Knowledge and skills: These are acquired from education, training and experience and are required in order to practice a methodology. The intended problem solver ' must become very conscious of the knowledge sets and skills that are required to practice a methodology, and the methodology creators must state what knowledge sets and skills we should possess if we are to become effective users' (Jayaratna 1994).

Structuring processes: These are unique to each individual. A methodology may be used to explicitly structure, an intended problem solvers, thinking and actions. By reconstructing their thinking and actions the problem solver gains new insight about the situation.

Roles: These can be defined as the set of explicit behavioural characteristics that can be attributed to someone responsible for performing a set of tasks. 'The intended problem solvers must examine what kind of role they are expected to perform or adopt in a given situation and take full responsibility for their role clarification and actions. It is the duty of the methodology creators to explain the role expectations implied by their methodologies' (Jayaratna 1994). Within the framework, the intended problem solver could act as an adviser, an analyst, a consultant, a designer, an implementer or a facilitator. 'Role conflicts' are caused by the difference between the intended problem solver's role expectations and the expectations others may have of that role.

Models and framework: A framework is a static structure, which can be perceived as a metamodel showing the connections of a set of models. Modelling helps to develop one's reasoning abilities.

4.5.4 The Problem Solving Process

Methodology users need to ask how specific methodologies help them formulate and solve problems. Systems analysis, as normally understood in the context of systems development, is the study of an existing system taken 'as given'. Systems design is the process of improving an 'existing' system, the boundaries of which are not in question. This means systems analysis and design are based on a view of systems 'as given' and explains why critical enquiry into problem formulation has largely remained outside the domain of most systems development methodologies. According to the NIMSAD framework, systems could be more usefully viewed as 'human constructs' and when applying this view to the activities of systems analysis and design, Jayaratna (1994) refers to them as systemic analysis and systemic design.

Systemic analysis is applied when determining the problem situation in problem formulations 'a process of critical enquiry of situations with the use of the notion of 'systems'', while systems analysis is ' the study of an existing system' (Jayaratna 1994).

NIMSAD consists of eight phases, five of which are concerned with the problem formulation, a further two phases are concerned with solution design and the remaining phase is concerned with implementation. To some extent, failures or weaknesses in conducting one phase inevitably cause problems for subsequent phases.

4.5.5. Phase 1 - Problem formulation

4.5.5.1 Stage 1: Understanding the 'situation of concern'

The intended problem solvers understand and appreciate a dynamic situation by their dynamic 'mental constructs' taking a system as a 'human construct'. Activities carried out during this stage include; drawing a system boundary, determining 'what' information is required and selecting the most appropriate methodologies of investigation for information gathering.

The identification of a boundary on the situation helps to identify possible areas of interest. If intended problem solvers do not subject their 'mental constructs' to self critical examination they may end up accepting the boundaries as defined by the clients. Implicit boundaries may also be constructed subconsciously, thus we are unaware of why or where those boundaries have been drawn. Since the boundary determines the focus of the investigation and establishes the 'situation of concern', we need to examine whether we have identified relevant elements (persons, materials, activities, flows) in the situation for study. The use of epistemological notion of system enables the construction of boundaries not simply for practical reasons i.e. client definitions. For example: During the development of an order processing system the client is asked if this includes supplier payment, if a negative response is elicited then the use of epistemological notions could alert the client to the implications of leaving the supplier payments out of the proposed system, and establishes both intellectual reasons as well as practical reasons for the inclusion or otherwise of particular processes in the system. Unless we are prepared to operate at this level of

awareness (ability to examine why and how we select relevant elements), we may not recognise the effects of a chosen methodology on our boundary construction

4.5.5.2 Stage 2: Performing the diagnosis

Performing the diagnosis is the way to express the understanding of the 'situation of concern' from stage 1. Diagnosis should be explicitly identified as conceptual/logical and physical expressions, which the problem solvers should formulate according to the techniques in an applied methodology. In NIMSAD, the expressions at this stage create a basis for all subsequent problem-solving stages, particularly for the logical/conceptual and physical design (stages 6 and 7). The diagnosis serves three additional needs:

- To help methodology users identify gaps in knowledge or misunderstandings;
- To help methodology users to communicate - particularly with the client and problem owners, to derive agreed understanding, to clarify differences of perception and to explore the different 'world images' of the participants;
- To serve as the basis for understanding further activities in problem solving (Jayaratna 1994)

Techniques/tools provided by methodologies differ from one to another, for example, object models, object relational models etc. In addition, the expressions should be described and documented in different models i.e. diagnosis models 1 and 2 for logical and physical aspects of the situation. These models deal with explaining the 'current state' of the 'situation of concern'. Later on, the models can be used to re-examine the 'situation of concern'. It is important to note that the content within the boundary provides a description of the situation, while the outline shape of the diagram expresses an impression of the state of the situation (Jayaratna 1994). The diagnosis is a result of the interactions of two dynamic processes; the 'situation of concern' and the intended problem solver's 'mental constructs'. The problem solver constructs the expressions by applying their 'mental constructs' to the 'situation of concern', the expressions may not, therefore, accurately reflect the action world. There is, therefore, a potential risk of problem solvers solving the wrong problem. The diagnosis captures a 'current state' of the 'situation of concern', i.e. 'where are we now?

4.5.5.3 Stage 3: Defining the prognosis outline

During this stage the 'desired state' is emphasised, i.e. 'where do we want to be and why?' To understand 'why' requires a shape of the feature of the 'desired state', not the content of the prognosis model. The 'current state' is what the intended problem solvers find at stages 1 and 2, (what currently happens) whereas the 'desired state' is defined as a prognosis outline at this stage.

The intended problem solvers can only define an outline shape for the prognosis without content. This means that the shape could be considered in the design stages later on. Such a shape can be understood as a 'desired state with supporting reasons' (or 'why') once designs are finished. The difference between the two 'states' reflects the existence of a 'problem'. One of the most serious weaknesses of many methodologies is that they force their users, in an intellectual sense, to accept the 'desired state' of the client without question or critical examination. This can be problematic as the users may or may not formulate their expectations in a rigorous way, even though they may be knowledgeable about their own domain (Jayaratna 1994).

Most methodologies do not alert their users to the 'desired states', they remain private to the clients and problem owners. 'Desired states' of clients are analogous with the requirements specifications of the system, if the specifications are not questioned in order to determine a better understanding, i.e. Why is the system required? Who is going to use the system? What is the purpose of the system? Then effectively there is no problem to solve only a system to be developed. Failure to question the validity of the 'desired state' may lead to irrelevant solutions, expensive investments and implementations, and significant maintenance. Once a 'desired state' is accepted without question, for whatever reason, there is little or no opportunity to discover the 'real' problem issues. This could be compared to an architect and builder, an architect gives a builder a blueprint for a new building (requirements specification) the builder does not ask about the purpose of the building, why the building is required or who the building is for; he simply takes the blueprint and constructs the building.

4.5.5.4 Stage 4: Defining problems

This is the process of identifying and critically examining the absence of elements and/or the current arrangement of elements in the diagnosis model that prevent the 'current state' from changing to the 'desired state'. The result leads to the identification of a problem area, which can be described in a set of problem statements. 'What' and 'when' questions should be asked about this problem area.

> 'In information systems domains these statements cover not only gaps or failures of information, but also the associated roles, responsibilities, processes, functions, structures, cultures and relationships. .... In order to generate these statements, intended problem solvers need to ask questions of the type *what* and *why*, and not of the type *how* and *who* (Jayaratna 1994).'

4.5.5.5 Stage 5: Deriving notional systems.

Notional systems are those systems that need to be developed if the client organisation is to overcome the previously defined 'problems', thereby helping to transform the 'current state' to the 'desired state'. The description of notional systems features and their expected behaviour is called the 'systems specification' or the 'requirements specification ' document. If clients/problem owners are unable to express clearly and precisely their expectations there is no identifiable shape in the prognosis stage Fig 4.6.

When this happens, the situation could be considered to be an 'ill-structured' one (Jayaratna 1994). Explication of the situation of concern and the client's concern should be examined instead of accepting the client's notional system without a question.

> 'The object here is to make the intended problem solver examine the concerns of the clients and problem owners in depth, and to make explicit the legitimacy and validity of the prognosis outlines, the 'problems' and the notional systems of the clients and the problem owners,' (Jayaratna 1994).

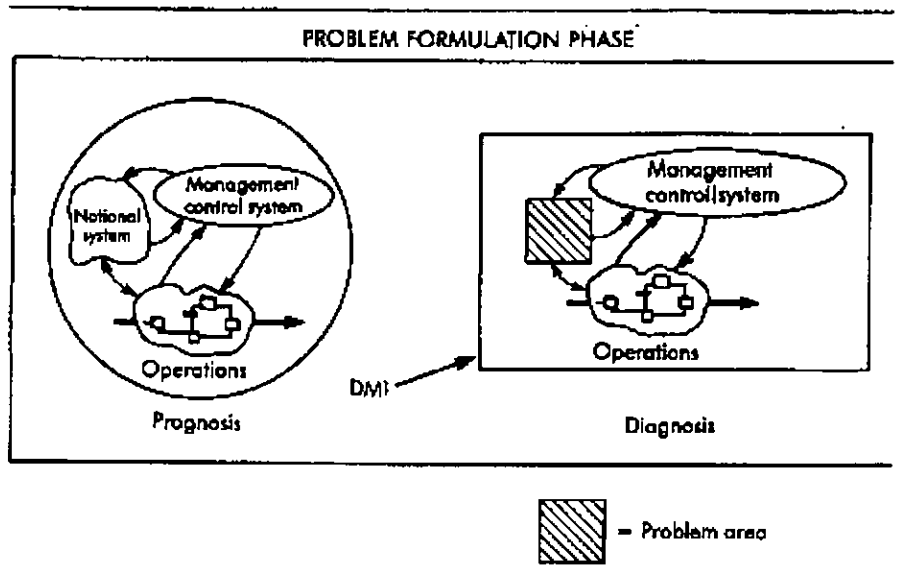


Figure 4.6 Deriving notional systems

(adapted from Jayaratna 1994)

4.5.6 Phase 2 - solution design

Solution design is defined by Jayaratna as 'the creative activity of constructing elements and organising them into integrated wholes, each capable of realising the notional system(s)' (Jayaratna 1994). Solution design includes both the conceptual/logical design and the physical design.

4.5.6.1 Stage 6: Performing the conceptual/logical design.

Methodology users have either to rely on implicit models or develop explicit logical/conceptual models in order to transform the notional system into a logical design. The outcome of this stage is the production of an agreed and acceptable logical design specification, which states the nature, and the function of the logical elements formulated from the diagnosis model. At this stage, modelling and design techniques included in the methodology are considered. The implementation of the logical design will differ with the methodology under consideration, i.e. some methodologies integrate logical design with physical design. For our purposes logical design is considered to be the realisation of the notional system and is seen as the basis of the physical design.

4.5.6.2 Stage 7: Performing the physical design.

Physical design can be considered as the deliberation and selection of 'ways and means' of realising the logical design. At this stage, the intended problem solvers

realise the features of the logical design by considering physical resources and constraints. Constraints may include economic, social, political and cultural environments, in which the proposed design models are expected to operate or perform, as well as input and output formats, data organisation, human computer interface, document standards etc. In this way a logical design is transformed into a physical model by addressing questions such as 'how' and 'whom'. Different design criteria for measuring reliability, accuracy, security and usability should be applied in order to determine the appropriateness of the physical design.

4.6 Phase 3: Design implementation.

4.6.1 Stage 8: Implementing the design.

In NIMSAD, design implementation is concerned with the realisation of the notional system within the context of a 'situation of concern' to fulfil the function of information systems development. The major aspects in implementation are: the 'situation of concern'; the models generated from previous phases of the problem solving process; and the need to discover a way of bridging the gap between design and realisation. According to Jayaratna (1994) these three aspects can be organised into five sets of tasks: environment preparation, systems development, changeover, strategy and planning, management and control.

> 'I consider the design implementation as a transformation from the 'thinking world' to the 'action world' at the end of the problem solving process. Collaboration among the intended problem solvers, the problem owners, stakeholders and clients, is needed to make the implementation a success. In particular, the significance of the collaboration has been recognised as a useful strategy for refining the diagnosis and prognosis models in face of continuous changes of the 'situation of concern'. Problem solvers need the support and the co-operation of clients and stakeholders to make the design realisable and contribute to organisation effectiveness (Jayaratna 1994).'

A complete problem solving process can now be shown diagrammatically Fig 4.7

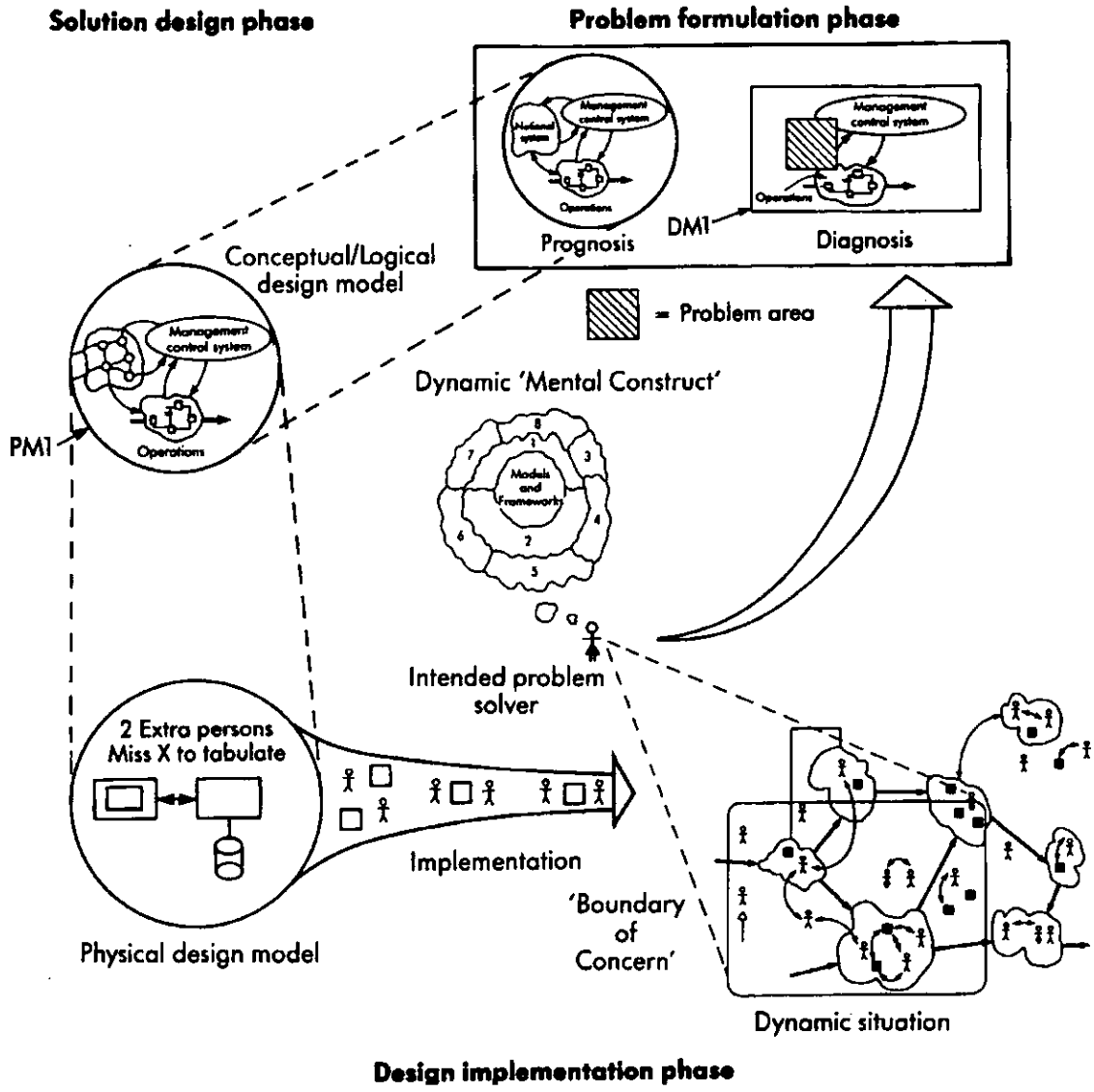


Figure 4.7 The Problem Solving Process

(Jayaratna 1994)

The intended problem solvers are located at the centre of the figure. They confront a dynamic situation, help to conceptualise the essential features, transform their structures by using models, and implement the solutions and examine the effects of the transformations. In order to achieve this they need to have communication and monitoring links with those involved in the situation and ensure that the methodological activities are applied appropriately.

4.7 Methodology evaluation

Methodology evaluation is regarded as the last and most important element in NIMSAD. Evaluation is carried out before, during and after the methodology

application and is conducted for the other three elements of the framework: the problem situation, the intended problem solver and the problem solving process. The evaluation element helps in the identification of the lessons learned from using the methodology. Jayaratna (1994) identifies a number of questions that could be used to aid the evaluation process. These questions are summarised in Table 4.1.

The salient points of table 4.1 being:
It is the methodology users who determine how the original methodology should be used, adapted/adopted and put into action, all of which influence the 'methodology' in use and the problem solving process. Any weakness in the methodology must be overcome by the intended problem solver's efforts or with the help of others. Jayaratna (1994) stresses that it is the methodology users that bring about the transformation of situations and not the methodology itself. The methodology can only help in the structuring of thinking and actions.

Table 4.1 Evaluation issues.

	The 'problem situation'	**The problem solver**	**The problem solving process**
Before intervention	To understand the reasons for the clients' concerns and expectations the 'desired states;	To understand what 'mental constructs' the problem solvers already have and distinguish those constructs from what the situation requires.	Why a methodology should be used; Selecting a methodology;
	To gain an understanding of issues and the clients' commitments to action;		
		To select a methodology in the situation	Changes made and why.
During intervention	To assess how the situation is unfolding;	To know how you (the methodology user) are performing	Predefined method in action -
	Are there any aspects that need managing?	What mistakes are being made?	To use a methodology for a particular problem solver in a particular problem situation.
	Are things going according to plan?	What skills/knowledge are necessary?	Learning about changes.
After intervention	To assess to what extent (1) the designed systems were implemented within the limits of resource, time and effort;	To assess the lessons learned and methodologies users' satisfaction with the methodologies and lessons for themselves	Lessons learned from experience of using a methodology. What modifications are necessary?
	(2) do systems do what they are supposed to do?		
	(3) Have the problems been resolved? - the relevance of the action system to the problem situation		
Overall	Resolution of client concerns in the 'problem situation'. Three levels of satisfaction:	To help the methodology users understand their strengths and weaknesses by using 'mental constructs'	The assessment of a methodology and the degree of assistance it provides in terms of models, concepts, techniques, tools etc.
	Model system has become action system;		
	Systems do what they are supposed to do;		
	System has transformed the situation.		

4.8 Justification of choice of Evaluation Framework

The choice of framework within which the selected object-oriented analysis methodologies were to be evaluated had to:

a) provide a frame of reference
b) be a problem-solving framework
c) have been used to evaluate methodologies
d) be available as a text.

The NIMSAD framework was chosen as it satisfies all the above criteria. NIMSAD defines an intellectual problem solving methodology as a goal-directed way of solving problems. Jayaratna (1994) states that the framework serves three purposes: understanding the area of problem-solving in general and of any nature; evaluating a methodology before, during and after use; and assisting in evaluating business/commercial practice. This provides a clear frame of reference within which to work.

NIMSAD framework has been used not only as a framework for evaluating problem-solving methodologies, at an academic level, but also within a business context. NIMSAD has been used by Jayaratna (1994) to evaluate three significantly different methodologies: Structured Analysis and Systems Specification (DeMarco 1979), ETHICS methodology ((Mumford (1983a) and (1983b)) and 'Soft' Systems Methodology ((Checkland 1981) and (Checkland and Howell 1998)). Jayaratna also refers to the use of NIMSAD framework within a business context.

Oates and Jayaratna (1995) used the NIMSAD framework to evaluate Yourdon Systems Method; while in Oates (1995) she used NIMSAD to evaluate NIMSAD as a methodology, she concludes that NIMSAD provides a useful conceptual framework within which to evaluate methodologies that solve problems. Barker (1997) used NIMSAD to critically evaluate Jackson Structured Design (JSD) but he draws no conclusions on the usefulness or otherwise of NIMSAD framework. Zhang (1999) used NIMSAD to critically evaluate how user participation is considered in Object Oriented Software Engineering (Jacobson et al. 1998), Object Modelling Technique (Rumbaugh et al. 1991) and Object Models, Strategies, Patterns and Applications (Coad et al. 1997).

The NIMSAD framework as a problem-solving framework was useful throughout my research as it helped to focus the mind on the important issues being raised by the various methodologies. However I also found that NIMSAD did not have the flexibility for semantic considerations that are seen as central to object identification.

The structure and concepts of NIMSAD framework will be used to evaluate three object-oriented analysis methodologies in Chapters Five to Seven.

4.9 Summary

This chapter considered different approaches for comparing/evaluating methodologies. Approaches considered included, taxonomies, visual comparisons, descriptive comparisons, structured thinking and frameworks. Avison and Fitzgerald's (1995) framework, Multiview (1990) and NIMSAD (1994) framework were considered.

NIMSAD framework was chosen, as an appropriate problem-solving framework, as the identification and formulation of objects is perceived as a problem in object-oriented analysis. The choice of NIMSAD was also justified.

Part two comprises the next four chapters and uses the NIMSAD framework to critically evaluate the selected methodologies (chapters five, six and seven); Coad and Yourdon's OOA (1991), Rumbaugh et al's OMT (1991) and Jacobson et al's OOSE (1998). Each chapter has an emphasis on how the methodology assists the methodology user in identifying objects. Chapter eight explores how objects are identified in practice; the chapter represents the results of a number of semi-formal interviews with a small number of colleagues (academics) and practitioners who use object-oriented analysis methodologies as part of their teaching or normal working practices.

Chapter Five: Critical Evaluation of Object Identification in Object Oriented Analysis (OOA).

5.1 Introduction to Object Oriented Analysis (OOA)

According to Coad and Yourdon (1991) Object Oriented Analysis (OOA) is based on concepts that we learned in kindergarten: objects and attributes, wholes and parts, classes and members, because we instinctively use these classifications and concepts Coad and Yourdon assert that this is a 'natural ' way to approach analysis.

In essence OOA Coad and Yourdon (1991) was developed to create object-oriented systems models. It produces a five-layer model of the system, identifying objects, structures, subjects, attributes and services. It consists of five activities: -

- ***Define Objects and Classes*** These include Structures, events, roles, other systems, devices, and organisational procedures.
- ***Define Structures*** The relationships between classes are represented as either general-to-specific or whole-to-part structures.
- ***Define Subjects*** Subject areas are identified in the top-level objects and refined to minimise interdependencies.
- ***Define Attributes*** The objects are examined in terms of their characteristics and associative relationships.
- ***Develop Services*** All the Services performed by classes and objects are identified.

However the activities are not viewed as sequential steps rather as a looping construct where a Class-&-Object may be identified, then its Structure, Subjects, Attributes and Services. Further Class-&-Objects are then identified and the process repeated. In this way the documentation is constantly being referred to and clarification sought from domain experts, managers, etc. until the entire problem domain has been considered. At any point during the analysis process one could have Class-&-Objects that are specified by name only; Class-&-Objects with Structures; Class-&-Objects with Structures, Attributes and Services specified; or Subjects (sub-domains) that are completely specified describing all associated Class-&-Objects, Structures, Attributes and Services. If a large or complex problem domain is being considered it may then be very difficult to determine when all aspects have been considered and therefore when the analysis could be said to be complete.

OOA has a strong focus on techniques - in most situations it would be feasible for two or more technically competent OOA methodology users to derive different object models. It could be argued that because of the cyclic nature of the methodology objects that are missed initially may be uncovered at a later stage and, due to its relative immaturity, analysts may be still developing their techniques and determining which problem situations are most applicable. Coad and Yourdon (1991) include application examples, using OOA that are stated as being applicable to most businesses and industries. These include: point-of-sale application; warehousing application; order processing application; soft real-time application and hard real-time application. OOA is applied to application examples, within the text, with objects identified from users' accounts of their business and object classes named by following the users' vocabulary.

5.2 Evaluation of OOA focused on Object identification

The rest of this chapter uses the NIMSAD framework to examine what OOA has to say about the problem situation, the intended problem solver and the problem solving process with an emphasis on object identification. Object identification is the starting point of any object analysis or object modelling methodology and is concentrated in the activities, defined by Coad and Yourdon (1991) as; define objects and classes, define structures and define attributes. Evaluation using the NIMSAD framework will concentrate on these activities.

5.2.1 Element 1:The 'problem situation'

NIMSAD alerts the problem solver to consider wider issues surrounding the use of a methodology – for example the size and the nature of the organisation, the identity and belief of the people within it (namely the client and the stakeholders), the behavioural patterns of the stakeholders, and their perceptions of the current situation. Therefore it is essential that a methodology is able to explore and consider a given 'problem situation'.

The 'problem situation' is drawn from an organisation and its business and is described by the members of the organisation. In order to understand the problem

domain: 'why this system' and 'why now' become necessary questions to ask as it is important to get domain experts to agree upon a formulated system purpose and features. What the organisation needs from this application system and why the system is required are not considered by the methodology nor are the implications of ignoring such questions.

The problem situation is viewed through the filter of the problem solver, who in turn may have compiled this view through discussions with the client. This can be problematic as there is a risk that the 'problem solver' may conceptually reduce the client or client organisation to the status of 'object' and this may lead the problem solver to misinterpret the client's view on the nature of the problem situation, and the required system which is intended to provide a solution.

The OOA methodology provides a basis for an initial expression of the system context, however, a diagram, drawn by a systems analyst making a technical decision, does not define the context. This indicates that Coad and Yourdon have adopted an ontological notion of systems. The boundary of the 'problem situation' is not explored although Coad and Yourdon (1991) acknowledge that:

> 'clients, managers, analysts, competitors, government regulators and standard bearers all affect the system context over time. System context is an indication of how much of the problem domain will be embraced by the automated system, what data will be held over time, and how much processing sophistication will be included'.

If disagreement exists amongst stakeholders as to the nature and scope of the 'problem situation' then OOA will not alert the user to this, nor will it provide any means of resolving conflicts.

Coad and Yourdon (1991) refer to ''system's responsibilities' as an arrangement of things accountable for, related together as a whole'. Used in this way 'system's responsibilities' are considered as the pre-specified boundaries of the problem situation with 'system' being used in an ontological sense, i.e. analysis process assumes that the world is composed of systems and sub-systems. 'Systems' and the problem situation, are taken as given without question, as is the boundary of the problem situation. This is at variance with a previous statement by Coad and Yourdon

'approaches to analysis are 'thinking tools' used to help in the formulation of requirements.' The 'thinking tools' referred to are the approaches considered by Coad and Yourdon during the development of OOA; functional decomposition, data flow modelling, information modelling and object-oriented programming. Coad and Yourdon are talking not about models for understanding a 'situation' but models to organise a solution.

The implication of this is that the analysts do not question the motivations and justifications for the 'system' nor do they question the relevance or otherwise of the boundaries perceived by the clients. Effectively any difficulties with the final solution can be claimed to be outside the system boundaries. This creates a 'safe' problem for which there could be a desired solution already in existence. For example: in an order processing system, the client states that supplier information is outside the problem domain. By accepting this statement and not considering the implications from the business perspective any problems related to supplier information in the implemented solution can then be identified as being outside the originally stated problem domain and therefore not considered. A more realistic approach would be to make the client aware of problematic issues that may arise relating to supplier information and only when all implications have been discussed should a decision be made.

5.2.2 Element 2: The intended problem solver

NIMSAD focuses on the role of the problem solver's 'mental constructs' that help to understand the complexity of the 'problem situation'. A methodology is a 'brain structuring' mechanism, and if it is to be used successfully, then its user must have the necessary characteristics and preparation. Additionally

> 'it helps us to examine whether the methodology alerts us to the need for developing our 'mental constructs' to a desirable level in order to apply the methodology successfully' (Jayaratna and Oates 1995).

The text (Coad and Yourdon 1991) is written for a specific target audience, that is those currently employed in an analysis capacity.

> ' We have aimed this book at the practising systems analyst, the person who has to tackle the real-world systems development projects every

day... the strategy used to identify objects comes from practice and experience in the field'.

Coad and Yourdon (1991) assume a fundamental understanding of computer technology and systems analysis concepts, experience of data flow diagrams and entity relationship diagrams as pre-requisites of object-oriented analysis. The methodology creators state that OOA constructs come from Entity Relationship Modelling, Semantic Data Modelling, Object-Oriented Programming and Knowledge Based Systems. From the perspective of data analysis and database design, the underlying logic for this statement needs to be challenged. Entity Relationship (ER) diagrams can be seen as the end result of the data analysis process and are therefore the first step in a design strategy. If they inform an analysis method then the analysis method is design led and the ER diagrams are part of a solution-defining problem. Coad and Yourdon (1991) are attempting to shift the design principles used into analysis as there is little or no real understanding of the analysis process from the data perspective, their previous experience being from a task or process oriented approach (Yourdon 1989).

Coad and Yourdon (1991) identify problem domain understanding as one of the major difficulties associated with traditional systems analysis and imply that an object-oriented approach will help with problem domain understanding. Object Oriented Analysis is identified as 'the challenge of understanding the problem domain'. Structured analysis methods are considered to be too limiting to be used in an object-oriented environment and therefore object-oriented analysis techniques need to be developed. Coad and Yourdon (1991) state

> 'analysis is the process of extracting the 'needs' of a system - what the system must do to satisfy the client, not how the system will be implemented.'

However, there is an underlying assumption about the design method and the coding language to be used. This also determines to a large extent the way the problem is initially perceived. This is confirmed by the following statement from the methodology creators

> 'the underlying concepts of the object oriented approach have had a decade to mature and attention has shifted from issues of coding, to issues of design, to issues of analysis. The underlying technology for building

> systems has become more powerful. The way of thinking about systems analysis is influenced by preconceived ideas of how we would design a system to meet its requirements ' (Coad and Yourdon 1991).

Coad and Yourdon (1991) state,

> 'an analyst needs to communicate throughout the analysis effort. He needs to communicate in order to extract information about the problem domain and requirements from the client. He may also need help in steering the client away from requirements that cannot be met within budget and schedule constraints.'

OOA makes the assumption that all participating groups can agree a requirements list, but the methodology does not offer any techniques for facilitating the resolution of such issues. The methodology does not attempt to highlight the ethical/moral issues or other elements of a 'mental construct' that need to be considered. In common with several other methodologies, OOA projects the problem solver in the role of defining *how* the system will fulfil the requirements of the specification, and the client in the role of defining *what* those requirements are.

OOA does not alert its users to their 'mental constructs', because of this lack of guidance as to the epistemological nature of the process and the fact that OOA does not provide models to identify the reasons for potential 'clashes' between differing opinions, it is possible that the methodology user may produce solutions which are inappropriate to the situation context.

In the organisational environment, object modellers as methodology users look for objects and organise object classes according to the information they receive during their initial observations, discussions and investigations. These discussions and investigations involve the future system users, referred to as 'domain experts' in OOA. These people have experience of working in a specific area of concern. This assumes that the information about the problem domain is objective and waiting to be discovered. There is no acknowledgement that the problem solvers have a responsibility for interpreting the information. The interpretation placed on the information by the problem solver influences the client / situation and cannot be seen as neutral. OOA does not alert the intended problem solver to this.

Jayaratna (1994) suggests that a useful way to gain this knowledge is to take the organisations as purposeful systems. If the intended problem solvers assume that they have the required subject/organisational knowledge they are likely to allow their own perceptions to 'colour' the requirements, that is they will not critically question the 'desired state' of the client. This also supports the argument by Jayaratna (1994) that many methodologies force their users to accept, particularly in an intellectual sense, the 'desired state' of the client without critical examination or question.

A methodology is meant to help people to develop their understanding and structure their thinking; OOA identifies the stages to be considered and gives examples, in the text, as to how the strategies can be applied in 'action world' situations. The methodology user is not given any other guidance on how to develop their experience, other than work through the given examples and then apply them in the target problem domain.

Coad and Yourdon (1991) imply that by taking an object-oriented approach to analysis a greater understanding of the complexity of the problem domain will be achieved. This must depend on the competence and expertise of the analyst; an experienced person is just as likely to miss the complexity of the problem domain using this methodology as with any other methodology, as is an inexperienced analyst. OO concepts only help to focus on formal operations performed by or on data.

According to the NIMSAD framework, system developers 'mental constructs' function in a dynamic situation and we should be aware of the characteristics that constitute our 'mental constructs'. Experience and knowledge are identified as helping to 'form explicit models for structuring our understanding'. Experience comes from solving earlier problems. The more experience we have of a particular problem domain or working environment the easier it may be for us to assess similar situations. Previous experience is used while using a new methodology and should be recognised as part of the system developer's 'mental constructs'.

OOA talks about the experience element of the framework but it does not alert its users to their 'mental constructs' nor does it 'help them to examine the high level of

preparation they should have in order to use the methodology successfully' (Jayaratna 1994). It is evident that OOA assumes that the use of its techniques will yield the same results irrespective of the nature of the 'mental constructs' of its users.

The ability, of the intended problem solver, to identify appropriate objects is implicit within OOA. Objects are seen as abstractions of the real world that provide a focus on significant aspects of the problem domain and the system's responsibilities. However, OOA does not provide explicit ways of performing this abstraction. Coad and Yourdon state

> 'the principle (of data abstraction) can be the basis for organisation of thinking and of specification of a system's responsibilities.'

The examples, given in the text, assume that the intended problem solver is familiar with the problem domain and can identify any objects that are appropriate in the given situation. The methodology does not alert the methodology users to the effect that culture; beliefs, shared understanding and knowledge can have on choice of relevant objects nor does it alert the methodology user to the importance of having a shared understanding of the terminology used in the problem domain. Users need to check the meanings incorporated into the specifications in order that results of the analysis can be verified. This can be difficult and users are unlikely to be able to make an adequate check on the meaning of words. The risk arising from this situation is that the analysts may wrongly understand the user's requirements. Inaccurate requirements may be sought, though represented in a correct syntax.

An emphasis on semantics, as part of the analysis phase, is a possible solution, in addition a way of preserving and clarifying meaning in requirements analysis should be considered.

5.2.3 Element 3: The problem solving process

The problem solving process has three essential phases: problem formulation, solution design and design implementation (Jayaratna 1994).

The NIMSAD framework sub-divides these phases into eight systemically based stages. By analysing the situation in a systemic way, it should be possible to make the transition from 'current state' to 'desired state'. Systemic design can then be utilised

to obtain models of notional systems, any or all of which should be able to facilitate the desired notional system(s).

5.2.3.1 Stage (i) Understanding the 'Situation of concern'

OOA does not alert its users to the conceptual construction of boundaries or identification of a 'situation of concern'. Boundary construction is either trial and error or predefined by the client. Questioning 'why' the system is required corresponds to understanding the 'boundary of concern' within the NIMSAD framework. Boundary constructions are based on the data that are meaningful to the OOA user, data that is considered by the OOA user to be relevant to the requirements list and that data which the client insists on as being relevant for their systems.

The OOA methodology creators state that

> 'structures focus the attention of the analyst and problem domain experts on the complexity of multiple Class-&-Objects using Structures the analysts push the edges of the system's responsibilities within a domain, uncovering additional Class-&-Objects (implicit in the requesting document that might otherwise be missed)'.

The initial system boundary is 'taken as given' by the client, the previous statement reveals the implicit recognition of the artificial nature of the boundary construction. However, the methodology users' 'mental constructs' that determine the nature of the boundary construction is not explained. Consideration of objects in structure terms i.e. generalisations, specialisation, inheritance and whole-part structures, may indicate that there are interface objects or control objects that were not apparent from documentation or discussion. These are the objects that are seen as 'pushing the edges of the system's responsibilities'.

5.2.3.1.1 Investigation models and techniques

The OOA methodology authors state that information is captured through interviews with stakeholders, and interpretation of the requirements specification as prepared by the client. How these interviews are structured in order to elicit the required information is not stated. As a result stakeholders may not elucidate their opinions and concerns about the current systems, or they may make assumptions because something seems so obvious to them that they believe it will be obvious to anyone else, and in

doing so not pass important information to the problem solver. Moreover the problem solver may not become aware of problems associated with the data, or the various opportunities that may arise through its use.

The effects of differing interpretations by stakeholders and problem solver with regard to the situation may not be taken into account, resulting in a system which may operate in a way fundamentally different from established working practice within the organisation, thus causing possible discontent and resistance to change amongst the workers.

Most people involved in the IT sector now accept that as technology changes and more and more clients/customers realise the strength of the technology they expect applications to be able to instantly react to those changes - be they changes in requirements or technology changes. A thorough initial analysis should be able to identify those areas that are liable to change as well as the core areas of the business / problem situation that are unlikely to change. It is assumed that the 'true requirements' referred to by Coad and Yourdon (1991) are those central to the business/problem situation that are unlikely to change over time, or as a result of technology change, i.e. the way in which data is used may change but not the data itself. This assumes that the data is useable regardless of the situation.

Encapsulation offers a technique for making systems resilient to changes in implementation caused by changes in data format. For example: British Telecom recently changed the format of many telephone numbers. Software written in an object-oriented style suffers less from the effects of such change since the internal implementation of 'phone-no' is hidden and the only changes that occur should be within this object, with other parts of the system not needing major changes (Graham 1995).

Coad and Yourdon (1991) state:

> 'systems analysis usually begins with a requesting document (from the client, or perhaps from the marketing division) and a series of discussions. In any case the audience includes, problem domain experts, developers and

possibly other interested parties (auditors, contracting officers, etc.) who may need to understand and agree with the proposed set of requirements'.

By taking a requesting document as a start point, the methodology creators are accepting that the client has identified a problem and an outline solution may also have been proposed. The inclusion of many of the stated stakeholders implies that there are many political, financial and organisational considerations that are not acknowledged by the client and the methodology user may not be made aware of.

Differences in personal experiences, values and expectation can create difficulties in understanding. As soon as there is a problem in understanding, people come back to a base level, and from there they can expand their common knowledge and reach their new agreement. (Dik 1989) refers to personal possession of knowledge and experience as 'pragmatic' information. If two participants in a conversation are from the same cultural community, they have a large amount of shared pragmatic information. Incorrect assumptions of the pragmatic information of the opposite side can lead the conversation astray. Therefore it is important for a speaker to acknowledge the assumptions shared with the hearer before communication occurs. Apart from the shared knowledge, the context where the communication takes place is important to the pragmatic affects.

Coad and Yourdon (1991) state,

> 'an analyst needs to communicate throughout the analysis effort. He needs to communicate in order to extract information about the problem domain and requirements from the client'.

The OOA methodology gives little advice as to relevant or appropriate methods of investigation, although it is stated that interviews and questionnaires may be employed. Observation is identified as an important investigation method as the intended problem solver 'strives to get an intuitive feel for the challenges and frustrations your client faces.'

The lack of attention to methods of investigation has serious implications for systems development. It means that OOA users may not be alerted to

'the different nature of problems, opportunities, or the effect that methods of investigation may have on the nature, content and subsequent interpretations attributed to data. The factors that influence these interpretations are interpersonal relationships, influence and authority of individuals, personal motives and prejudices. Given that the methodology is not alerting the methodology users these will have to be addressed by the methodology users using their own skills and knowledge '(Jayaratna 1994).

5.2.3.2 Stage (ii) Performing the Diagnosis

Diagnosis is defined by Jayaratna (1994) as 'an expression of our understanding of a situation of concern'. The methodology should therefore help the user to understand the nature and scope of the situation, and any issues that surround them, in an explicit and clear fashion. There are two separate facets to this determination of issues – firstly, the methodology should provide clear help in outlining the 'problem situation', and secondly, it should help the user to clarify the reasoning that underlies the 'situation of concern'. OOA attempts to gain an understanding of the current 'problem situation' by 'identifying' objects that are relevant to, or have an effect upon the problem situation.

Coad and Yourdon (1991) identify strategies for determining the requirements based on notation, where to look, what to look for and what to consider and challenge. These strategies however do not question the 'mental constructs' of the intended problem solver nor do they question the clients' perception of the problem situation or their motivations.

OOA makes a useful contribution to performing the diagnosis by providing techniques for defining objects, classes, structures and subjects. The activities in OOA are based on identifying and specifying details of Class-&-Objects. A class is the generalised grouping and objects are individual instances of a class. For example: Class Person, Object Helen: Helen can be considered to be an individual representation of a PERSON, she has certain characteristics/attributes that are common to all PERSON objects, that are of interest within the problem situation. Object Alan could be added to the PERSON class in which case there would be certain characteristics that are common to both Helen and Alan that are inherited from the PERSON class. There are

likely to be other attributes associated with either Helen or Alan that are unique to them which may or may not be relevant to the problem situation. Only attributes relevant to the problem situation are included.

Coad and Yourdon (1991) state 'strategy (for identification of objects) comes from practice and experience in the field'. The OOA methodology creators use approaches that they are familiar with, without understanding exactly what principles are being applied and why. The initial starting point for identifying objects, according to Coad and Yourdon (1991), is to identify the nouns within the discussion transcripts and the documents provided. These potential objects, identified by nouns, are related mainly to the data requirements of the stated problem situation. The approach taken by the OOA methodology creators indicates that the methodology user should determine what they can about the problem domain and problem situation by asking questions. Clients, users and experts should all be consulted as should any documents; invoices, receipts, reports etc. This is no different to the approach taken by structured systems analysis methodologies.

The problem situation is then considered, against the following set of criteria; generalisation and specialisation structures; other systems that interact with the information system under consideration; peripheral devices; events or time based processes; roles played; operational procedures; other sites and organisational units that the information system may need to interact with. This is a mechanistic approach to validating objects that have been identified, in order to justify their inclusion as potential objects. The relevance of the selected objects is left to judgement and experience of the methodology user.

The methodology is not assisting the methodology user to identify appropriate strategies, and gives little insight into how objects can be identified. Textual analysis is used initially to identify potential objects. This assumes that methodology users, clients and other stakeholders consulted have a shared understanding of both the terminology and semantics used within the organisation. Having achieved this classification theory is then applied in order to identify appropriate Class & Objects.

Coad and Yourdon (1991) state that

> 'Analysis notation and strategy can build on three constantly employed methods of organisation. In practice applying 'Objects and Attributes', 'whole and parts', and 'classes, members and distinguishing between them' provides significantly greater insight into a problem domain and a system's responsibilities than applying only 'objects and attributes''.

This is based on classification theory and the organisation of knowledge. OOA does not alert the methodology user to this nor does it offer any assistance, relying on the methodology user(s) experience and knowledge to determine appropriate classifications.

Experience is an invaluable source for developing knowledge and skill, and helps to form implicit models for structuring our understanding of situations. These models dictate what information to seek or what action to take in a given situation. The more experience we have of a particular working environment, the easier it may be for us to assess similar situations. Experience makes us more confident and enables us to assume 'expert' status. However, while experience-based models may reduce time, provide a range of easy and 'obvious' solutions and help to develop confidence, the same models may prevent us from exploring new ideas or becoming effective listeners to others' ideas (Jayaratna 1994).

5.2.3.3 Stage (iii) Defining the prognosis outline

By 'defining the prognosis outline' NIMSAD means that the methodology should be able to facilitate the identification of a clear defined 'desired state' in relation to the 'situation of concern'. Jayaratna (1994) expresses this as defining 'where we want to be and why?'

As OOA does not make its users aware of how to perform the diagnosis it cannot help them to understand or, more significantly, question the rationale for the clients' 'desired states'. OOA relies on the clients being clear about the rationale for their requirements. It does not offer any steps for examining the validity of these expectations.

5.2.3.4 Stage (iv) Defining problems

NIMSAD uses this stage to investigate what the methodology can offer in terms of providing an answer to the question ' What are the problems which currently prevent the transformation of the 'problem situation' from its 'current state' into the desired state'?

As OOA does not concern itself with the clients 'desired states' (prognosis outline) it has no means of defining problems either. Problems are formed at a client level. Some of the changes to requirements are legitimate, as they are made in response to changes in the environment, for example: legislation, customer preferences etc. However, many are due to weaknesses in clients' problem formulation processes and their methodologies. Many clients do not spend sufficient time and effort on explicit reasoning about their requirements. In other words, clients very often do not use methodologies for arriving at their requirements. OOA is not dealing with the relevance of problems but the construction of solutions to a requirement.

In order for the problem solver to realise the existence and severity of these problem areas, they are reliant upon the client having a sufficiently wide and in-depth understanding of the situation and the issues surrounding it, and the ability to explain these issues explicitly. In the absence of such help, the problem solver needs to rely on their experience to understand the nature of the problem.

5.2.3.5 Stage (v) Deriving Notional Systems

NIMSAD defines notional systems as

> 'those systems which need to be developed if the client organisation is to overcome the previously defined 'problems', thereby helping to transform the 'current state' to the 'desired state' (Jayaratna 1994).

This means identifying systems using the 'mental construct', which, if designed and implemented, would eradicate one or more of the perceived problems with the current system.

As OOA does not help its users to participate in the definition of 'desired states' it cannot define problems, nor can it arrive at notional systems. The concern of the methodology is to record the features of the notional system(s) expressed by the client

in a concise form in order to facilitate its design tasks. In practice, user requirements guide the whole methodology process and they are taken into account at the beginning of the project. The methodology helps to explore the definition of elements that can be expressed in class, object and structure form.

5.2.3.6 Stage (vi) Performing Conceptual / Logical Design

The conceptual and logical design of a system is intended to formalise the structures, roles, tasks, functions, information and attitudes of the notional system(s) into a form that represents a clear, logical method of achieving the ideas identified in the notional system(s).

OOA recognises that roles played by objects, structures and interfaces, both with users and with other systems are necessary. Techniques are available within the methodology to assist the methodology user in capturing the relevant data.

> 'Requirements also include interfaces that the software must deal with, environments that software must accommodate and any other applicable design constraints' (Coad and Yourdon 1991).

However, any judgement is based on the methodology user's 'mental constructs' i.e. past experiences and subjective interpretation of the system. The methodology relies on its users' considerable experience for gaining an impression of the state of the situation.

There is no distinction between logical and physical aspects of modelling. The diagram sets are modified and expanded as further Class-&-Objects are identified / discovered. It is the methodology user's knowledge of the problem domain that determines which Class-&-Objects are relevant or are included in the model. Coad and Yourdon (1991) state that the identification of objects and classes matches the technical representation of a system more closely to the conceptual view of the real world. However they do not expand on this statement, they imply that the development of the object model closely resembles the conceptual view of the real world and as the object model can be implemented as object and class in an object-oriented programming language, the technical representation therefore matches the conceptual model. This is a solution driven and opportunity seeking approach and it raises a number of issues. As there is not one conceptual view of the world it must be

asked 'Whose view of the 'real world'?' By using Soft System Methodology (Checkland and Howell (1998) and Mumford (1983a)) an accommodating view of the 'action world' may be derived but in the absence of any discussion by Coad and Yourdon (1991) are we correct in assuming that the 'action world' is the methodology user's view of the consensus viewpoints of all the stakeholders? The emphasis is not on matching the technical requirements, but rather finding out what problems there are in the 'action world'. In fact this is mixing up current/desired states and the use of design concepts to derive desired states (Jayaratna 1994).

5.2.3.7 Stage (vii) Performing the Physical Design

The physical design of a system is the process of transforming the logical / conceptual design into a form suitable for implementation (given a set of identified constraints).

The boundary domain of the design is set by the client's requirements. In OOA this is referred to as the problem domain. The domain is established in consultation with the client. In OOA the model is progressively developed and modified as new Class-&-Objects are identified, the classes are then further refined by considering generalisation /specialisation of the classes and whole part structures. As a result there is no clear distinction between the analysis phase and the design phase, nor is there a distinction between the conceptual design and the physical design. As OOA can be developed cyclically it is possible to have Class-&-Objects in the same model at different stages of development.

Coad and Yourdon (1991) state,

> 'OOA maps problem domain and systems responsibility directly into a model instead of indirect mapping from a problem domain to function/sub function or problem domain to flows and bubbles',

having initially developed traditional analysis methods using flows and bubbles it appears that they are abandoning their previous convictions. It could be argued that by analysing the data and selectively applying the rules of first normal form then one also has a direct mapping between the problem domain and system's boundaries and a design model.

5.2.3.8 Stage (viii) Implementing the Design

The implementation of a system is the realisation of the notional system(s) within the 'situation of concern' (Jayaratna and Oates 1995).

As OOA does not cover implementation issues it cannot be evaluated in practice. Therefore OOA cannot be considered to be a problem solving methodology rather it is a design methodology. As a method we can examine what it is good at, in addition to a way of organising data into a new user oriented way rather a process oriented way.

5.2.4 Element 4: Evaluation

'It is the evaluation which helps us to measure the effectiveness of the problem solving process and the problem solver in the 'problem situation' – unless the element is considered there is no way of establishing that the 'problems ' have been successfully resolved' (Jayaratna and Oates 1995).

Evaluation of OOA occurs prior to intervention. The methodology creators acknowledge that 'analysis means extracting the 'needs' of a system - what the system must do to satisfy the client, not how the system will be implemented'. A discussion on the suitability of application areas to an object-oriented approach also considers the methodology prior to intervention.

Object-oriented methodologies are described as a new paradigm, the OOA methodology creators do raise some questions to alert the methodology users to the applicability and suitability of the methodology. For example: Is this the time to start using OOA? Is the object-oriented paradigm mature? Is there a good object-oriented implementation technology? Is the organisation sophisticated enough? Is the organisation building systems that will exploit object-oriented techniques?

There is no evaluation within OOA during and after the application of the methodology.

5.3 Summary

This chapter began with a brief introduction to OOA and then evaluated OOA using the NIMSAD framework with an emphasis on object identification. The main findings of this evaluation are as follows.

OOA was developed to create object-oriented systems models and is based on classification theory and the organisation of knowledge: objects and attributes, wholes and parts, classes and members. OOA does not alert the methodology user to this nor does it offer any assistance on how to apply this theory. Instead OOA relies on the methodology users experience and knowledge to determine appropriate classifications. OOA does not provide explicit ways of performing this abstraction. The examples given in the text, assume that the intended problem solver is familiar with the problem domain and can identify appropriate objects in the given situation.

OOA does not alert the methodology users to the effect that culture; beliefs, shared understanding and knowledge can have on choice of relevant objects nor does it alert the methodology user to the importance of having a shared understanding of the terminology used in the problem domain.

The initial starting point for identifying objects, in OOA, is to identify the nouns within the discussion transcripts and the documents provided. The approach taken by the OOA methodology creators indicates that the methodology user should determine what they can about the problem domain and problem situation by asking questions. Clients, users and other stakeholders should be consulted as should any documents; invoices, receipts, reports etc. This is no different to the approach taken by structured systems analysis methodologies.

This assumes that methodology users, clients and other stakeholders have a shared understanding of both the terminology and semantics used within the problem domain. Experience is implicitly assumed to be gained by working through the given examples and applying the strategies to the target problem domain. OOA relies on the methodology users considerable experience to gain an impression of the state of the situation.

The OOA methodology takes an ontological view of the problem situation in so far as the boundary of the problem situation is not explored but taken as given. The rationale for the requirements is not considered by the OOA methodology as a result of this there is no clear 'desired state'. To this end OOA is not dealing with the relevance of the problems but the construction of solutions to a requirement.

Methodology evaluation is of considerable importance to organisations and the NIMSAD framework is useful in that evaluation. OOA, however, limits the evaluation of the methodology to that conducted prior to intervention.

Chapter Six: Critical Evaluation of Object Identification in Object Modelling Technique

This chapter gives a brief overview of Rumbaugh et al's. (1991) Object Modelling Technique (OMT). OMT is then evaluated using NIMSAD framework. The evaluation concentrates on how OMT helps the methodology user to determine what objects to use when modelling the problem area.

6.1 Introduction to Object Modelling Technique (OMT)

Object Modelling Technique (OMT) (Rumbaugh et al. 1991) was developed to cover both the analysis and design phases of the software development life cycle. It is based on entity relationship modelling, which is extended to model classes, inheritance and behaviour. Three models are developed that cover different aspects of a software system: data, control and function. In OMT, the development process is divided into steps according to a development life cycle. Analysis in OMT is concerned with understanding and modelling the application and the domain within which it will operate. The initial input to the analysis phase is a problem statement that describes the problem to be solved and provides a conceptual overview of the proposed system. Systems analysts build the object model and other models based on the problem statement. OMT has four phases: analysis, system design, object design and implementation. The first step is to generate a 'problem statement'; this is expressed as three models of the system, representing objects, behaviour and transformations.

The procedure for creating these models is:

- **Object Model** focuses on the problem domain and the identification of relevant object classes from the application domain.
  Identify objects and classes
  Prepare a data dictionary
  Identify associations between objects
  Identify attributes of objects and links
  Use inheritance to organise and simplify object classes
  Verify that access paths exist for likely queries
  Iterate and refine the model
  Group classes into modules

- **Dynamic Model** modelling the application in terms of users and their actions.
  Prepare scenarios of typical interaction sequences
  Identify events between objects
  Prepare an event trace for each scenario

Build a state diagram

- **Functional Model** concerns all object categories, covering both static and dynamic constraints alike.
  Identify input and output values
  Build data flow diagrams to show functional dependencies
  Describe functions
  Identify constraints
  Specify optimisation criteria

The principle tools used by the methodology are:-

- object diagrams and data dictionaries for object modelling;
- scenarios, event traces and state diagrams for dynamic modelling and
- data flow diagrams for functional modelling.

An object model describes the possible patterns of objects, attributes and links that can exist in a system. The attribute values and links held by an object are called its state. Over time the objects stimulate each other, resulting in a series of changes to their states. An individual stimulus from one object to another is an event.

The dynamic model consists of multiple state diagrams; one state diagram for each class with important dynamic behaviour, and shows the pattern of activity for an entire system. Each state machine executes concurrently and can change state independently. The state diagrams for the various classes combine into a single dynamic model via shared events. An event is something that happens at a point in time. An event has no duration. One event can logically precede or follow another, or the two events may be unrelated.

The functional model consists mainly of data flow diagrams (DFD), which specify the meaning of operations and constraints. A DFD shows the functional relationships of the values computed by a system, including input values, output values and internal data stores. A data flow diagram shows the flow of data values from their sources in objects through processes that transform them to their destinations in other objects. This is the same model used by structures analysis methodologies, which show the flow of data values from their source in entities through processes. This implies that for Rumbaugh et al there is no significant difference between an object and an entity as they use the terms interchangeably.

The functional model specifies what happens, the dynamic model specifies when it happens and the object model specifies what it happens to. The functional model specifies the results of a computation without specifying how or when they are computed.

Further objects may be identified during the discussion of states and functions but in the main the majority of objects within the problem domain are identified during the development of the object model. 'The object model is most fundamental, however, because it is necessary to describe what is changing or transforming before describing when or how it is changes' (Rumbaugh et al. 1991).

The actual development process of OMT is iterative and as the methodology creators state the seamlessness of object-oriented development makes it easier to repeat the development steps at progressively finer levels of detail. Rumbaugh et al (1991) see each iteration as adding or clarifying features that have already been identified, they do not acknowledge that each iteration could also lead to conflicts, inconsistencies and errors.

6.2 Evaluation of OMT focused on Object Identification

The rest of the chapter uses the NIMSAD framework to examine what OMT has to say about the problem situation, the intended problem solver and the problem solving process with an emphasis on object identification. Object identification is the starting point of any object analysis or object modelling methodology and is concentrated in the activities, defined by Rumbaugh et al. (1991) as; identify objects and classes; identify associations between objects; and identify attributes of objects and links.

6.2.1 Element 1: The 'problem situation'

NIMSAD requires the problem solver to consider wider issues surrounding the use of a methodology – for example the size and the nature of the organisation, the identity and belief of the people within it (namely the client and the stakeholders), the behavioural patterns of the stakeholders, and their perceptions of the current situation. Therefore it is essential that a methodology is able to explore and consider a given 'problem situation'.

Rumbaugh et al state that

> 'the problem statement describes the problem to be solved and provides a conceptual overview of the proposed system'.

The implication of this is that the analysts do not question the motivations and justifications for the 'system' nor do they question the relevance or otherwise of the boundaries perceived by the clients. The methodology creators are therefore adopting an ontological notion of systems and are accepting the problem statement as given.

OMT methodology states that the analysts must work with the requester (client) to refine the requirements in order to determine more clearly what it is the requester actually wants. This involves:

> 'challenging the requirements and probing for missing information. However psychological, organisational and political considerations for doing this are deemed to be beyond the consideration of the methodology' (Rumbaugh et al. 1991).

How can systems developers identify the 'real' needs of system users without considering the organisational issues? The 'hard' systems thinking model that has influenced IS thinking in the past is that organisations can be fully understood as being goal-seeking, adaptive-regulatory mechanisms. This is now being rejected in favour of more complex systems models of human organisations as

> 'purposeful, socio-political systems in which shared meaning and symbolic relationships are maintained and modified through human discourse and interaction' (Lewis 1994).

In failing to empathise with the ways in which potential users understand their world, the developers might easily design information systems that breach particular, norms or taboos. These norms and taboos may be perfectly rational in the context of the organisation and rely on a shared understanding of the problem domain and the terminology used. To enable one person to understand another, there must be some norms governing the use of signs, which are established and shared in a language community. Meaning can only be understood if the context where the sentence is uttered is taken into account (Lui 1999).

'Challenging the requirements and probing for missing information' does not constitute determining what the client actually wants. OMT methodology creators adopt an ontological perspective and accept the problem statement as given. They do recommend the methodology user to clarify requirements but at no point do they undertake systemic analysis and conduct a critical enquiry of the situation of concern (Jayaratna 1994). The methodology creators are effectively concentrating on the organisational actions and decisions that have observable, substantive outcomes such as inventory levels or average time taken to satisfy an order, i.e. identifying task responsibilities, charting the flow of material, data and are not considering how the organisational activities are perceived, interpreted and legitimated.

How to distinguish the 'real' needs of system users without considering the organisational issues is unclear in OMT, particularly when those requirements do not represent the 'real' ones from the systems users' perspective. Statements such as

> '... If you do exactly what the customer asked for, but the result does not meet the customer's real needs, you will probably be blamed anyway' (Rumbaugh et al. 1991),

indicate that in the methodology creators opinion system developers could be blamed even if the system functions as the customer requested. Here, the issue is related to the measurement of system success and must be related both to managing the client as well as the problem solving process. As suggested in OMT, system success may depend on trade off priorities (Rumbaugh et al. 1991). The limited advice given testifies that they present their methodology based on a solid background of designing software application systems. This means they are well aware of the psychological, organisational and political considerations in the context of developing and implementing new or changed software application systems but are not prepared to state them or offer guidance on their consideration.

According to Rumbaugh et al. (1991) developers can identify a system boundary by listing the inputs and outputs. Illustrative examples, of a problem statement, within the text, show that this statement is taken as delimiting a software application system. For example: the text uses a case study of an Automated Teller Machine (ATM), in this example all interaction between the 'system' and the outside world pass through the ATM, all input and output values are parameters of ATM events. By identifying the

input and output values, for example: bank code and card code from a cash card; password, transaction type, amount from the ATM user; cash, receipt and messages from the ATM; a diagram comparable to a context diagram in a traditional systems analysis methodology is being considered. The system boundary is therefore being predefined and is not open to negotiation or further consideration. The OMT creators suggest that a problem statement is the starting point for stating requirements; accordingly, what they call 'system boundary' has a more limited scope than the 'boundary of concern' suggested by NIMSAD. System boundaries, according to OMT, can be explored by considering how the boundary conditions, initialisation, termination and failure, are handled.

The problem situation is viewed through the filter of the problem solver, who in turn may have compiled this view through discussions with the client. This can be problematic as there is a risk that the 'problem solver' may conceptually reduce the client or client organisation to the status of 'object' and this may lead the problem solver in misinterpreting the client's view on the nature of the problem situation, and the required system which is intended to provide a solution.

The 'action world' system described by the problem statement must be understood, and its essential features abstracted into a model. Statements in natural language are often ambiguous, incomplete and inconsistent. The analysis model is a precise, concise representation of the problem situation that permits answering questions and building a solution. The process of constructing a rigorous model of data relevant to the problem domain forces the developer to confront misunderstandings of the data and associated relationships early in the development process while they are still easy to correct.

The OMT creators consider their methodology as suitable to all kinds of environments and applications. The text includes descriptions of implementation of object-oriented designs in various target environments, including object-oriented languages, non-object-oriented languages and relational databases. Software application systems development lies in understanding system requirements, planning a solution, and implementing a program in a particular language. OMT creators state that the methodology applies equally well whether applied using a rapid prototyping approach

or more traditional life cycle approach. The notion of objects provides a solid basis for specifying, designing and implementing software in one pass, or gradually building software through multiple passes, evidenced by the examples given in the text by the methodology creators. Rumbaugh et al demonstrate this in the text by presenting a number of case studies of object-oriented applications that cover a variety of application domains and implementation targets ranging from object diagram compiler to computer animation and electrical distribution design systems, case studies of object-oriented applications developed by the authors while working at the General Electric Research and Development Centre are also presented.

6.2.2 Element 2: The intended problem solver

NIMSAD focuses on the role of the problem solver's 'mental construct' that helps understand the complexity of the 'problem situation'. A methodology is a 'brain structuring' mechanism, and if it is to be used successfully, then its user must have the necessary characteristics and preparation. Additionally

> 'it helps us to examine whether the methodology alerts us to the need for developing our 'mental constructs' to a desirable level in order to apply the methodology successfully (Jayaratna and Oates 1995).

The text, (Rumbaugh et al. 1991), is intended for use by both software professionals and students, and although prior knowledge of object-oriented concepts is not required the authors assume familiarity with basic computing concepts. The methodology creators use terminology that is

> 'universally accepted when possible, otherwise they try to choose the best terms among various alternatives' (Rumbaugh et al. 1991).

The methodology creators do not discuss how this is achieved nor do they alert the methodology user to the implied beliefs or shared understanding that the methodology creators assume.

The text is presented in a number of sections focusing initially on object-oriented concepts, then the Object Modelling notation. OMT provides substantial details on techniques, advice, practical tips and exercises to guide its users on how to apply the methodology. This makes the methodology users believe they will achieve success. The ability of the intended problem solver to understand the notation is very

important, again the methodology creators do not discuss how this can be achieved other by already familiar examples in the text.

A methodology is meant to help methodology users to develop their understanding and structure their thinking. OMT identifies the stages to be considered and gives examples in the text as to how the strategies have been applied in 'action world' situations. The methodology user is not given any other guidance on how to develop their experience, other than work through the given examples and then apply them to the target domain.

Abstraction is viewed as an important theme in object orientation by the OMT methodology creators and is perceived as a fundamental human capacity that permits us to deal with complexity. Abstraction focuses on the individuals' competencies and reasoning abilities. A model is created as a result of abstraction.

> 'To build complex systems, the developer must abstract different views of the system, build models using precise notations, verify that the models satisfy the requirements of the system, and gradually add detail to transform the models into an implementation' (Rumbaugh et al. 1991).

Abstraction is seen as having two meanings in the OMT methodology: a feature of object-orientation and an ability of system developers. The text does not assist the methodology user in developing their abilities in abstraction. Rather abstraction is seen as an innate ability of the methodology user. The reasoning abilities of the methodology user relates to the discussion of 'mental constructs' of methodology users in NIMSAD. According to Jayaratna (1994) system developers learn to understand the concepts underlying their own thought processes, such as what makes them reason in a particular way, and clearly articulate those thought processes. The methodology does not alert the methodology user to the importance of this ability, consequently if an ill-considered abstraction is identified the resulting object model may not reflect the same meaning as that intended by the client or user.

The OMT authors state

> 'Objects are distinguished by their inherent existence and not by description of properties'.

For example: all classes must make sense in the application domain, unnecessary or incorrect classes can be discarded by considering whether they are: - redundant classes; irrelevant classes; vague classes; attributes; operations; roles; or implementation constructs. There is no guidance given to the methodology users as to how one identifies which are relevant objects in a problem domain, rather the methodology creators

> 'attempt to foster a pragmatic approach to problem solving drawing upon the intuitive sense that object-oriented technology captures ..... using it systematically on real problems' (Rumbaugh et al 1991).

The relevance or irrelevance is based on the methodology users judgement and opinion, which is very subjective and depends, to a large extent, on the methodology users understanding of the problem domain and on the shared understanding of the client, users and problem solver.

An OMT object model provides

> 'an intuitive graphical representation of a system and is valuable for communicating with customers and documenting the structure of a system' (Rumbaugh et al. 1991).

This indicates that customers are treated as problem owners but not as problem solvers. Users co-operate with systems analysts in building an object model, but as OMT emphasises models, techniques and tools, users are considered as problem owners and not methodology users. The OMT creators do not alert their methodology users to the abstract features of the graphical representation of a system that they advocate. Without learning it how can the system users understand the representation, thereby validating the proposed model?

OMT is presented as a series of steps, with techniques and notational conventions associated with each step, but the order of steps can be interchanged or combined when appropriate.

> 'You can combine several steps once you are experienced... If you are just learning object modelling, however, we recommend that you follow the steps in full detail the first few times' (Rumbaugh et al. 1991).

There is no further explanation of how to combine steps and under what conditions. This is left to 'experience'. By implication the novice user lists all candidate objects

and then considers each one in turn, to determine its relevance to the problem domain and to confirm whether it is an object or an attribute. The 'experienced user', by implication selects appropriate objects from the problem domain intuitively and differentiates between relevant and irrelevant objects without writing them down. The danger of this is that the experienced user relies on previous practice and application areas and may 'see' an 'obvious solution' without exploring the problem situation in sufficient detail. There is no further explanation of what might facilitate the transition from a new to an experienced user, although experienced users seem to play an important role in the OMT methodology.

There is no attempt to highlight the ethical issues or other elements of the methodology users' 'mental constructs', which should be consciously considered according to the NIMSAD framework.

6.2.3 Element 3: The problem solving process

The problem solving process 'has three essential phases: problem formulation, solution design and design implementation' (Jayaratna 1994).

6.2.3.1 Stage (i) Understanding the 'Situation of concern'

OMT does not alert its users to the conceptual construction of boundaries or identification of a 'situation of concern'. Boundary construction is either trial and error or predefined by the client. Questioning 'why' the system is required corresponds to understanding the 'boundary of concern' within the NIMSAD framework. Boundary constructions are based on the data that are meaningful to the OMT user, data that is considered by the OMT user to be relevant to the requirements list and that data which the client insists on as being relevant for their systems.

Problem solvers recognise the 'problem situation' after perceiving the user's problem statement, and formulate it in their own ways to develop 'notional systems' (Jayaratna 1994). The methodology creators assert that

> 'analysis begins with a problem statement generated by clients and possibly the developers. The problem statement should state what is to be done and not how it is to be done, it should be a statement of needs, not a proposal for a solution' (Rumbaugh et al. 1991).

The methodology creators recognise that the problem statement may be incomplete or informal and that analysis makes it more precise and exposes ambiguities and inconsistencies,

> 'most problem statements lack essential information, which must be obtained from the requester or from the analyst's knowledge of the real world problem domain' (Rumbaugh et al. 1991).

To rely on the analyst's knowledge of the problem domain could lead to disastrous results because of the possibility of assumptions being made by the analyst that are in conflict with the clients perspective. The consequence of developing an object model and subsequently a solution based on erroneous assumptions could be catastrophic. The purpose of analysis is to fully understand the problem and its implications.

Within OMT the problem statement serves as the methodology context and therefore has a predefined boundary. OMT does not attempt to discuss broader issues; instead the 'application domain experts' are involved in problem formulation and interpersonal communication with the methodology users as problem solvers. The initial problem domain is taken as given and little consideration is given to the wider context in which the problem is located. Since the boundary determines the focus of the investigation and establishes a 'situation of concern', there is a need to examine whether relevant elements (persons, materials, activities, flows) have been identified in the situation for study. By failing to do this we may well be solving the 'wrong problem'. For example: difficulties are experienced with producing appropriate information relating to courses from a student record system. If administrators have been identified as the users a different interpretation of the problem will be given to that perceived by academics. The resulting system could still be seen as problematic when taken from an academic perspective.

Rumbaugh et al. (1991) argue that

> 'an object-oriented approach takes objects, in the first place, which are then used to understand the problem domain'.

As stated in OMT

> 'we define an object as a concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand. Objects serve two purposes; they help to promote a particular understanding of the real world

and provide a practical basis for computer implementation' (Rumbaugh et al. 1991).

Thus a problem boundary in terms of object becomes essential in formulating an object model and describing the structure of objects.

The 'problem situation' is abstracted in terms of objects and their relationships, from a problem statement formulated by the requester. The objects are exactly defined by the graphical notation system of OMT.

6.2.3.1.1 Investigation models and techniques

The analyst works with the requester to refine requirements; there is no indication or assistance within the text, to assist the methodology user or to help them determine appropriate methods. It is implicit that interviews, both formal and informal, and questionnaires are used to elicit the required information. The dynamic model uses a series of scenarios of typical dialogues within the application area in order to extract events that can then be used to further clarify the requirements.

6.2.3.2 Stage (ii) Performing the Diagnosis

Diagnosis can be defined as the 'thorough analysis of facts or problems in order to gain understanding'. The methodology should therefore help the user to understand the nature and scope of the situation, and any issues that surround them, in an explicit and clear fashion. There are two separate facets to this determination of issues – firstly, the methodology should provide a clear outline of the 'problem situation', and secondly, it should help the user to clarify the reasoning that underlies the 'situation of concern'. The methodology creators state

> 'problem statement should state what is to be done and not how it is to be done. It should be a statement of needs not a proposal for a solution' (Rumbaugh et al. 1991).

If, as Rumbaugh et al imply, the problem statement is a statement of needs, what is the role of the problem solver? And who is performing the diagnosis?

OMT states that the

> 'problem statement is written in consultation with requesters, users and domain experts'

and acknowledges that problem statements often lack essential information, in addition 'many problem statements, mix 'true' requirements with design decisions.' The methodology user is encouraged to communicate with the requester to 'clarify ambiguities and misconceptions' and to identify the 'true' requirements. How this is achieved is not explicit within the text. Missing information can be

> 'obtained from the requester or from the analyst's own knowledge of the 'real world' domain' (Rumbaugh et al 1991).

This implies that the users are not involved with the development of an object model once the initial problem statement has been written. Rumbaugh et al. (1991) state

> 'problem statement is just a starting point for understanding the problem, not an immutable document. The purpose of subsequent analysis is to fully understand the problem and its implications'.

From whose perspective the problem statement is written is not identified or discussed further.

OMT attempts to help problem solvers gain an understanding of the current 'problem situation' by 'identifying' objects that are relevant to, or have an effect upon the problem situation: identifying relationships (associations) between objects and identifying attributes. Rumbaugh et al. (1991) offer little guidance as to how objects should be identified other than intuitively claiming that they 'foster a pragmatic approach to problem solving'. Guidance on object identification is restricted to examples, given in the text, and left to the experience of the methodology user. OMT has specific stages devoted to object identification, however much of the text is devoted to descriptions and guidelines of what an object is. These are generally not sufficient because they rely heavily on identification of nouns which is the same approach taken by many of the traditional structured analysis methodologies when identifying entities. The methodology creators' state;

> 'Decomposition of a problem into objects depends on judgement and the nature of the problem.'

Problems cannot be decomposed into objects, although it is possible to subjectively abstract relevant objects from a problem situation. OMT however does not offer any guidance to is users on how this can be achieved. There is no indication that the methodology user should ask for clarification from the problem owner, in order to

obtain assistance with the abstraction of relevant objects. It is left to the methodology user themselves to determine what are relevant objects.

OMT methodology creators advise that candidate object classes can be found in the written description of the problem. It is suggested that the methodology user should write down every class that comes to mind. This does not help with the identification of potential object classes within the problem domain. The list of candidate classes is then refined so that it includes only those objects and classes that are deemed to be essential / relevant within the problem domain. OMT provides a list of criteria (see table 6.1) to assist the methodology user in this task but ultimately the inclusion or otherwise of objects is left to the methodology users discretion and his/her understanding of the problem domain.

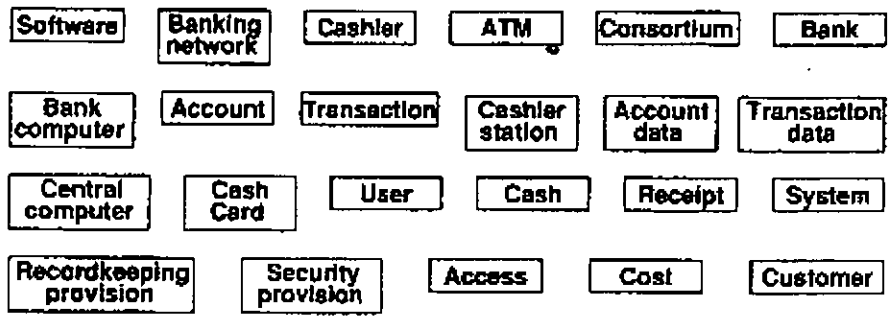
Criteria	Reason for exclusion
Redundant classes	If 2 classes express the same information the most descriptive name should be kept
Irrelevant classes	A class that has little or nothing to do with the problem domain should be eliminated. This involves judgement as in another context the class could be important
Vague	Classes should be specific
Attributes	Words that describe individual objects should be regarded as attributes
Operations	If word describes an operation that is applied to objects and not manipulated in its own right, then it is not a class
Roles	Should reflect the intrinsic nature and not the role it plays in an association
Implementation constructs	Constructs relevant to implementation should be removed from analysis model – may be needed at design stage.

Table 6.1 Criteria used to refine candidate objects

The methodology creators also state that 'additional classes can be identified from our knowledge of the problem domain'. No explicit help is given to help the methodology user identify these additional classes. It is implied that a thorough understanding of the problem domain and experience, in both the problem domain and the methodology, will enable the intended problem solver to determine appropriate objects.

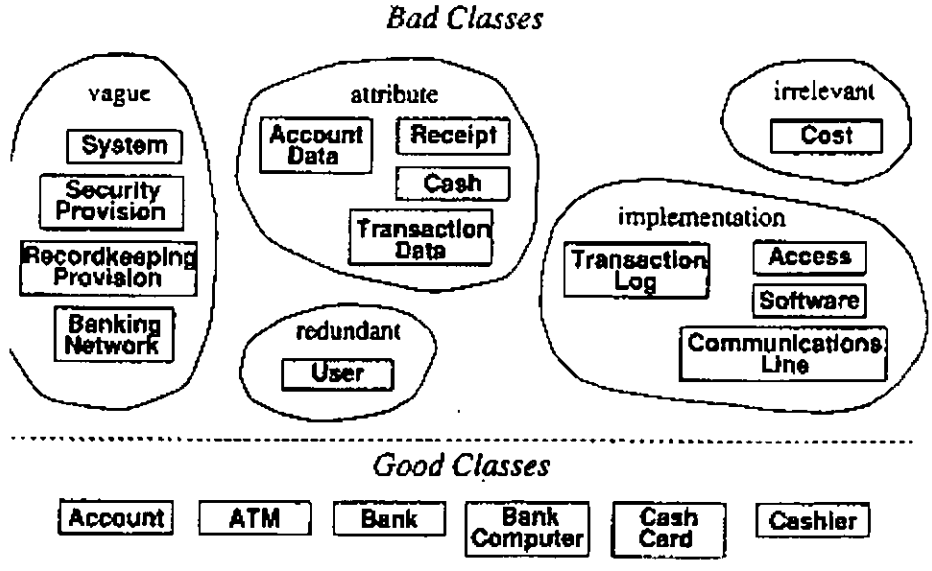
The ATM example is used, presumably because it is familiar to most readers, to identify all classes from the problem statement and the additional ones based on knowledge of the problem domain. The use of familiar examples may be useful to

explain the steps of the methodology but it is difficult to translate how these steps are performed in an unfamiliar or new problem situation. The methodology creators do not offer any further assistance to the methodology user, other than a reliance on 'experience'. These classes are then refined in order to remove irrelevant classes (Fig 6.1)



ATM classes extracted from problem statement nouns

ATM classes identified from knowledge of problem domain



Eliminating unnecessary classes from ATM problem

Figure 6.1 Identification of relevant classes

(adapted from Rumbaugh et al 1991)

Any dependency between two or more classes is an association. A reference from one class to another is an association. Associations are similar to the notion of relationships in entity relationship modelling. Associations often correspond to verb phrases (next to, part of, contained in, drives, talks to, has) again OMT advises the methodology user to extract all the candidates from the problem statement, get them down on paper first and refine later. For example: Vendors own House, Potential Buyers view a House.

These associations are those that would be identified during the normalisation process in structured analysis methodologies or in data oriented design methodologies used in database design. Some associations depend on 'action world' knowledge or assumptions. For example: Potential buyers make an Offer for a Property. These must be verified with the requester, as they may not be explicit in the problem statement. Again the methodology creators provide a mechanistic set of criteria to be applied in order to identify irrelevant associations (see table 6.2).

Criteria	**Reason for exclusion**
Associations between eliminated classes	Is one of the classes has been eliminated then the association no longer exists
Irrelevant or implementation associations	If the association is outside the problem domain or deals with implementation details it should be removed
Actions	Associations should not identify transient events
Ternary or derived associations	Associations that can be restated in terms of other associations should be eliminated

Table 6.2 Criteria used to refine associations

Using the ATM example the methodology creators identify verb phrases and implicit verb phrases from the problem statement. Again by assuming the reader is familiar with the problem domain the methodology creators identify those verb phrases which depend on knowledge of the problem domain i.e. how an ATM functions (Fig 6.2). The process used to identify these is not explicit relying on the 'experience' of the methodology user.

Verb Phrases

Banking network includes cashiers and ATMs
Consortium shares ATMs
Bank provides banks computers
Bank computer maintains accounts
Bank computer processes transactions against account
Bank owns cashier station
Cashier station communicates with bank computer
Cashier enters transaction for account
ATMs communicate with central computer about transaction
Central computer clears transaction with bank
ATM accepts cash card
ATM interacts with user
ATM dispenses cash
ATM prints receipt
System handles concurrent access
Banks provide software
Cost apportioned to banks

Implicit Verb Phrases

Consortium consists of banks
Bank holds accounts
Consortium owns central computer
System provides record keeping
System provides security
Customers have cash cards

Knowledge of problem domain

Cash card accesses accounts
Bank employs cashiers

Figure 6.2 Associations from ATM problem statement

(adapted from Rumbaugh et al 1991)

Attributes are unlikely to be fully described in the problem statement. The methodology users are encouraged to draw on their knowledge of the application domain and the real world to find them and to consider only attributes that directly relate to a particular application. For example: in a mailing list, city could be considered as an attribute, however in a census, City would be an object. The methodology creators provide a mechanistic list of criteria to assist the methodology user in identifying irrelevant attributes (see table 6.3).

Criteria	Reason for exclusion
Objects	If independent existence of the 'thing' is important rather than its value, then it is an object. Distinction between object and attribute may depend on application context
Qualifiers / Names	If the value of an attribute depends on a particular context then it may be a qualifier on an association rather than an attribute. A name may also indicate a qualifier on an association
Identifier	Object ID's are implicit in object models and should not be classed as attributes.

Table 6.3 Criteria used to refine attributes

The OMT methodology creators provide practical tips and guidelines that are 'gleaned from practice in constructing object models'. These examples offer descriptive statements as to how objects, associations and attributes can be identified but essentially the final decision is based on the 'experience' of the methodology user.

In line with object-oriented concepts OMT supports the notion of aggregation, generalisation, specialisation and inheritance. Inheritance can be added in two directions: by generalising common aspects of existing classes into a superclass (bottom up) or refining existing classes into specialised subclasses (top down). Some attributes or classes may have to be redefined slightly to fit generalisation or specialisation. OMT methodology sees this as acceptable and states that some generalisations will suggest themselves based on 'existing taxonomy in the real world.'

An object model, within OMT, is rarely complete after a single pass. The entire system development process is one of continual iteration, different parts of a model are often at different stages of completion. If a deficiency is found, the methodology user is encouraged to return to an earlier stage if necessary to modify or refine it. Some refinements can only come after the dynamic and functional models are completed. The methodology creators offer a list of criteria to apply to determine whether there are any missing objects (see table 6.4). Unnecessary classes can be identified by a lack of attributes, operations and associations.

Criteria	Solution
Asymmetry in associations and generalisations	Add new class by analogy
Disparate attributes and operations on a class	Split class so each part coherent
Difficulty in generalising clearly	One class may be playing 2 roles, create an additional class
An operation with no good target class	Add missing target class
Duplicate associations with the same name and purpose	Generalise to create missing superclass that unites them
A role substantially shapes the semantics of a class	Create a separate class

Table 6.4 Criteria used to identify missing objects

Rumbaugh et al (1991) use scenarios of both typical dialogues and exception cases in order to identify events that occur in the problem domain. There is no suggestion that scenarios could identify additional objects (Jacobson et al. 1998).

OMT helps to provide a graphical representation of objects and their relationships located within the problem domain, it does not provide a clear outline of the 'problem situation' not does it clarify why or how the problem has been identified. OMT is presented as a scientific way of undertaking systems development yet many of the stated criteria depend on subjective statements.

6.2.3.3 Stage (iii) Defining the Prognosis Outline

By 'defining the prognosis outline' NIMSAD means that the methodology should be able to facilitate the identification of a clear defined 'desired state' in relation to the 'situation of concern'. Jayaratna (1994) expresses this as defining 'where we want to be and why?'

OMT relies on the clients being clear about the rationale for their requirements. It does not offer any steps for examining the validity of these expectations. OMT does not assist the methodology user in the identification of 'current state' and 'desired state' what it is doing is taking the problem statement and producing an object model to represent the requirements of the problem statement. Rumbaugh et al. (1991) state that the problem statement 'should be a statement of needs, not a proposal for a solution'. By modelling the 'needs' the object model is perceived to have 'solved' the problem without initially identifying what the problem is.

6.2.3.4 Stage (iv) Defining problems

NIMSAD uses this stage to investigate what the methodology can offer in terms of providing an answer to the question ' What are the problems which currently prevent the transformation of the 'problem situation' from its 'current state' into the desired state'?

As OMT does not concern itself with the clients 'desired states' (prognosis outline) it has no means of helping in defining problems either. Problems are formed at a client level. Some of the changes to requirements are legitimate, as they are made in response to changes in the environment, i.e. legislation, customer preferences etc. There may be political or organisational reasons for the requirements but the analyst should recognise that these externally imposed design decisions are not essential features of the problem domain. However, OMT does not offer any techniques for

facilitating such issues, the methodology creators stating 'psychological, organisational and political considerations ... are deemed to be beyond the consideration of the methodology'.

According to Rumbaugh et al (1991) many clients do not spend sufficient time and effort on explicit reasoning about their requirements. In other words, clients very often do not use methodologies for arriving at their requirements. OMT is not dealing with the relevance of problems but the construction of solutions to meet a requirement. This is implied by the methodology creators when they suggest that the methodology users check the problem statement from clients and make their own decision about the problem they should solve,

> 'because the statement may be informal, incomplete and lacking essential information. Analysts must work with the requester to refine the requirements so they represent the requester's true intent. This involves challenging the requirements and probing for missing information' (Rumbaugh et al. 1991).

6.2.3.5 Stage (v) Deriving Notional Systems

NIMSAD defines notional systems as

> 'those systems which need to be developed if the client organisation is to overcome the previously defined 'problems', thereby helping to transform the 'current state' to the 'desired state' (Jayaratna 1994).

This means identifying systems using the 'mental construct', which, if designed and implemented, would eradicate one or more of the perceived problems within the current system.

As OMT does not help its users to participate in the definition of 'desired states' it cannot define problems, nor can it arrive at notional systems. The concern of the methodology is to record the features of the notional system(s) expressed by the client in a concise form in order to facilitate its design tasks. In practice, user requirements guide the whole methodology process and they are taken into account at the beginning of the project. The methodology helps to explore the definition of elements that can be expressed in class, object and structure form.

6.2.3.6 Stage (vi) Performing Conceptual / Logical Design

The logical and conceptual design of a system is intended to formalise the structures, roles, tasks, functions, information and attitudes of the notional system(s) into a form that represents a clear, logical methodology of achieving the ideas identified in the notional system(s).

The goal in constructing an object model is to capture those characteristics from the 'action world' that are important to a software application system. It is a conceptual process, independent of programming language until the final stages (Rumbaugh et al. 1991). In OMT, the logical and conceptual design of a software application system is intended to formalise its structure, to organise the system into sub-systems, to allocate sub-system to processors and tasks and to handle access to the global resources. With OMT, functionality is the major concern when partitioning the system into sub-systems.

The methodology provides support to this stage by considering related issues in system design, such as the architectural framework, data store, interface, dynamic simulation and hardware. The techniques that OMT utilises are sufficiently rigorous to make them fairly useful tools for methodology users to enable representation of logical components (sub-systems) and their interactions. OMT methodology does not pay much attention to the 'problem situation', but rather to the technical support to handle the problem situation by considering objects and models.

6.2.3.7 Stage (vii) Performing the Physical Design

The physical design of a system is the process of transforming the logical / conceptual design into a form suitable for implementation (given a set of identified constraints). The systems design phase of OMT, is the process of establishing a suitable system architecture that organises the system into sub-systems based on both the analysis structure and the proposed architecture. The steps in achieving the architectural design include dividing a system into sub-systems, choosing the implementation of control, and choosing approaches for handling data to be stored.

During system design, decisions are made about how software application system problems will be resolved. In other words, system design is a high level strategy for

solving the software application system problems, which may provide a solution to a business problem. Here, systems design belongs to the logical design of an overall organisation of a system called 'system architecture' by Rumbaugh et al. (1991).

6.2.3.8 Stage (viii) Implementing the Design

The implementation of a system is the realisation of the notional system(s) within the 'situation of concern' (Jayaratna 1994). The object design phase of OMT is the process of defining classes, their interface and implementation. Internal classes are added when needed, and algorithms and data structures are optimised. Data structures and algorithms needed for implementation are the focus of object design. Both application domain objects and computer domain objects are considered within object design. In implementation, the object classes and relationships developed during object design are finally translated into particular programming language, database and hardware.

The object design phase elaborates and refines the analysis model, which is then optimised to produce a practical design. During object design there is a shift in emphasis from application concepts towards computer concepts. First the basic algorithms are chosen to implement each major function of the system; the structure of the object model is optimised for efficient implementation based on these algorithms. The design must also account for concurrency and dynamic control flow as determined during system design. The implementation of each association and attribute is determined. Finally the sub-systems are packaged into modules. The lack of explicit diagnosis and prognosis models, as discussed in the NIMSAD framework, can be expected to make implementation a more demanding task.

6.2.4 Element 4: Evaluation

'It is the evaluation which helps us to measure the effectiveness of the problem solving process and the problem solver in the 'problem situation' – unless the element is considered there is no way of establishing that the 'problems have been successfully resolved' (Jayaratna 1994).

Rumbaugh et al (1991) compare their own methodology with other software engineering approaches, including the object-oriented approaches of Booch (1994), Meyer (1988), Coad and Yourdon (1991). The OMT methodology creators base the

comparison for identifying strengths and weaknesses of each approach on 'a set of flexible criteria'. The purpose for such a comparison is to identify major differences and similarities between OMT and other approaches and effectively only contributes to an evaluation of OMT before application.

This comparison does not replace a methodology evaluation using the NIMSAD framework. The OMT methodology creators do not consider assessing the methodologies before, during and after application. The OMT methodology creators do not raise questions relating to the influence of object-oriented thinking on systems developers with experience of function-oriented system development. Object-oriented methodologies are described as a new paradigm but again no questions are raised to justify this viewpoint. The comparison of methodologies used by the OMT creators is more likely to raise questions relating to the differences between object-oriented methodologies and function oriented methodologies, and also question why object-oriented thinking should be introduced.

6.4 Summary

This chapter began with a brief introduction to OMT and then evaluated OMT using the NIMSAD framework with an emphasis on object identification. The main findings of this evaluation are as follows.

OMT methodology creators adopt an ontological perspective and accept the problem statement as given. They recommend that the methodology user clarify requirements but at no point do they undertake systemic analysis and conduct a critical enquiry of the situation of concern. The methodology creators are effectively concentrating on the organisational actions and decisions that have observable, substantive outcomes such as inventory levels or average time taken to satisfy an order, identifying task responsibilities and charting the flow of material etc and are not considering how the organisational activities are perceived, interpreted and legitimated.

The methodology creators claim to use terminology that is 'universally accepted when possible, otherwise they try to choose the best terms among various alternatives'. How

this is achieved is not discussed nor does OMT alert the methodology user to the implied beliefs or shared understanding that the methodology creators assume.

Abstraction is viewed as an important theme in object orientation by the OMT methodology creators and is perceived as a fundamental human capacity that permits us to deal with complexity. Abstraction focuses on the individuals' competencies and reasoning abilities. OMT does not alert the methodology user to the importance of reasoning ability and abstraction, nor does it assist the methodology user in developing his/her abilities in abstraction. Rather abstraction is seen as an innate ability of the methodology user.

OMT has specific stages devoted to object identification; these are generally rely heavily on identification of nouns which is the same approach taken by many of the traditional structured analysis methodologies when identifying entities. The relevance or irrelevance is based on the methodology users' judgement and opinion, which is very subjective and depends, to a large extent, on the methodology users understanding of the problem domain and on the shared understanding of the client, methodology user and other stakeholders.

Additional classes can be identified from knowledge of the problem domain; however, no explicit help is given to help the methodology user identify these additional classes. It is implied that a thorough understanding of the problem domain and experience, in both the problem domain and the methodology, will enable the intended problem solver to determine appropriate objects. The OMT methodology creators provide practical tips and guidelines that are 'gleaned from practice in constructing object models'. These examples offer descriptive statements as to how objects, associations and attributes can be identified but essentially the final decision is based on the 'experience' of the methodology user.

The experienced user appears to play an important role in OMT, being able to select appropriate objects from the problem domain intuitively and then differentiate between relevant and irrelevant objects without writing anything down. The danger of this is that the experienced user relies on previous practice and application areas and may 'see' an 'obvious solution' without exploring the problem situation in sufficient

detail. Despite the emphasis on experienced users OMT methodology offers no explanation of what might facilitate the transition from a new to an experienced user.

OMT relies on the clients being clear about the rationale for their requirements. It does not offer any steps for examining the validity of these expectations. OMT does not assist the methodology user in the identification of 'current state' and 'desired state' what it is doing is taking the problem statement and producing an object model to represent the requirements of the problem statement. By modelling the requirements the object model is perceived to have 'solved' the problem without initially identifying what the problem is.

The techniques that OMT utilises are sufficiently rigorous to make them fairly useful tools for methodology users to enable representation of logical components (subsystems) and their interactions. OMT methodology does not pay much attention to the 'problem situation', but rather to the technical support to handle the problem situation by considering objects and model.

Chapter Seven: Critical Evaluation of Object-Oriented Software Engineering

This chapter gives a brief overview of Jacobson et al's (1998) Object-Oriented Software Engineering (OOSE). OOSE is then evaluated using NIMSAD framework. The evaluation concentrates on how OOSE helps the methodology user to determine what objects are relevant when modelling the problem domain.

7.1 Introduction to Object Oriented Software Engineering (OOSE)

Object-Oriented Software Engineering (OOSE) (Jacobson et al. 1998) is a methodology for object-oriented analysis and design derived from Jacobson's Objectory.[2] Objectory is a proprietary methodology and OOSE is said to be a simplified version, which is inadequate for use in production.

> 'You will need the complete... description which, excluding large examples, amounts to more than 1200 pages' (Jacobson et al. 1998).

Objectory is unusual among OO methodologies as it attempts to address the entire system development life cycle.

The underlying enterprise philosophy implies that tools provide support for activities in three categories: architecture, method and process. Architecture signifies the choice of techniques to be utilised, the methodology makes explicit the procedures to be followed and the process provides for scaling the methodology up. The development of software systems is presented as analogous with the construction industry. Development is progressive, involving iterative cycles of analysis, construction and testing.

OOSE develops five different models to capture different aspects of an information system, as shown in Fig 7.1, requirements model, analysis model, design model, implementation model and test model. An earlier model becomes the input for the next model. For example the analysis model is built upon the basis of the requirements model.

[2] Objectory was formulated in 1985 and initially presented at OOPSLA'87 (Jacobson et al 1993)

The creators of OOSE view system development as a sequence of successive changes resulting in a number of versions of an information system over its life cycle, the changes are based on requirements for new or changed systems.

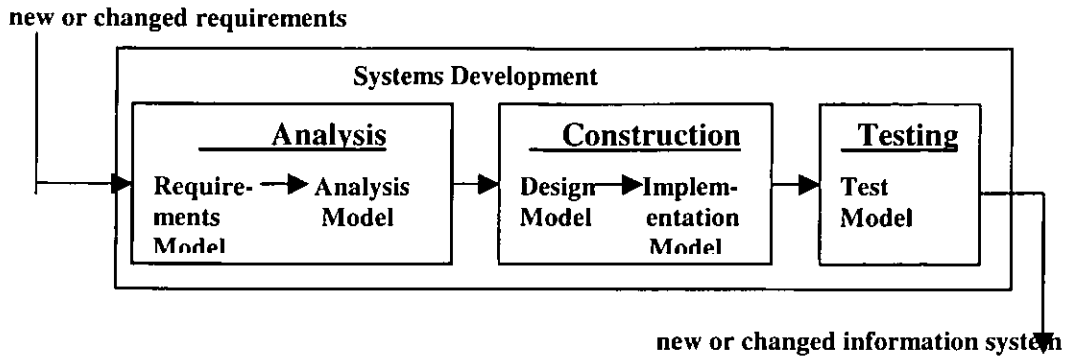


Figure 7.1 Systems development viewed as a process of model building

(Jacobson et al. 1998)

OOSE creators also view the transformation from one model to the next as a process of change. Fig 7.2 illustrates this in the transition from an analysis model to a design model via a design process. Such a model change comprises shifting from one frame of reference, i.e. from an analysis space that includes the consideration of behaviour, presentation and information, to another frame, a design space that adds implementation environment to the analysis space for the information system under development.

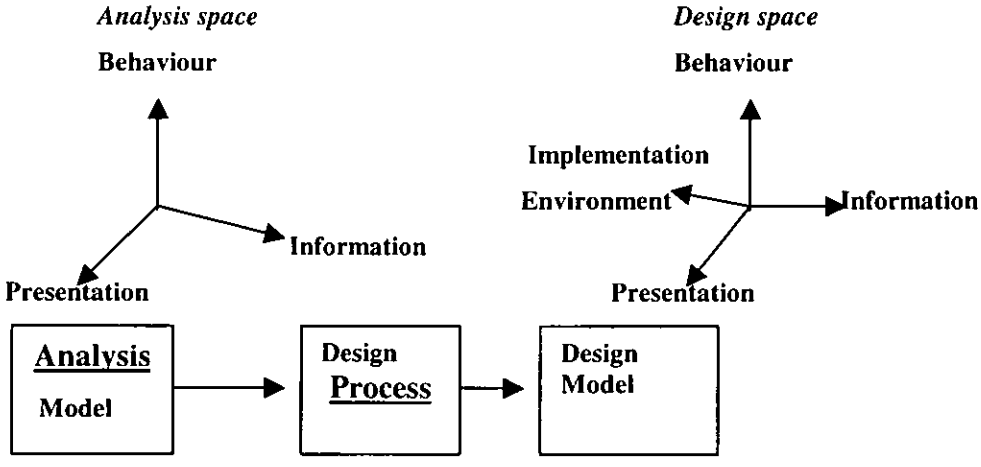


Figure 7.2 A Process changes one model into another

(Jacobson et al. 1998)

7.1.1 Use case driven

All models that apply OOSE are use case driven. Use cases denote descriptions of the functionality expected from an information system by its potential users in their

different interactions with it. The importance of use cases lies in the idea that they can be controlled by users and taken by system developers as part of the requirement model. The use cases and how they are connected are described in a special 'use case model' (Jacobson et al. 1998).

The use case model is built during the first analysis activity, the requirements analysis. It constitutes an integrated part of the requirements model. It is also used as the basis in building all remaining models. In these later models it may have to be rechecked and elaborated in collaboration with potential users. As all models, after the requirements model, are related to the use case model, the traceability of features is supported from one model to another, aiding documentation reuse.

The concept of use cases was employed before the advent of object-orientation. This means that system users risk using function-orientation when thinking in terms of use cases. However, this provides a way for systems' users to participate in systems development without learning the object-oriented thinking and views that the OOSE methodology offers. Table 7.1 shows the sequence of events identified by the OOSE methodology creators, and the order in which it is suggested that they are carried out.

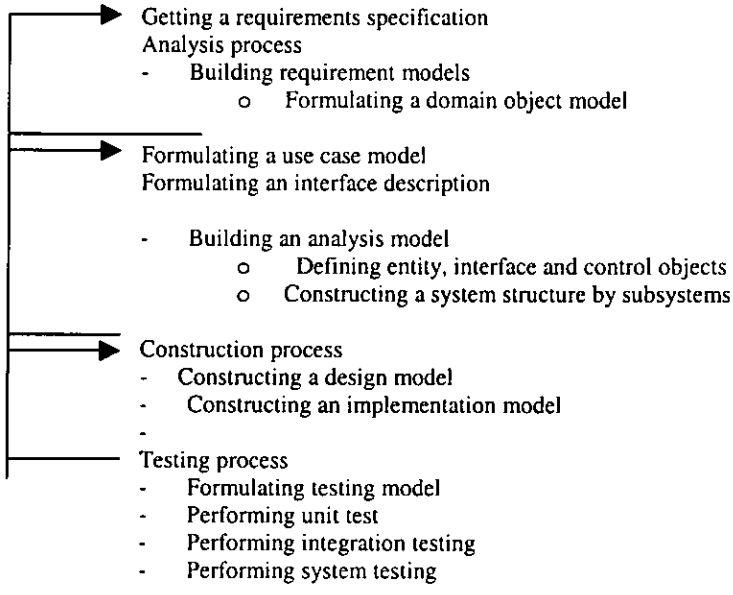


Table 7.1 Problem Solving process in OOSE

(Jacobson et al. 1998)

7.2 Evaluation of OOSE focused on Object Identification

The rest of the chapter uses the NIMSAD framework to examine what OOSE has to say about the problem situation, the intended problem solver and the problem solving process with an emphasis on object identification. Object identification is the starting point of any object analysis and is concentrated in the activities defined in OOSE as; formulating a domain object model and defining entity objects, interface objects and control objects.

7.2.1 Element 1: The 'problem situation'

How does OOSE alert its methodology users to this need for a broad outlook at the start of a systems development project? The OOSE methodology creators issue a warning that in some cases systems development can be preceded by enterprise development.

> 'When the requirements of the enterprise are less well known, the specification work is preceded by enterprise development... In some cases, the customers are highly experienced and can provide the specification during the initial contact with the potential producers. In other cases, the customer approaches a producer and solicits assistance in solving his problems' (Jacobson et al. 1998).

The OOSE creators ignore any deeper understanding, or questioning related to the problem situation, as they presume that appropriate enterprise development has already occurred. In response to who are the clients? posed by NIMSAD framework the answer is 'an orderer as in other cases of product development' (Jacobson et al 1998). OOSE users are alerted to the fact that the clients and other stakeholders may raise different requirement specifications. OOSE methodology users are also alerted to the fact that direct users, supported by an information system, serve a number of indirect users, whose work will also be influenced by a new system. The importance of capturing 'requirements from a user-oriented perspective' is stressed (Jacobson et al 1998). However, the methodology creators state that

> 'the success of the system is highly dependent upon whether it has been possible to capture and formulate the users' requirements in such a way that

the requirements can be formalised and transformed into working programs' (Jacobson et al. 1998).

This claims to assist the methodology user in arriving at 'good' requirements specifications. There is little discussion, in the text, as to how the methodology user determines 'goodness', nor is there any explicit help in the methodology, (see section 7.2.3.2 for a discussion on 'goodness' of objects).

The OOSE methodology does not encourage its users to ask questions such as: What can and will the users do differently once their purported information needs are met? How committed are the users to changing their actions? Raising these and similar questions when reflecting on the problem situation, as suggested by the NIMSAD framework (Jayaratna 1994) could help to integrate information system development with organisational development. The OOSE methodology does not give any explicit advice as to how its methodology users could encourage future systems users to consider the effects of the new information system on those users who are only indirectly involved.

NIMSAD framework warns methodology users not to take the 'problem situation' as stated by the client 'as given'. OOSE methodology users are not alerted to the fact that they must not take the first requirements specification provided by their client 'as given'.

Throughout the text OOSE creators assume that any enterprise development needed has been or will be undertaken in parallel with the work of the methodology users. As shown by several examples in Jayaratna (1994), relying on such suppositions may turn out to be a costly mistake. By alerting managers to the need for some integrated enterprise development work, methodology users can improve the success rate of their information system in satisfying requirements. Such enterprise development, will in many cases, necessity future system users becoming more involved.

The methodology creators advise their methodology users to take clients requirements specifications only as a starting point. According to OOSE receiving the requirements specification, in some form, is the starting point for systems development. The methodology creators stress the need for carefully checking the requirements

specification that is received initially. They advise methodology users to do this in collaboration with current system users and potential system users when building the requirements model. OOSE creators regard requirements specifications as being the functional requirements for data processing and data presentation that information systems have to meet. At an early stage of systems development, the main 'problem', according to OOSE, seems to be producing a requirements specification precise enough to become part of a contractual agreement between a client and a software producer.

This may create a problem if the client is not very familiar with the work situation of people within the area of use. The delivered information system, in most cases may not be used by the client, but by a number of indirect users (Mumford 1983a; 1983b). For this reason OOSE alerts its users to the importance of understanding the users information requirements, rather than those of their clients (Jacobson et al 1998). In many cases of information system development, this shift of attention from a client to a number of intended information system users will be of benefit. However in cases where both the client and other stakeholders form the user group, there may be real conflicts of interest between parties. The advice to 'listen to the users' then becomes an ambiguous statement, which users? How does the methodology user determine the most appropriate perspective? The OOSE methodology users are not alerted to this risk.

By following the OOSE philosophy it is possible to assume that requirements are the input to the production of an information system. The output (the information system) will not solve any real problems if the production is based on incorrect input. To avoid solving 'irrelevant' problems, systems development should always explicitly relate to organisational development. According to Jayaratna (1994), organisational needs should be seen as the purpose of systems development, and solution design should contribute to the effectiveness. In OOSE, the methodology users do not become explicitly alerted to this kind of thinking.

All organisations are confronted by endless changes. This causes the well-known problem of systems 'maintenance', which could be called keeping systems relevant in an ever-changing environment. OOSE methodology confronts these problems as a

methodology for effectively producing a series of system versions over the whole life cycle of an information system. The OOSE creators achieve this by presenting what they call ' a strategy of incremental development (Jacobson et al. 1998).

7.2.2 Element 2: The problem solver

NIMSAD focuses on the role of the problem solvers 'mental construct' that helps understand the complexity of the 'problem situation'. Additionally

> 'it helps us to examine whether the methodology alerts us to the need for developing our 'mental constructs' to a desirable level in order to apply the methodology successfully' (Jayaratna and Oates 1995).

OOSE makes a distinction between different system developers according to their experience and knowledge in system development, for example: experienced OOSE methodology user; experienced software engineer; newcomer to the field, and manager. The methodology creators suggest that the above groups each read different parts of the text. For the roles of system analyst, project manager and designers, the OOSE methodology creators suggest different education programmes to teach them what they need to know about OOSE prior to initiating the first project.

According to NIMSAD the system users not only have to learn how to use an information system, but they also need encouraging to understand models of handling business problems embedded in the information system, such as forecasting and inventory management. The OOSE user-directed documentation identifies the need for users to understand some of them. The problem of elucidating embedded business models, however, has not been explicitly addressed in OOSE.

The ability to apply object-oriented thinking is identified, by OOSE methodology creators, as being important when using their methodology. Object-oriented thinking belongs to the discussion of 'mental constructs' in NIMSAD, it is only one aspect influencing problem solvers' 'mental constructs'. In discussing 'knowledge and skills' Jayaratna (1994) states,

> 'as intended problem solvers, we must become very conscious of the knowledge sets and skills that are required to practice a methodology, and methodology creators must state what knowledge sets we should possess.'

Implicitly this can be interpreted to cover the skill of 'object-oriented thinking' as advocated as necessary by the OOSE methodology creators. A shift from a function–oriented perspective to an object-oriented perspective could also be interpreted as being part of 'reasoning ability' in Jayaratna (1994). Human ability to reason about some field of common interest is largely contingent upon a traditional way of talking about it. 'Reasoning ability' within NIMSAD is discussed at an individual level only.

Object-oriented thinking integrates static and dynamic aspects of entities. It is a change for methodology users from function-oriented system development. OOSE purports to teach its methodology users an object-oriented view by introducing fundamental concepts of object-orientation and applying them in different cases. In OOSE, this new approach to thinking is mainly offered to problem solvers. System users are not informed or trained to develop such thinking, although they join in use case formulation. This means that system developers and users will generally have different styles of thinking. One is a dynamic object-oriented view and the other a function-oriented view. The OOSE methodology creators alert the methodology user to the importance of object-oriented thinking but they do not consider whether the system users should also be made aware of the differences between function-oriented thinking and object-oriented thinking.

The creators of OOSE discuss the risk of conflicts when discussing information systems. There may occasionally be conflicts of interest behind some of the differences in views on information needs between client and users. These differences generally exist because the client knows less about the users' requirements than they think they do. This is the limited type of conflict, within a customer organisation, that OOSE makes its methodology users aware of. OOSE creators suggest that system developers should develop information systems with the system users and not the clients. The co-operation they stress is only between system developers and users. Conflicts between clients' requirements and those of the users could be ignored. The OOSE methodology does not give any reasons why the methodology users should take the system user's perspective, nor does it instruct its users how to handle conflicts with respect to a requirements specification between them and the clients, despite explicitly mentioning this type of conflict.

System users play a central role in formulating the 'right problem'. This clearly indicates that methodology users, as problem solvers, go directly to the problem owners to get the 'right' problems defined and solved. Jayaratna (1994) repeatedly alerts the reader to the fact that in the enterprise, which will become the context for an information system, there will be conflicts of interest, which could become obstacles to the effective use of that system.

OOSE methodology creators recognise that

> 'it is important to carry on a dialogue with the prospective orderers and users, so that the system that is built is really what is wanted. In the analysis phase, it will then be possible to build models that will make it easier for us to understand the system' (Jacobson et al. 1998).

The problem solvers understanding of the system is increased in different stages. This means that the problem solvers learn incrementally. Incremental understanding does not involve every system developer, since each role only conveys limited tasks in system development activities. The education programme advocated by the methodology creators (Jacobson et al. 1998 chapter 15) does not offer all system developers the same learning content, but only a global view of the methodology. Nor is it possible for all system developers to go through the whole process.

In systems development, system users are often defined as people who (will) use the system, whereas the problem solver in NIMSAD is taken as a role, which could be played by different people. 'Role conflict', as a phenomenon, has been discussed in the 'mental constructs' (see Chapter 4). It is described as a kind of mismatch of role expectations between problem solvers and others (users, clients, stakeholders etc). Jayaratna (1994) also suggests that methodology creators should explain the role expectations implied by their methodologies.

OOSE specifies a number of roles on the basis of competence implied by each role in the systems development phases, i.e. analysis, construction and testing. Different phases need different competencies. According to NIMSAD the client may play the role of the intended problem solver, even if not explicitly stated, the repeated use of the phrase ' clients and problem solvers' indicates that a problem owner could play the

role of an intended problem solver. The creators of OOSE assign this role only to information systems professionals.

System developers transcribe and consolidate use cases, received from the system users, into a use case model. Use cases, as described by system users, for the most part will not be framed in an object-oriented way. When methodology users transform these into a use case model as part of their requirements model, they should apply an object-oriented view. How this transition is achieved has not been explicitly addressed in OOSE, for example: what questions should the methodology users ask the systems users in order to check for accuracy in translating the model. When use cases seem unclear or incomplete in building later models, there may be a risk that they go back to the original documented descriptions given by system users and not to a dialogue with the system users themselves. This risk is not explicitly discussed in OOSE.

7.2.3 Element 3: The problem solving process

The problem solving process 'has three essential phases: problem formulation, solution design and design implementation' (Jayaratna 1994).

The NIMSAD framework sub-divides these phases into eight systemically based stages. By analysing the situation in a systemic way, it should be possible to make the transition from 'current state' to 'desired state'. Systemic design can then be utilised to obtain models of notional systems, any or all of which should be able to facilitate the desired notional system(s).

7.2.3.1 Stage (i) Understanding the 'Situation of concern'

According to Jayaratna (1994) a rational problem solving process should start with an understanding of the organisational situation and methodology users should not take the 'problem situation' as stated by a client for granted. They should, in dialogues, with clients, problem owners and other stakeholders, determine where to draw a boundary around a dynamic 'situation of concern', i.e. around the field of activities which is to be considered in the systems development work.

The OOSE creators write about the concept of 'boundary' in the context of delimiting an information system, in terms of what is and can be bounded by the use case scenarios. They call it the 'system border' and it is used to indicate a clear distinction

between what refers to an information system to be developed and what belongs to the 'surrounding world' with respect to using the system (Jacobson et al. 1998). OOSE creators offer no guidance on how to determine whether a realistic 'system border' has been identified other than referencing the use case scenarios.

The context in which the OOSE creators introduce the concept of a 'system boundary' is the transition from an analysis model to a design model. In terms of NIMSAD this means a transition from logical design to physical design within the 'solution design' phase. The OOSE design model includes a modelling technique called 'interaction diagrams'. In these, the interactions of the information systems objects should be documented, however, due to implementation considerations, these objects generally will not show a one-to-one mapping of the problem area objects of the analysis model. For this reason the objects of the OOSE design are called 'blocks'.

In Jacobson et al. (1998) the criterion for distinguishing 'system borders' has been stated in the following way,

> ' in almost all the interaction diagrams we have a bar for the surrounding world, namely for what is outside that which we want to describe (in the OOSE design model). We usually call this bar the system border. This bar represents the interface with everything outside the blocks in the diagram, such as external actors, and consequently it can correspond to different interfaces outside the system. There may be many such border bars' (Jacobson et al. 1998).

This shows that distinguishing an OOSE system border has an entirely different purpose than deciding on a 'boundary of concern' within the problem formulation phase of NIMSAD. The OOSE concept of 'system border' belongs to the solution design in NIMSAD, however it has been discussed here as the OOSE term 'system border' may give the impression of covering the same broad area of a 'boundary of concern' as discussed in NIMSAD. The interaction diagrams of OOSE, in some cases, might also be useful in the context of delimiting a 'boundary of concern'.

The actions users undertake, based on data from an existing information system, belong to their field of activity. The boundary of the 'boundary of concern' will cut

across these fields of activity and often through the fields of activity of other stakeholders.

7.2.3.1.1 Investigation models and techniques

The OOSE analysis process starts with the building of a requirement model. The methodology users have to base this model on descriptions of use cases furnished by potential system users. To acquire an adequate set of these presupposes effective communication between system developer and system users. Having the users draw use cases on paper and utilising prototyping in dialogue with systems developers are two suggested means of investigation. 'Close participation' is the term used to describe the process of formulating use cases. In order to do this for a number of use cases, users have to imagine themselves in many different interactions with the system. This means they have to see themselves in a number of shifting roles, called 'actors' in OOSE. To what extent can they see the difference between these roles? If they are unable to distinguish between roles then the use cases are likely to be distorted. The expected information needs they express in the use cases may then differ considerably from the information needs in the corresponding future interactions with the system. The OOSE methodology does not go further in advising how to achieve communication skills.

Semantic analysis begins by identifying agents/actors who create the social world around them. An agent has some abilities to accomplish things. These abilities are identified and put into the analysis model to construct a picture of the world. Semantic analysis focuses attention of analysis on the agents and the actions.

In semiotics the world to be modelled is constructed by the community of agents, i.e. problem owners. The agents/actors know the meanings of words in their own world, their interpretations are the only ones justified. It is not allowed, within semiotics, for an analyst to invent artificial terms or introduce new concepts when modelling the agent's actions. The purpose of this is to force the analyst to speak the same language as the problem owners. Any ambiguity in the terms or concepts used in describing the problem should be resolved by putting them into a context of actions, which are already described and understood. When doing so, if the problem-owners are inspired

with some new terms, the problem owners and the analyst may use them only after careful consideration.

7.2.3.2 Stage (ii) Performing the Diagnosis

Diagnosis can be defined as the 'thorough analysis of facts or problems in order to gain understanding' (Jayaratna 1994). The methodology should therefore help the user to understand the nature and scope of the situation, and any issues that surround them, in an explicit and clear fashion. Jacobson et al (1998) appear to agree with this definition of analysis as they state

> 'the purpose (of analysis) is to formulate the problem and to build models able to solve the problem under ideal conditions'.

OOSE begins by developing a problem domain model in order to determine the scope and nature of the problem. This is identified as a logical view of the system using objects that have a 'direct counterpart in the application environment that the system must know about'. The problem domain model identifies the actors / roles played by people (or other systems) interacting with the system we are interested in.

The methodology creators state,

> ' objects can be found as naturally occurring entities in the application domain. An object becomes typically a noun, which exists in the domain.... often a good start to learn the terminology for the problem domain' (Jacobson et al. 1998).

This assumes a shared understanding of the problem domain and a common set of norms and values. These are not discussed within the text nor is the methodology user made aware of them. The identification of nouns, used within the application domain, is making use of selected areas of textual analysis although the methodology creators do not refer to this. According to Jacobson et al. (1998) analysis of the terminology allows the methodology user to identify candidate objects in the system, these can then be evaluated to determine how the system needs to handle the objects, inclusion of the object in the final model is then based on this evaluation.

The methodology creators further assert that,

'there is no problem in finding objects, the difficulty is in selecting those objects relevant to the system' (Jacobson et al. 1998).

In order to select relevant objects the methodology user is advised to consider the modifications that are likely to occur to the object during its lifetime as,

'objects which are to remain essential throughout the system's life cycle will probably always exist' (Jacobson et al. 1998).

OOSE methodology users are advised to put the candidate objects into context in order to determine whether it is an appropriate object. What is really of interest is how an object interacts with and reacts to other objects and under what conditions.

The methodology creators assert that

'you cannot draw any conclusions about ('right' object types) until a system has been changed a number of times'.

The most important criterion, according to Jacobson et al, for determining a 'good object' is

'that is should be robust against modification and help the understanding of the system' (Jacobson et al. 1998).

This is no help to analysts working on a new problem domain or new system as it implies that 'good' objects are only recognised after a period of time has elapsed and modifications made. The 'good' objects are those that remain or have evolved.

All information systems that are built will be modified at some point in time, the OOSE methodology creators assert that in order to accommodate these changes a robust model must be created initially. Jacobson et al. (1998) claim

'after we have worked with a model for a while, a stable structure will evolve for the system'.

How is a stable structure identified? Who determines whether the structure is stable? The methodology creators give no guidance or criteria on which to base the definition of stability. Jacobson et al (1998) assert that

'stability also depends on the fact that modifications often begin from some of these items (essential and always exist)'.

It is unclear from the text what timescale is being referred to, as in some instances modifications clearly refer to those made over a long period of time,

'by working a long time with the early models, we (system developers) will obtain a good understanding of the system' (Jacobson et al. 1998).

Here the system refers to an information system to be designed and implemented in its nth version. The implication here is that only time will tell if you have created a 'good' object model. This opinion is based on the methodology creators experience of block based design techniques used within the telecommunications industry.

According to Jacobson et al. (1998) use cases may be viewed as objects as they describe the interactions between an actor and the system, this is explained as the actor invoking new operations on a specific use case.

The methodology users refer to the initial objects that are identified from the original requirements as problem domain objects, having developed a requirements model and had it signed off by the client the analysis model is then started. The analysis model describes the system using three different types of object; entity object, interface object and control object, which the methodology creators argue allows for an overview of the system. The methodology creators argue that by identifying only one type of object, for example, Coad and Yourdon (1991) only a simple or general model can be created. Developing the analysis model entails distributing the behaviour specified in the use case descriptions among the objects in the analysis model. In this way the behaviour each object is responsible for in the use case can be explicitly stated.

Entity objects model the information in the system that should be held for a long period of time, these objects should survive particular use cases. The methodology creators advise the methodology user to consider the needs of the use case as a guideline for justifying those entity objects that should be included. Jacobson et al. (1998) state

'most entity object are found early and are obvious. These 'obvious' entity objects are often identifies in the problem domain object model. Entities usually correspond to some concept in real life. The hard thing is to model only the entity objects actually needed'.

There is an assumption of a shared understanding of the problem domain and language being used, evidenced by the assumption that entity objects are obvious. The

methodology creators state that only the entity objects that can be justified from the use case descriptions should be included. As methodology users we should look at the information that must be kept for a long period of time as a source of 'needed' entity objects. Examples of using use cases to identify entity objects are given in the text see below, italics indicate information functionality.

> 'When the customer returns a deposit item, it is measured by the system. *The measurements are used to determine what kind of can, bottle or crate has been deposited.* If accepted, *the customer total is incremented, as is the daily total for that specific item type.* If the item is not accepted, the light for NOT VALID' is highlighted on the panel. When the customer presses the receipt button, the printer prints the date. The *customer total is calculated and the following information printed on the receipt for each item type:*
> *name*
> *number returned*
> *deposit value*
> *total for this type*
> *Finally the sum that the customer should receive* is printed on the receipt' (Jacobson et al. 1998).

From this information the methodology creators reason that because we need to remember how many cans, bottles and crates of each type have been deposited each day, we need something that handles this information. The entity objects; can, bottle and crate, are therefore identified. These entity objects can also handle size, deposit values and other information tied to these objects, because some of the information is common to these entity objects we can extract the common properties (i.e. deposit value, day total) and place them in an abstract entity object, deposit Item (Fig 7.3).

This is a relatively simple example that the methodology creators expect the readers to understand without further explanation. The same entity object may be identified in other use case descriptions. The methodology creators expect the methodology users to transfer the technique used in these relatively simple scenarios to the application area under investigation but no additional help is given as to how this should be conducted. The methodology user is warned that an entity object identified in one scenario can be an attribute in another scenario.

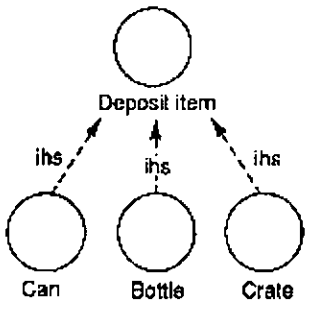


Figure 7.3 Can, bottle and crate have common properties inherited from Deposit Item

(adapted from Jacobson et al. 1998)

For example: we need to model and car and its owner. Are these two separate entity objects? In one system we can develop two entity objects, Car and Owner. In another system we could have an entity object Car with an attribute owner, in yet another system we could have and entity object Person with an attribute car. We need to consider the use cases in order to determine the most appropriate representation. In the first case, two use cases could use Owner and Car independently of each other, in the second Owner is always used in combination with its owned Car, and in the third case Car is only used in association with a specific owner or Person.

Interface objects describe communication between the system and the user. The methodology creators assert that interface objects can be identified easily as they can be clearly identified from the system interface descriptions that accompany the requirements model. In addition they usually start from an actor and describe what the actor does and how they communicate with the system. An example of an interface object is the user interface functionality for requesting information about a person. Jacobson et al (1998) state that

' interface objects are quite simple to identify'

and they propose three strategies for identifying them.

1. 'Either they are clearly identified from the system interface description accompanying the requirements model
2. or we can start from the actors

3. or we can read the use case descriptions and extract functionality that is interface specific'.

In order to demonstrate the strategies the methodology creators use a recycling machine problem scenario and from this identify 'concrete' actors; *customer* and *operator* each of which needs its own interface object. The customer needs the panel with push buttons and slots in which to insert items to be recycled. The operator needs an interface to change information in the system and to generate daily reports. An interface object to call the operator when an alarm is raised is also needed, as is an interface to print receipts. These interface objects are represented diagrammatically as Fig 7.4.

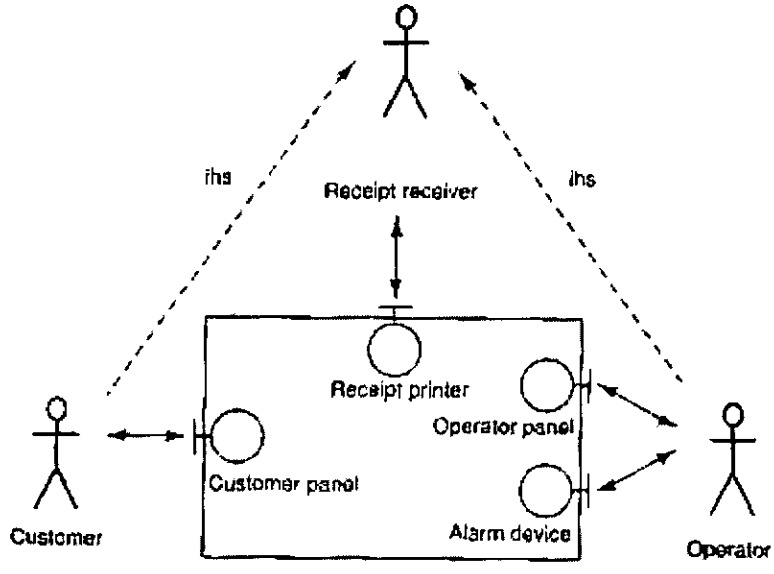


Figure 7.4 Interface objects in the recycling machine, customer panel, operator panel, receipt printer and alarm device

(adapted from Jacobson et al. 1998)

Both of these strategies rely on the methodology users understanding of the problem situation and a shared understanding, by users and methodology users, of what a recycling machine needs in order to operate. No further explanation is given by the methodology creators to expand on how these strategies should be applied.

The third strategy, identifying interface objects from the use case is demonstrated by an example from the recycling machine where interface functionality is marked in the textual description. For example:

'When the customer *returns a deposit item, it is measured* by the system. The measurements are used to determine what kind of can, bottle or crate has been deposited. If accepted, the customer total is incremented, as is the daily total for that specific item type. If the item is not accepted, *the light for NOT VALID' is highlighted on the panel.* When the customer *presses the receipt button, the printer prints the date.* The customer total is calculated and the following information *printed on the receipt* for each item type:

name

number returned

deposit value

total for this type

Finally the sum that the customer should receive is *printed on the receipt.*'(Jacobson et al. 1998)

From these italicised areas the same interface objects, customer panel, operator panel, receipt printer and alarm device are identified. By using the three strategies together the methodology user can double check that all reasonable interface objects have been identified, although the methodology creators do not explicitly recommend the methodology user to do this.

Control objects, according to the methodology creators, are 'the most ephemeral and last only as long as the use case'. Control objects usually contain the behaviour that does not belong to the interface object or to the entity object. An example of a control object is calculating taxes using several different factors. This behaviour could be placed on any of the other types of object but the actual behaviour does not belong to any specific entity object or interface object. Jacobson et al. (1998) give some heuristics to help find and specify control objects. Each use case normally involves interface objects and entity objects, any behaviour that remains after the interface objects and entity objects have been assigned their behaviour will be placed in the control objects. Usually one control object is assigned for each concrete and abstract use case. Typical types of functionality placed in control objects are transaction-related behaviour, or control sequences specific to one or more use cases, or functionality that separates the entity objects from the interface objects. The same use case description is used to identify functionality typical of control objects, see below.

'When the customer returns a deposit item, it is measured by the system. *The measurements are used to determine* what kind of can, bottle or crate has been deposited. *If accepted*, the customer total is incremented, as is the daily total for that specific item type. *If the item is not accepted*, the light for NOT VALID' is highlighted on the panel. When the customer presses the *receipt button, the printer prints* the date. The *customer total is calculated* and the following information printed on the receipt for each item type:

name

number returned

deposit value

total for this type

Finally the sum that the customer should receive is printed on the receipt.'(Jacobson et al. 1998)

This is a concrete use case that inherits an abstract use case Print. The first option, one control object for each (abstract and concrete) use case would give two control objects. Returning Item is a use case that involves coupling the interface objects and the entity objects together – a good candidate for a control object. The methodology creators do not assign a control object to Print because the behaviour, printing to a line printer, should be tied to the interface object.

The methodology creators state that

'descriptions of other control objects can be found indirectly from the use cases by checking how the use case runs over the other object types'

however there is no further guidance on how to carry out this exercise given in the text.

Each use case description is examined in order to identify entity objects, interface objects and control objects. As new objects are identified previous use cases are re-examined in order to verify decisions already made. When all the entity objects, interface objects and control objects have been identified for a use case description they are combined in a use case view which shows how the objects will be used in the use case Fig 7.5

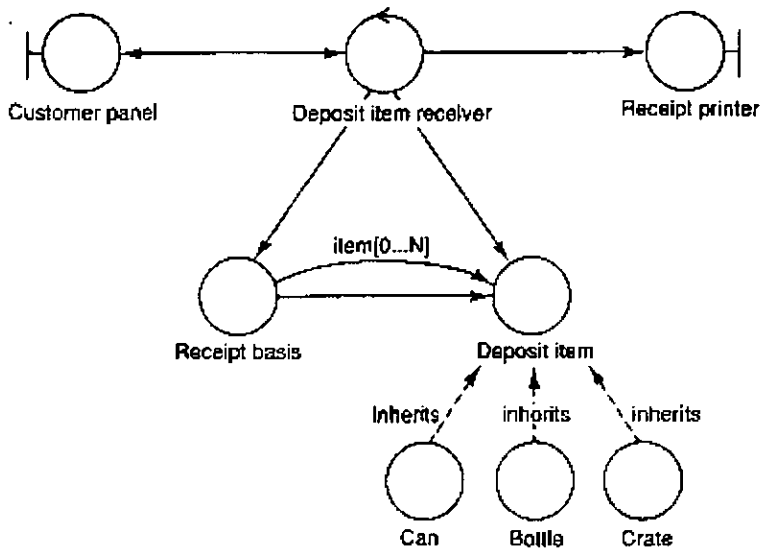


Figure 7.5 The objects supporting the use case Returning Item

(adapted from Jacobson et al 1998)

According to Jacobson et al (1998) the analysis model gives a

> 'conceptual configuration of the system, consisting of control objects, interface objects and entity objects'.

The purpose of this model is to develop a robust and extensible structure as the base for construction.

7.2.3.3 Stage (iii) Defining the prognosis outline

By 'defining the prognosis outline' NIMSAD means that the methodology should be able to facilitate the identification of a clear defined 'desired state' in relation to the 'situation of concern'. Jayaratna (1994) expresses this as defining 'where we want to be and why?'

OOSE does not make its users aware of how to perform the diagnosis, nor does it help them to understand or question the rationale for the clients 'desired states'. OOSE relies on the client being clear about the rationale for the requirements. It does not offer any steps for examining the validity of these expectations.

7.2.3.4 Stage (iv) Defining problems

NIMSAD uses this stage to investigate what the methodology can offer in terms of providing an answer to the question ' What are the problems which currently prevent

the transformation of the 'problem situation' from its 'current state' into the desired state'?

As OOSE does not concern itself with the clients 'desired states' (prognosis outline) it has no means of defining problems either. OOSE spends time with both clients and users discussing use cases in order to create a requirement specification that can be 'signed off'. The reasoning behind these requirements is never explicitly considered. OOSE is not dealing with the relevance of problems but the construction of solutions to requirements.

7.2.3.5 Stage (v) Deriving Notional Systems

NIMSAD defines notional systems as 'those systems which need to be developed if the client organisation is to overcome the previously defined 'problems', thereby helping to transform the 'current state' to the 'desired state' (Jayaratna 1994). This means identifying systems using the 'mental construct', which, if designed and implemented, would eradicate one or more of the perceived problems with the current system.

Documenting a requirements specification according to NIMSAD belongs to 'deriving the notional system'. The OOSE creators presume, as a general case, that a comprehensive enterprise development analysis has been made. As OOSE does not help its users to participate in the definition of 'desired states' it cannot define problems, nor can it arrive at notional systems, these are implicitly found in the minds of the users.

7.2.3.6 Stage (vi) Performing Conceptual / Logical Design

The logical and conceptual design of a system is intended to formalise the structures, roles, tasks, functions, information and attitudes of the notional system(s) into a form that represents a clear, logical method of achieving the ideas identified in the notional system(s).

In addition to use cases, a requirement model also documents a problem domain object model and interface descriptions. The requirements model is then used to produce a robust analysis model. The OOSE creators stress the need to maintain an updated analysis model. Such a model documents the logical design of an information

system. They stress that this analysis model has to be kept free from consideration of its physical implementation. They reason that an implementation independent analysis model gives a better starting point for later versions, which may be implemented on a new technical platform.

7.2.3.7 Stage (vii) Performing the Physical Design

The physical design of a system is the process of transforming the logical / conceptual design into a form suitable for implementation (given a set of identified constraints). Solution design is based on the work of the problem formulation phase (Jayaratna 1994). The reason for this lies in the fact that an information system alone cannot increase the effectiveness of an enterprise or any part of it. Beneficial effects have to be co-produced by actions of managers, system users and other stakeholders. In other words an information system has to contribute to a resolution of a business problem.

The OOSE construction process is intended to give a design model and an implementation model. The design model has to take the implementation environment into consideration. This corresponds to the stage of physical design within NIMSAD. Without documented diagnosis and prognosis models the OOSE methodology users lack some of the documents required for their implementation planning.

7.2.3.8 Stage (viii) Implementing the Design

The implementation of a system is the realisation of the notional system(s) within the 'situation of concern' (Jayaratna 1994).

In order to support an industrialised form of software production, the construction process aims at building a model of an information system, which describes it in a way that allows it to be built largely by assembling existing components. In undertaking this design, the OOSE creators instruct methodology users to do this in a way that makes the new components exhibit functionality and be useful as building blocks in other systems as well. The keyword here is reuse. Reuse is generally taken as the use of results from earlier development work, including code reuse and documentation reuse. In OOSE the suggested reuse 'should occur on several different levels of granularity' (Jacobson et al. 1998). The OOSE methodology creators discuss reuse at the lowest level, which consists of components from which to build application modules. Reuse is not considered in NIMSAD.

7.2.4 Element 4: Evaluation

'It is the evaluation which helps us to measure the effectiveness of the problem solving process and the problem solver in the 'problem situation' – unless the element is considered there is no way of establishing that the 'problems ' have been successfully resolved' (Jayaratna 1994).

The OOSE methodology does not advise its users to evaluate the effectiveness of the methodology in use, nor does it offer models/criteria to probe into whether the business problem is given adequate consideration. The NIMSAD framework gives serious consideration to this problem.

The OOSE methodology creators compare their methodology with other object-oriented system development methodologies Booch (1991), Coad and Yourdon (1991), Rumbaugh et al. (1991), and Wirfs-Brock and Wilkerson (1989). The aim of these comparisons is to influence potential users of the OOSE methodology. Their comparative assessment is less comprehensive than that suggested by NIMSAD framework and is mainly at the descriptive level. The OOSE methodology creators do not consider assessing the methodology before, during and after application.

7.3 Summary

This chapter began with a brief introduction to OOSE and then evaluated OOSE using the NIMSAD framework with an emphasis on object identification. The main findings from this evaluation are as follows.

OOSE takes a development approach that is used in the construction industry and applies the same principles to information systems development. The underlying enterprise philosophy implies that tools provide support for activities in three categories: architecture, method and process. The development of software systems is presented as analogous with the construction industry. Development is progressive, involving iterative cycles of analysis, construction and testing.

The ability to apply object-oriented thinking is identified, by OOSE methodology creators, as being important when using their methodology. Object-oriented thinking

integrates static and dynamic aspects of entities. It is a change for methodology users from function-oriented system development. OOSE purports to teach its methodology users an object-oriented view by introducing fundamental concepts of object-orientation and applying them in different cases. The OOSE methodology creators alert the methodology user to the importance of object-oriented thinking but they do not consider whether the system users should also be made aware of the differences between function-oriented thinking and object-oriented thinking.

OOSE creators suggest that system developers should develop information systems with the system users and not the clients. The co-operation they stress is only between system developers and users. Conflicts between clients' requirements and those of the users could be ignored. The OOSE methodology does not give any reasons why the methodology users should take the system user's perspective, nor does it instruct its users how to handle conflicts with respect to a requirements specification between them and the clients, despite explicitly mentioning this type of conflict.

The methodology creators advise their methodology users to take clients' requirements specifications only as a starting point. According to OOSE receiving the requirements specification, in some form, is the starting point for systems development. At an early stage of systems development, the main 'problem', according to OOSE, seems to be producing a requirements specification precise enough to become part of a contractual agreement between a client and a software producer.

This may create a problem if the client is not very familiar with the work situation of people within the area of use. The delivered information system, in most cases may not be used by the client, but by a number of indirect users. For this reason OOSE alerts its users to the importance of understanding the users information requirements, rather than those of their clients.

All models that apply OOSE are use case driven. Use cases denote descriptions of the functionality expected from an information system by its potential users in their different interactions with it. The importance of use cases lies in the idea that they can be controlled by users and taken by system developers as part of the requirement

model. The use cases and how they are connected are described in a special 'use case model'.

OOSE relies on the identification of nouns, used within the application domain, to identify relevant objects. According to Jacobson et al. (1998) analysis of the terminology allows the methodology user to identify candidate objects in the system, these can then be evaluated to determine how the system needs to handle the objects, inclusion of the object in the final model is then based on this evaluation. This assumes a shared understanding of the problem domain and a common set of norms and values. These are not discussed within the text nor is the methodology user made aware of them.

OOSE methodology creators do not perceive identifying objects as difficult. The difficulty arises in selecting objects relevant to the system. OOSE methodology users are advised to put the candidate objects into context in order to determine whether it is an appropriate object. What is really of interest is how an object works with other objects and under what conditions.

The most important criterion, according to Jacobson et al, for determining a 'good object' is 'that it should be robust against modification and help the understanding of the system' (Jacobson et al. 1998). This is no help to analysts working on a new problem domain or new system as it implies that 'good' objects are only recognised after a period of time has elapsed and modifications made. The 'good' objects are those that remain or have evolved.

Each use case description is examined in order to identify entity objects, interface objects and control objects. As new objects are identified previous use cases are re-examined in order to verify decisions already made. When all the entity objects, interface objects and control objects have been identified for a use case description they are combined in a use case view which shows how the objects will be used in the use case.

OOSE does not make its users aware of how to perform the diagnosis, nor does it help them to understand or question the rationale for the clients' 'desired states'. OOSE

relies on the client being clear about the rationale for the requirements. It does not offer any steps for examining the validity of these expectations.

As OOSE does not concern itself with the clients' 'desired states' (prognosis outline) it has no means of defining problems either. OOSE spends time with both clients and users discussing use cases in order to create a requirement specification that can be 'signed off'. The reasoning behind these requirements is never explicitly considered. OOSE is not dealing with the relevance of problems but the construction of solutions to requirements.

Chapter Eight: Finding Objects in Practice.

Having critically evaluated a number of object-oriented analysis and design methodologies in order to determine how they recommend users to define / identify objects the next task was to determine how users identify / define objects in practice. This would answer the part of the research question that asks 'how do we identify objects?' and 'how do we derive objects from the requirements determination?' An exploratory phenomenological approach (see 2.2.6) was taken to conduct 6 semi-formal interviews. The aim of the interviews was to identify how the target group thought they identified objects and by using a case study scenario determine what they do in practice.

The participants were known to the researcher; the advantages of this were:

- less explanation was required immediately prior to the expert's account; therefore there was less danger of influencing the outcome.
- Participants were more motivated to give a complete account because they wanted to help the researcher.

The disadvantages were:

- a less representative sample was obtained, however see 8.1.
- a smaller range of object-oriented methods was acknowledged, as all participants were associated with the same university.

Each participant was interviewed twice (see 8.2).

- On the first occasion to discover how they thought they identified objects
- On the second occasion to discover whether they used the processes they had described during the first interview when given a complex scenario.

This chapter is based on the results of 6 semi-formal interviews with 3 academic colleagues and 3 practitioners who use object-oriented analysis methodologies as part of their teaching or normal working practices.

Differentiation between software engineers and systems analysts has been made, but not, in general, between academics and practitioners, as there were few significant differences in the results.

8.1 Sampling

My sample was serendipitous and consisted of three academics and three practitioners and both software engineers and systems analysts. Table 8.1 identifies the variables within the sample. One academic was an analyst the other two software engineers; one practitioner was a software engineer the other two systems analysts. My sample also consisted of three females and three males. The sample was as varied as possible given the size. The participant's age range was late 20's to mid 50's with an even spread between academics and practitioners.

	Academics	Practitioners
Software Engineers	2	1
Systems Analysts	1	2
Female	2	1
Male	1	2

Table 8.1 Population sampling

There was no correlation between these variables. The female academics had previously been practitioners and one of the male practitioners has previously been an academic.

8.2 Investigation Method

The participants were involved in two sessions, each lasting an hour. The initial session was a semi-structured interview (Smith 1995) during which the participant was asked how they identified objects. The purpose of this was to determine whether practitioners used approaches recommended by various object-oriented analysis and design methodologies or other methods. Participants were given the choice of writing down their thought processes, explaining them while I made notes or a combination of the two.

For the second session, the participants were given a complex scenario to read and model. Participants were asked to identify the objects from the scenario. The purpose here was to determine whether participants identified / defined objects from the scenario using the processes they had described in the first session. In other words did

the participants actually do what they said or thought they did? Observational methods (Coolican 1990) were used as a technique for this part of the investigation as naturally occurring behaviour was being observed.

8.3 Raw Data

Table 8.2 indicates the issues raised by the participants during the interviews. I have not differentiated between the first and second interview nor have I differentiated between academics and practitioners, as there were few significant differences in the results.

	Software Engineers	Systems Analysts
Looking for nouns	3	3
Relevant things/entities	3	3
Events	2	3
Attributes/objects	2	3
Relationships	1	3
Divide into 'sub-systems'	1	3
Textual description	2	2
Object model	3	1
Code-ability	3	-
Iteration/redraw	-	3
Boundary definition	-	2
Operations to be carried out	3	-
ER modelling	1	2
Abstraction	2	1
Formal methodology	1	-
Questions for clarification	-	1

Table 8.2 Raw Data

8.4 Analysis

Having briefly described the techniques used for this investigation and tabulated the raw results I shall now evaluate the responses using the NIMSAD framework. This

will ensure that the same criteria used to evaluate the methodologies are used to evaluate the participant's responses.

8.4.1 Problem Situation

All participants assumed they were working from a written description of the problem situation. All participants began by looking for nouns or 'things' that make sense in the problem domain. They used arbitrary judgement to decide which objects were outside the problem situation and therefore unimportant to the current model. One analyst stated she would produce a list of questions / queries/issues that would need to be clarified before an acceptable model could be produced. The software engineers took an ontological perspective of the given scenario and did not question the boundaries of the problem situation. They all assumed the requirement specification, as given, was as complete as was necessary.

The problem situation from an academic perspective is bound by the description given, further exploration of the problem domain was not considered. This is due mainly to the teaching situation, whereby material presented to students' has, in general, tightly defined boundaries. Looser descriptions would leave much of the problem situation 'open to question', which would be difficult to handle in the teaching context.

8.4.2 Problem Solver

Only one of the participants stated he would use any formalised methodology - UML. The other participants relied on their own experience of analysis methods: be they traditional structured methods or object-oriented methods, no one explicitly stated the methodology they were using.

Those participants with software engineering backgrounds, selected objects on the basis of code-ability, that is on identifiable state or behaviour that would ultimately need to be coded. This indicates a reliance on experience of similar scenarios or problem domains, whereby seemingly relevant 'chunks of code' could be identified. There was an implication that one of the criteria for identifying relevant objects was the ability to code the actions with which the object could be associated.

The analysts were not interested in code-ability they were more interested in the relationships between objects. The analysts stated they would redraw the model a number of times and re-arrange objects and relationships in order to clarify them. They would also re-visit the description and clarify issues in order to identify extra objects / attributes / operations.

8.4.3 Problem Solving Process

The analysts selected individual areas of the case study to develop/model in depth. Having created a model for each identified sub-system they linked them together.

> 'I look for things – entities I can see, then I look for events – entities that happen at a particular time, these might be objects or attributes of the thing the event happens to. Then I construct an ER diagram, I look for entities that have attributes in common, for example: people, places, transactions and make super-objects/classes from these' (Analyst 1).

> 'I read through the description and list the main objects (as I think). I'm probably looking for nouns, but I would filter out repetitions and what I perceive to be attributes and nouns that represent the 'system'. I start to develop a diagrammatic model from this list, then I go back and check the model against the description to find missing elements and to look at cardinalities of relationships' (Analyst 2).

The analysts did not initially consider the operations to be carried out by identified objects. These were considered to be part of the design phase and as such were left until an initial model had been developed. As the model was developed the relationships between objects were continually revisited until the analyst felt that the model was 'accurate'.

The software engineers identified central objects with primary operations. Other objects were then identified and relationships with the central objects identified.

> 'I list candidate classes from descriptions supplied from requirements capture, especially from use cases, - i.e. I look for noun phrases. I rule out candidate classes that are outside the system being modelled, synonyms of other candidates and those that are just plain silly' (Software Engineer 1).

The criteria being used to rule out classes that are outside the system being modelled were not identified and by implication they relied on 'experience'.

> ' I look for 'things' usually specific nouns; either common (groups) or proper – specific objects rather than abstract objects. Not everything will ultimately be included, things that are trivial – usually scalar (single fields i.e. colour) and with no obvious significant operations will become attributes of objects. Once a group of potential classes has been identified I look to see if abstracting out common properties/operations can sensibly identify abstract classes. There may be other things that make sensible classes, for example: if the problem involves a group of objects, it may make sense to have a collection class representing that group. Relationships between classes may give rise to additional classes, i.e. records containing details of a transaction. Additional objects/classes may be identified when considering the user interface' (Software Engineer 2)

One software engineer stopped himself during the modelling process and stated 'that's wrong I'm supposed to be using UML'. This was the only participant to claim to be using any formal object-oriented methodology

All participants stated they would look for nouns or 'things' within the stated problem situation. Effectively participants selected 'important things' from the problem domain case study, i.e. those things that had some relevance to the case study from the participants' own perspective. Academics described these 'things' as 'nouns' in order to justify their selection and ultimately in order to assist students in selecting appropriate objects. Most 'important things' were expressed as concepts that made it easy for the participants to formulate them into an object model according to the noun-selection principle. The 'important things' were then re-considered in order to create object classes.

Some methodologies i.e. Wirfs-Brock and Wilkerson (1989); Coad and Yourdon (1991); Rumbaugh et al. (1991) and Booch (1994) recommend object identification from text descriptions of the problem situation. The nouns represent objects and the

verbs represent operations on the objects and relationships between objects. These methodologies reveal something in common:

- Nouns are keys to objects
- Text descriptions are a source of needed objects
- Systems analysts define object classes. They act as object selectors.

Everything is to be viewed as an object in an object-oriented system. Different people, for example: system users and developers understand the concept 'object' from their own perspective. The system users field of activity can be seen as an organisational context from which they select concrete and abstract 'things' i.e. products to be sold, the concept 'sales' appears to be more abstract than the product sold. System users understand their business in terms of concrete and visible things.

'Important things' express unique insights, useful in defining objects and object classes. 'Things' are system users views on 'objects' and are acceptable for their requirements. 'Objects' are defined by developers from a systems development perspective. The defined objects satisfy the needs of the requirements specification in respect of analysis, design and implementation.

In theory and practice 'important things' are not exactly equal to objects in the field of object-orientation. According to the applied methodology, systems analysts determine where those things should go – i.e. what kind of model. Different things can be formulated into different models for different purposes, i.e. object model, function model and control model of OMT, which things are included in the object model depends on how significant those things are to the system and not just to the system users' field of activity. Systems analysts select from the 'important things' those objects that are to be considered as classes.

The aim of finding 'important things' lies in defining the right objects for an object model. The essential and important are selected from the non-essential and unimportant. A model only integrates the essential part according to its closeness to the purpose of modelling a system. The problem associated with the question of what is and is not 'essential' is one of understanding and semantics, and for research

purposes permits us to say that importance is relevant to what the system users want and expect from a system.

Picking 'important things' differs from 'drawing out nouns' even though 'important things are expressed in the form of nouns. Noun drawing is based on the requirements specification, which has already been constructed by someone else such as managers. Systems analysts pick up the nouns and define them in object models. A requirements specification in this sense should not be taken 'as given' but as a way of opening communication between system developers and users.

8.5 Lessons Learned

The sample is too small to draw any reliable generalised conclusions, but the results do indicate that there is scope to conduct a larger scale study involving both academics and practitioners.

The results indicate that both academics and practitioners rely on nouns or 'important things', within the written description, as a source of objects. Experience in using a methodology or familiarity with a problem domain are relied upon to abstract relevant classes from the listed objects. Software engineers were different from systems analysts in that they considered code-ability, the operations to be carried out and worked directly with object models, although only one identified a formal methodology being used. Systems analysts were more concerned with the data and relationships and went through more iteration before they were satisfied with their model; they also questioned the boundary definitions.

Chapter Nine: Conclusions

Having examined three seminal texts (chapters 5, 6 and 7) and conducted 6 semi-formal interviews using an exploratory phenomenological approach (chapter 8) this chapter begins by identifying the major findings. In the following three sections I expand on these conclusions:

- from a methodological perspective
- from the evaluated methodologies
- from the use of the NIMSAD framework.

The chapter concludes with a discussion of future work.

9.1 Major Findings

This work supports the claims that object-oriented representations are more natural:

- nearer to the real world 'things' being modelled
- nearer to natural human ways of thinking about them

An object-oriented analysis approach focuses on objects, their classification, generalisations and specialisations. (Coad and Yourdon 1991; Rumbaugh et al 1991; Embley et al 1992; Booch 1994; Conger 1994; Elmasri and Navanthe 1994; Graham 1995; Kroenke 1995; Norman 1996). The identification of objects is therefore essential if we are to develop realistic models of the problem domain.

This work does not support the claim that:

- object-oriented approaches are a radically different way of thinking about problems.

The researcher finds this claim neither proved nor disproved but her current view is:

- object-orientation is the recommended approach to data modelling (Graham 1995; Booch 1991; Booch 1996);

Traditional structured analysis methods (Yourdon 1991; DeMarco 1979; Cutts 1987) focus on the business functions; data analysis methods focus on the flow of data and the structure of the data files within the organisation context. The normalisation process removes dependencies and data duplication so that re-creation of an entity can

involve joining multiple tables (Connolly 1995; Codd 1970; Chen 1994). An object-oriented approach is seen as more 'natural' as we think in terms of objects.

- object identification is done by practitioners (Coad and Yourdon 1991; Rumbaugh et al 1991; Jacobson et al 1998), but the methodology creators, teachers and researchers cannot clearly say how it is done

When the object-oriented literature does offer suggestions on how to identify objects and entities, they are drawn from the 'craft knowledge' of experienced practitioners rather being derived from any theory of how human beings make sense of their world or inquire into problem situations (Coad and Yourdon 1991; Rumbaugh et al 1991; Jacobson et al 1998). The danger of this is that the experienced user relies on previous practice and application areas and may 'see' an 'obvious solution' without exploring the problem situation in sufficient detail.

- texts do not differentiate between objects and entities

In much of the object-oriented literature the concept of 'object' is described, in that objects have identity, state, behaviour and properties and can be characterised by a set of operations that can be performed on it or by it and its possible states (Shlaer and Mellor 1988; Coad and Yourdon 1991; Rumbaugh et al 1991; Embley et al 1992; Booch 1994; Booch 1996; Jacobson et al 1998). In many cases objects are perceived as being the same as entities identified by traditional analysis and design methodologies.

The lack of attention given to the question of how to identify entities and objects may be explained by the original use of data analysis in the design stages of development, where it is relatively straightforward to identify the 'things' about which data are required to be held. The use of data analysis is shifting to earlier stages of development where the problem is less well defined. There is a continued reliance on a pragmatic 'common sense' approach to identify objects (Lewis 1994). The 'common sense' of our tradition is that in order to perceive and relate to things, we must have some context in our minds that corresponds to our knowledge of them. Organisational

culture may give relevance to items/things/entities/objects that appear strange to 'outsiders'.

Literature relating to identification and formulation of objects is very sparse although sections purporting to cover this are found in textbooks relating to either specific object-oriented methods or to object-orientation in general be that from a software engineering perspective, an analysis and design perspective or a database perspective, for example, (Shlaer and Mellor (1988), Goodman (1989), Khoshafian and Abnous (1990), Booch (1991), Coad and Yourdon (1991), Rumbaugh et al. (1991), Bertino and Martino (1993), Conger (1994), Graham (1995) Jacobson et al. (1998)). How objects are identified depends very much on the authors' perspective and can vary from using entity and object interchangeably to the implication that identification of objects is not a problem, as everyone implicitly knows what objects are.

- it is important that we understand how we identify objects

'Object' is a fundamental concept of object-orientation. For any object-oriented method the single most crucial aspect of the analysis and design phase is the determination of which objects and classes to include in the model. It requires a clear vision of which objects belong to the system to produce a relevant object-oriented model (see 1.1).

Inspection of the current literature revealed that the methods for identifying and formulating objects are based on natural language descriptions of the required system. This introduces many inherent ambiguities as it depends on the writer's linguistic ability as well as the designer's knowledge and experiences in the problem domain (Ramachandran and Taylor 1994). Some texts, for example Meyer (1988) and Shlaer and Mellor (1988) suggest that there is no problem in identifying objects. This is difficult to reconcile with observations of novice and inexperienced data analysts who frequently find great difficulty in identifying a set of 'relevant' entities or objects (Lewis 1994).

Identification of nouns as a 'first pass' method of identifying objects is the same approach taken by traditional structured analysis methods where nouns and verbs are selected to identify entities and processes.

- Object-orientation advocates a different way of thinking about data.

As many of the creators of object-oriented analysis and design methodologies initially developed structured analysis methods one assumes they are relying on this past experience when identifying objects. This would suggest that they are using approaches they are familiar with without understanding what principles are being applied and why. Jacobson et al (1998) explicitly advocate an object-oriented way of thinking but do not expand on how this can be achieved; neither Coad and Yourdon (1991) nor Rumbaugh et al (1991) explicitly discuss object-oriented thinking.

If we cannot formulate a method for thinking in terms of objects we neither understand what we are advocating nor can we transmit our new insight to others.

- It is difficult to improve the performance of object-oriented models if we do not understand the method used to produce them

The methodologies considered used an object-oriented approach to model the current problem situation, none of them explicitly stated how by using an object-oriented approach any problems would be solved.

- It is difficult to teach by relying on craft knowledge

If worked examples are used students attempt to identify a 'pattern' or unstated approach being used and then transfer this approach to future problems. Reverting back to familiar methods such as nouns and verbs is seen as a 'safe' approach.

- In the real world there are many potential objects and classifications. How you find them depends on shared understanding of the problem domain, context

and requirements. How you chose relevant objects and classifications depends on the context.

The following sections expand and discuss the conclusions I have reached.

9.2 Conclusions on object formulation from a methodological perspective

Object identification is an essential area of any object-oriented analysis methodology. In OOA object identification is restricted mainly to descriptive statements and 'education by example' i.e. methodology users learn object modelling through different application examples. Nouns are selected that are seen to be relevant to the problem situation. These potential objects are related mainly to the data requirements of the stated problem situation. The approach taken by the OOA methodology creators indicates that the methodology user should determine what they could about the problem domain and problem situation by asking questions. Clients, users and other stakeholders should be consulted as should any documents; invoices, receipts, reports etc. This is no different to the approach taken by structured systems analysis methodologies.

Guidance on object identification in OMT is restricted to examples given in the text and left to the experience of the methodology user. OMT has specific stages devoted to object identification, however much of the text is devoted to descriptions and guidelines of what an object is. These rely heavily on identification of nouns which is the same approach taken by many of the traditional structured analysis methodologies when identifying entities. Additional classes, according to OMT, can be identified from knowledge of the problem domain. However, no explicit help is given to help the methodology user identify these additional classes. It is implied that a thorough understanding of the problem domain and experience, in both the problem domain and the methodology, will enable the intended problem solver to determine appropriate objects.

OOSE relies on use case scenarios to identify entity objects, control objects and interface objects. The methodology user enlists the support of various system users in order to identify the various objects. Initially noun phrases are used to identify objects.

The identification of nouns, used within the application domain, is making use of selected areas of textual analysis although the OOSE methodology creators do not refer to this. According to Jacobson et al. (1998) analysis of the terminology allows the methodology user to identify candidate objects in the system, these can then be evaluated to determine how the system needs to handle the objects, inclusion of the object in the final model is then based on this evaluation.

The most important criterion, according to Jacobson et al, for determining a 'good object' is 'that is should be robust against modification and help the understanding of the system' (Jacobson et al. 1998). This is no help to analysts working on a new problem domain or new system as it implies that 'good' objects are only recognised after a period of time has elapsed and modifications made. The 'good' objects are those that remain or have evolved.

There is no guidance given to the methodology users as to how one identifies which are relevant objects in a problem domain. The relevance or irrelevance is based on the methodology users' judgement and opinion, which is very subjective and depends, to a large extent, on the methodology users' understanding of the problem domain and on the shared understanding of the client, methodology user and other stakeholders.

All three methodologies choose noun phrases as a way of identifying objects. Thus the requirements specification becomes an original source for objects. Each methodology involves clients, users and other stakeholders, to some extent, in clarifying the requirements specification. However after the initial input from clients and other stakeholders the methodologies rarely consult them again. The models are rarely checked for accurate reflection of the requirements or understanding of the problem.

Classification of the initial objects, according to Coad and Yourdon (1991), is based on concepts that are learned in kindergarten: objects and attributes, wholes and parts, classes and members. The selection of those objects relevant to the problem situation

is seen as a 'natural ability' of the methodology users and is implicit in OOA. OOA does not alert the methodology user to this nor does it offer any assistance on how to apply classification theory. Instead OOA relies on the methodology users experience and knowledge to determine appropriate classifications.

Abstraction is viewed as an important theme in object orientation by the OMT methodology creators and is perceived as a fundamental human capacity that permits us to deal with complexity. Abstraction focuses on the individuals' competencies and reasoning abilities. OMT does not alert the methodology user to the importance of reasoning ability and abstraction. The text does not assist the methodology user in developing their abilities in abstraction. Rather abstraction is seen as an innate ability of the methodology user.

OOA identifies the steps to be taken when applying the methodology and gives examples in the text, but there is no other guidance on how the methodology user can develop their experience. Experience is implicitly assumed to be gained by working through the given examples and applying the strategies to the target problem domain.

The experienced user plays an important role in OMT, being able to select appropriate objects from the problem domain intuitively and then differentiate between relevant and irrelevant objects without writing anything down. The danger of this is that the experienced user relies on previous practice and application areas and may 'see' an 'obvious solution' without exploring the problem situation in sufficient detail. Despite the emphasis on experienced users OMT methodology offers no explanation of what might facilitate the transition from a new to an experienced user.

Experience is an invaluable source for developing knowledge and skill, and helps to form implicit models for structuring our understanding of situations. The more experience we have of a particular working environment, the easier it may be for us to assess similar situations. Experience makes us more confident and enables us to assume 'expert' status. However, while experience-based models may reduce time, provide a range of easy and 'obvious' solutions and help to develop confidence, the same models may prevent us from exploring new ideas or becoming effective listeners

to others' ideas. OOA relies on the methodology users considerable experience to gain an impression of the state of the situation.

The OOA methodology takes an ontological view of the problem situation in so far as the boundary of the problem situation is not explored but taken as given. The given requirements are discussed with a variety of stakeholders and existing documents are consulted in order to determine what is being modelled. The rationale for the requirements is not considered by the OOA methodology as a result of this there is no clear 'desired state'. To this end OOA is not dealing with the relevance of the problems but the construction of solutions to a requirement. Producing object-oriented solutions for a requirement is likely to be as error prone as solutions produced using traditional methodologies.

OMT methodology creators adopt an ontological perspective and accept the problem statement as given. They recommend that the methodology user clarify requirements but at no point do they undertake systemic analysis and conduct a critical enquiry of the situation of concern. The methodology creators are effectively concentrating on the organisational actions and decisions that have observable, substantive outcomes such as inventory levels or average time taken to satisfy an order, identifying task responsibilities and charting the flow of material etc and are not considering how the organisational activities are perceived, interpreted and legitimated.

OOSE takes a development approach that is used in the construction industry and applies the same principles to information systems development. The underlying enterprise philosophy implies that tools provide support for activities in three categories: architecture, method and process. The development of software systems is presented as analogous with the construction industry. Development is progressive, involving iterative cycles of analysis, construction and testing.

OOA does not help its users to participate in the definition of 'desired states' it cannot define problems, nor can it arrive at notional systems. The concern of the methodology is to record the features of the notional system(s) expressed by the client in a concise form in order to facilitate its design tasks.

OMT relies on the clients being clear about the rationale for their requirements. It does not offer any steps for examining the validity of these expectations. OMT does not assist the methodology user in the identification of 'current state' and 'desired state' what it is doing is taking the problem statement and producing an object model to represent the requirements of the problem statement. By modelling the requirements of the problem statement the object model is perceived to have 'solved' the problem without initially identifying what the problem is.

According to OOSE receiving the requirements specification, in some form, is the starting point for systems development. At an early stage of systems development, the main 'problem', according to OOSE, seems to be producing a requirements specification precise enough to become part of a contractual agreement between a client and a software producer. OOSE does not make its users aware of how to perform the diagnosis, nor does it help them to understand or question the rationale for the clients' 'desired states'. OOSE relies on the client being clear about the rationale for the requirements. It does not offer any steps for examining the validity of these expectations. As OOSE does not concern itself with the clients' 'desired states' (prognosis outline) it has no means of defining problems either. OOSE spends time with both clients and users discussing use cases in order to create a requirement specification that can be 'signed off'. The reasoning behind these requirements is never explicitly considered. OOSE is not dealing with the relevance of problems but the construction of solutions to requirements.

Methodology evaluation is of considerable importance to organisations and the NIMSAD framework is useful in that evaluation. It is important for methodology users to know how effective the methodology is and the lessons that can be learned after applying a methodology. OOA, however, limits the evaluation of the methodology to that conducted prior to intervention, as do OMT and OOSE.

Rumbaugh et al (1991) compare their own methodology with other software engineering approaches, including the object-oriented approaches of Booch (1994), Meyer (1988), and Coad and Yourdon (1991). The OMT methodology creators base the comparison for identifying strengths and weaknesses of each approach on 'a set of flexible criteria'. The purpose for such a comparison is to identify major differences

and similarities between OMT and other approaches and effectively only contributes to an evaluation of OMT before application.

The OOSE methodology does not advise its users to evaluate the effectiveness of the methodology in use, nor does it offer models/criteria to probe into whether the business problem is given adequate consideration. The OOSE methodology creators compare their methodology with other object-oriented system development methodologies Booch (1991), Coad and Yourdon (1991), Rumbaugh et al. (1991), and Wirfs-Brock and Wilkerson (1989). The aim of these comparisons is to influence potential users of the OOSE methodology. Their comparative assessment is less comprehensive than that suggested by NIMSAD framework and is mainly at the descriptive level. The OOSE methodology creators do not consider assessing the methodology before, during and after application.

9.3 Conclusions drawn from evaluated methodologies

Object-oriented methodologies address the object model and class structure in order to reflect states and transition of states in the 'real world.' This has been called object-oriented thinking or object-oriented view in object-oriented methodologies.

The ability to apply object-oriented thinking is identified, by OOSE methodology creators, as being important when using their methodology. Object-oriented thinking integrates static and dynamic aspects of entities. It is a change for methodology users from function-oriented system development. OOSE purports to teach its methodology users an object-oriented view by introducing fundamental concepts of object-orientation and applying them in different cases. The OOSE methodology creators alert the methodology user to the importance of object-oriented thinking but they do not consider whether the system users should also be made aware of the differences between function-oriented thinking and object-oriented thinking.

None of the three object-oriented methodologies considered systemic analysis and design advocated by Jayaratna (1994). They generally paid little attention to problem formulation, instead they move from requirements specification directly to systems design. This indicates two problems:

- the three methodologies are weak at identifying business problems;
- by taking a new information system 'as the given notional system' narrows the system boundary.

This can be seen as a reason why OOSE and OMT do not stress the 'why' questions and why OOA only addresses 'why' questions when checking 'real' information needs. The three methodologies guide their users on information systems analysis and design actions, rather than on broadly considering how to resolve a 'problematic situation' and transforming it into a desired situation.

The three methodologies cover the last three stages of the problem solving process as described by NIMSAD, i.e. performing logical design, performing physical design and implementing design. For these stages the three methodologies cover more detail than the NIMSAD framework. They discuss such features as documentation, testing, testing and to some extent non-functional requirements (OMT and OOA) that are not explicitly covered by NIMSAD framework.

The first five stages of problem formulation, as described by NIMSAD, comprise understanding the 'situation of concern', diagnosis, prognosis outline, defining 'problems' and deriving notional systems. The three methodologies take the requirements specification as given and do not consider wider implications of the 'proposed system' they are limited to accepting the 'proposed system' as the notional system. Therefore no problem solving is being conducted. Rather the methodology takes the requirements specification as given and develops an information system based on that information. Different techniques are used to enable user participation in deriving valid specifications for design, for example: OOSE recommends use cases and enterprise development if specifications remain unclear, and OMT and OOA recommend talking to users in order to clarify requirements. These techniques do not equate to an inquiry corresponding to the critical questions raised by NIMSAD when applying the first four stages of problem formulation.

Coad and Yourdon (1991) have created an object-oriented methodology where object components are a central concept. Rumbaugh et al (1991) present a methodology that to a high degree seeks a compromise with traditional analysis and design

methodologies and contains many practical instructions. Jacobson et al (1998) emphasises a thorough description of the computerised systems use based on a development approach used in the construction industry.

Object-oriented analysis methodologies imply that the analysts can step into any organisation and undertake an analysis exercise. Shlaer and Mellor (1988); Coad and Yourdon (1991); Rumbaugh et al. (1991) state that reading documents and talking to the client and users will supply the necessary background knowledge and enable the analyst to understand the problem domain. However having gained an understanding of the problem domain does not mean that the analyst shares the same understanding of the terminology used.

Information analysis is concerned initially with understanding the business itself and business needs it then considers how the needs can be met by providing an organisational and technical system. Business needs are rarely considered by object-oriented analysis methodologies. There is an assumption (Jacobson et al. 1998) that an in-depth business analysis has been performed prior to the object-oriented analysis. Coad and Yourdon (1991) and Rumbaugh et al. (1991) concentrate on the perceived problem area and do not consider the organisational requirements. Indeed Rumbaugh et al. (1991) state that 'psychological, organisational and political considerations for doing this (analysis) are deemed to be beyond the consideration of the methodology'.

9.4 Conclusions drawn from the use of NIMSAD

The discussion of 'mental constructs' helps problem solvers to recognise their competence in solving any problem. The dynamic feature of this construct will guide system developers to act in a dynamic situation. It is important that problem solvers recognise the 'mental constructs' that they have, and those that they will need when the situation changes, for example: when an enterprise adopts new ways of marketing or production. NIMSAD does not pay explicit attention to the changes within system developers 'mental constructs' needed when shifting from one methodology to another or from one method of systems development to another radically different one. For example: how the existing knowledge and experience of the systems

developer influences them when they move from function-oriented to object-oriented development.

The NIMSAD framework is weak in technical aspects for evaluating object-oriented methodologies and the three object-oriented methodologies are weak in formulating business problems. 'Why' questions; considered in every element of NIMSAD, in order to put the information system development into a broader context and to consider the effects of making particular decisions; are rarely considered in the methodologies studied. Instead, 'how' issues have been the main focus of all three methodologies. Traceability and reusability exist as the most important features in object-oriented methodologies, but are beyond the consideration of the NIMSAD framework.

NIMSAD framework can be utilised, in a broad sense, not only for methodology evaluation but also for project managers and intended problem solvers from an organisational perspective. This means that the framework does impact on the theoretical and practical development of systems development methodologies.

According to Jayaratna (1994) a framework has several structural properties and has the potential to become a methodology. This assumes that potential methodology users 'decide to select the stages of the problem solving process and create a structure in which to carry out the chosen steps' (Jayaratna 1994). Furthermore potential methodology users have to 'justify to themselves the rationale for undertaking the steps, in that order'. This should also apply to methodology creators when they design a methodology.

OOA methodology creators (Coad and Yourdon 1991) do not discuss methodology philosophy or frameworks. They advise their methodology users to take modelling activities in any order. Steps are unimportant in the problem solving process but their examples all exhibit some order in modelling activities. OOA methodology creators distinguish between experienced and inexperienced methodology users.

Generally speaking, the sequence of stages in OMT (Rumbaugh et al. 1991) is important. This methodology prescribes its work steps in an ordered sequence. The

OMT methodology creators, however, think that an ordered sequence of steps is not as important to experienced developers. The object model should be formulated before the dynamic model and the function model, since the 'experienced developers are able to combine several steps or perform certain steps in parallel for portions of a project'

In OOSE methodology (Jacobson et al. 1998) have taken an early version of their 'rationale enterprise philosophy' as a framework for their methodology. The first version of a coherent methodology description for OOSE existed in 1986. Since then, their framework has been further developed along with their methodology. A framework explicit or not can be seen as a necessary step toward methodology creation.

A methodology evaluation framework draws attention to what is included in a methodology, while a methodology focuses on what is to be accomplished and how. They are also different in use. Users of a methodology, not users of a framework, resolve business and information systems problems. People can however, use a framework to evaluate a systems development methodology.

NIMSAD framework as a problem-solving framework was useful during the project as it helped to focus the mind on the important issues being raised by the various methodologies. However, issues of semantics and semioitics could not be effectively resolved using the NIMSAD framework. With the benefit of hindsight, Multiview may have been more useful in this area.

9.4.1 Problem Formulation

The biggest difference between the evaluated methodologies and the framework lies in the problem formulation, which has been stressed in NIMSAD. The phenomena, as stated by Jayaratna (1994) are quite common:

> 'Most methodologies guide their users to accept the client's notional systems at the initial entry stage of the project and let this condition their enquiry and boundary construction. This can have serious and undesirable consequences. If the problems, particularly their contributory factors, remain outside the 'situation of concern' then no matter what improvements are introduced, the problems will continue to persist' (Jayaratna 1994).

What will happen if the client's notional system at the entry stage is accepted? The stages of 'defining the prognosis,', 'defining problems' and 'deriving notional systems' are carried out unconsciously by the client and offered during stage 1 'situation of concern'. OMT and OOSE do not alert their users to the risks involved in uncritically accepting the client's requirements as given. One reason could be that the methodology creators take the information systems development problem as the problem to be solved. This means they consider a problem with a very narrow meaning compared to that advocated by the NIMSAD framework. Object-oriented methodologies often offer solutions to information systems problems.

From my evaluation of object-oriented methodologies I found that 'problem formulation' differs considerably from the other phases (solution design and implementation design) of the problem solving process of NIMSAD. Problem formulation can be carried out in many ways. The most important issue is considering what should be included in a notional system and whether a notional system accurately reflects the stakeholders' needs and requirements. A requirements statement from clients or stakeholders should not signify the end of their participation in the information systems development, but the beginning. User roles are becoming more significant in information systems development and they (users) should be consulted for requirements specification, prototyping and interface design. In this respect, OMT takes the 'user statement' in natural language as a starting point for discussion. OOSE suggests that methodology users use 'requirements specification' from system users, but still co-operate with other users i.e. formulating use cases to construct information systems. OOA advocates starting from the requirements specification but also talking to the users and watching their work practices.

All three methodologies implicitly assume that the client or stakeholders and the problem solver have a shared understanding of the problem domain and the terminology being used.

9.4.2 Framework as a basis for learning

The NIMSAD framework helps problem solvers (or methodology users) with their thinking processes (Jayaratna 1994). The learning task for a methodology user is

ongoing. Methodology learning aims at understanding and using the methodology more effectively in order to build appropriate information systems. A framework such as NIMSAD provides a possible means of evaluating a methodology as it generates questions about methods.

There are three kinds of learning provided by the NIMSAD evaluation framework:

- the methodology users must become familiar with the problems to be solved from the 'problem situation' – context learning;
- the methodology user must become familiar with their own ability to solve information systems problems by examining their own 'mental constructs' – self-learning;
- the methodology users must understand the selected methodology being used to solve information systems problems – methodological learning.

In addition methodology users can increase their competence by solving many different information systems problems.

The OOSE methodology presents particular features for training and supporting systems developers. It can be seen as one way of learning through methodology application.

9.5 Summary

Table 9.1 summarises the issues that were identified when evaluating OOA, OMT and OOSE using the NIMSAD framework. The activities identified constitute a framework from which many instances of a particular methodology can be constructed. This allows the problem solver to be guided by the domain expert when constructing parts of the object model.

What could be learned from the NIMSAD framework in this context is that the methodology user should have reasons for inclusion of steps and the ordering of those steps.

In order to answer the research questions:

- How do we identify objects?

Table 9.1 Issues raised during evaluation.

Method evaluation issues raised by NIMSAD framework	Issues raised by OOA methodology	Issues raised by OMT methodology	Issues raised by OOSE methodology
The 'problem situation'			
Constructing a boundary of concern		Organisational considerations beyond the scope of OMT	Presumed as given by earlier enterprise development work
Clients;' problem formulations and solutions not taken as given	Presumes nothing as given Identifying a system purpose and its features by discussion with clients and users of the system	Requesters' requirement specification taken as a starting point	Orderers' requirement specifications taken as a starting point, but checked carefully with users
Avoidance of 'wrong' problem solution by explicitly relating systems development to organisational development	Re-engineer with objects and re-engineer on the boundary	Requesters' requirements specification agreed upon by all participants	Meeting never ending changes of requirements by a strategy of incrementally augmenting system functionality
Observing and resolving conflicts between stakeholders in the 'boundary of concern'			Observing and resolving conflicts between orderers and users: the requirement specification
The intended problem solver			
	Object-oriented as a new paradigm		
'Mental constructs'	Object-oriented view	Object-oriented thinking	Object-oriented view
Problem solver as a role	Systems developers as observers and 'modellers'	System users as problem owners	Producers as problem solvers
Assumption of client playing the role as 'intended problem solver'	System users as developer' guides, distinguishing actors and participants.	Experience of methodology users	System users describing use cases
			Training and supporting system developers
The problem solving process			
Problem formulation	Identifying system purpose and features	Analysis	Getting a requirements specification
Solution design	Selecting objects	System design	Analysis process
Design implementation	Establishing responsibilities	Object design	Construction process
	One model from concept to code	Implementing the design	Testing process
		Understanding of 'boundary of concern'	Reuse of information system components
		Defining object classes for the problem	
		Iterative modelling process	
		Reusability	
The evaluation			
Evaluation by raising critical questions on what a methodology offers		Comparing OMT with other methodologies	Project management, Methodology selection, testing as an integrated part of the methodology application

- Do object-oriented analysis methods help us to identify objects?

I considered three seminal texts to determine how they helped us to identify objects and concluded that all the methods considered rely on,

- Language features
- Experience
- Expertise of the user.

All the methods rely on a shared understanding of:

- Terminology of the problem domain
- Terminology of the methodology
- The context in which the problem is set.

As an academic it is important to encourage students to question what they read and hear. In academia we need to constrain scenarios in order that students concentrate on particular areas or aspects. In practice some scenarios may also have very precise clearly defined boundaries but many may be open to interpretation.

On reflection I set out to do action research instead I have conducted an exploratory investigation that has included semi-formal interviews and observation. Action research is the next phase of the research.

9.6 Future Work

Future research will concentrate on refining and developing methods to carry out object-oriented construction. This should take into account different world-views and different social contexts in which people make sense of their worlds.

Future work would include:

- Investigating claims that object-oriented approaches are radically different ways of thinking about problems.

This could be developed in several ways including:

- Developing a theoretical basis for conducting object-oriented modelling

- Action research with companies with a view to understanding the relevance of the concepts in practice in order to determine:
  - what experience and prior knowledge do practitioners rely on in order to understand the problem domain?
  - do practitioners use particular object-oriented methodologies?

References

Abbott, R. J. (1983). "Program Design by Informal English Descriptions." Communications of ACM **26**(11): 882 -894.

Avison, D. E. (1992). Information Systems Development. A Database Approach. Alfred Waller.

Avison, D. E. and G. Fitzgerald (1995). Information Systems Development: Methodologies, Techniques and Tools, McGraw-Hill.

Avison, D. E. and A. T. Wood-Harper (1990). Multiview: an Exploration in Information Systems Development. Oxford, Blackwell Scientific Publications.

Banville, C. and M. Landry (1989). "Can the field of MIS be disciplined?" Communications of ACM **32**(January): 48 - 61.

Barker, S. (1997). Critical Evaluation of Jackson Structured Design (JSD) Using The NIMSAD framework. Information Systems Methodologies, 5th International Conference on Information Systems Methodologies, Preston.

Baskerville, R., L and A. Wood-Harper, T (1996). "A Critical Perspective on Action Research as a Method for Information Systems Research." Journal of Information Technology **11**: 235 - 246.

Benbasat, I., D. Goldstein and M. Mead (1987). "The Case Research Strategy in Studies of Information Systems." MIS Quarterly **11**: 369 - 386.

Bertino, E. and L. Martino (1993). Object-Oriented Database Systems, Concepts and Architecture, Addison Wesley.

Bessagnet, M. N., J.-M. Larrasquet and N. Jayaratna (1994). Object Oriented Approaches to the Analysis and Design of Information Systems to Support Modern

Organisational Forms. Second Conference on Information Systems Methodologies., Edinburgh.

Boland, R. J. and R. Hirschheim, Eds. (1987). Critical Issues in Information Systems Research. Wiley Series in Information Systems.

Booch, G. (1991). Object-Oriented Design with Applications., Benjamin Cummings.

Booch, G. (1994). Object Oriented Analysis and Design with Applications, Benjamin Cummings.

Booch, G., I. Jacobson and J. Rumbaugh (1996). The Unified Modeling Language for Object Oriented Development, Rational Software Corporation.

Bouzeghoub, M. and E. Metais (1991). Semantic Modeling and Object Oriented Modeling: Two Complementary Paradigms. Proc 10th Int. Conf. on Entity-Relationship Approach., San Mateo, California.

Brown, A. (1991). Object-Oriented Databases. Applications in Software Engineering, McGraw-Hill.

Brown, A. W. (1989). "From Semantic Data Models to Object Orientation in Design Databases." Information and Software Technology **31**(1): 39 - 46.

Bryman, A. (1989). Doing Research in Organisations, Routledge.

Buchanan, D., D. Boddy and J. McCalman (1988). Getting In, Getting On, Getting Out, and Getting Back. Doing Research in Organisations. A. Bryman, Routledge.

Checkland, P. and J. Scholes (1990). Soft Systems Methodology in Action, Wiley.

Checkland, P. B. (1981). Systems Thinking, Systems Practice. Chichester, Wiley.

Checkland, P. B. and S. Howell (1998). Information, Systems and Information Systems: Making sense of the field, Wiley.

Chen, P. P.-S., Ed. (1976). The Entity-Relationship Model - Towards a Unified View of Data. Readings in Database systems (1994), Morgan Kaufmann.

Clark, A. (1972). Action Research and Organizational Change. London, Harper & Row.

Coad, P., D. North and M. Mayfield (1997). Object Models: Strategies, Patterns and Applications. New Jersey, Yourdon Press.

Coad, P. and E. Yourdon (1991). Object Oriented Analysis. Eaglewood Cliffs, NJ:, Prentice Hall.

Codd, E. F., Ed. (1970). A Relational Model of Data for Large Shared Data Banks. Readings in Database Systems (1994), Morgan Kaufmann.

Conger, S. (1994). The New Software Engineering. Belmont, California, International Thomson Publishing.

Connolly, T., C. Begg and A. Strachan (1995). Database Systems A Practical Approach to Design, Implementation and Management, Addison-Wesley.

Coolican, H. (1990). Research Methods and Statistics in Psychology , Hodder and Stoughton

Cutts, G. (1987). SSADM: Structured Systems Analysis and Design Methodology, Paradigm.

Date, C. J. (1995). An Introduction to Database Systems, Addison Wesley.

Davis, L. and A. T. Wood-Harper (1990). Multiple Perspectives in Desk-top Computing. Desktop Information Technology. K. M. Kaiser and H. J. Oppelland, North-Holland.

de Champeaux, D. and P. Faure (1992). "A Comparative Study of Object Oriented Analysis Methods." Journal of Object Oriented Programming **5**(1, Mar/Apr): 21 - 33.

DeMarco, T (1979). Structured Analysis, Yourdon Press.

Dickson, G. W., J. A. Senn and N. L. Chervany (1977). "Research in Management Information Systems: the Minnesota Experiments." Management Science **23**(May): 913 - 925.

Dik, S. C. (1989). The Theory of Functional Grammar, Part 1: the Structure of the Clause. Dordrecht, Netherlands, Foris Publications.

Douglas, J. D. (1976). Investigative Social Research. Beverly Hills, Sage.

Edwards, J. (1989). Lessons learned from more than ten years of practical application of the object-oriented paradigm. Proc CASExpo-Europe, London.

Elmasri, R. and S. B. Navanthe (1994). Fundamentals of Database Systems, Benjamin Cummings.

Embley, D., B. Kurtz and S. Woodfield (1992). Object Oriented Systems Analysis - a Model Driven Approach.

Eriksson, H.-E. and M. Penker (1998). UML Toolkit, Wiley.

Fichman, R. G. and C. F. Kemerer (1992). Object-Oriented and Conventional Analysis and Design Methodologies. Comparison and Critique. Computer: 22 - 39.

Firesmith, D. G. (1993). Object-Oriented Requirements Analysis and Logical Design, Wiley Professional Computing.

Fitzgerald, G., N. Stokes and J. R. G. Wood (1985). "Feature Analysis of Contemporary Information Systems Methodologies." Computer Journal **28**(3).

Gadamer, H.-G. (1975). Truth and Method, Seabury Press.

Gadamer, H.-G. (1976). The Historicity of Understanding. Critical Sociology, Selected Readings. P. Connerton, Penguin Books: 117 - 133.

Galliers, R. D. (1991). Choosing Appropriate Information Systems Research Approaches: A revised taxonomy. Proc. of IFIP TC8/WG/8. Information Systems Research: Contemporary Approaches and Emergent Traditions.

Galliers, R. D. (1995). "A Manifesto for Information Management Research." British Journal of Management **6**: 545 - 552.

Goodman, N. (1989). "Object Oriented Database systems." INFOD **Fall 89**.

Goldberg, A. and D. Robson (1983) Smalltalk-80: The language and its Implementation. Addison-Wesley.

Graham, I. (1995). Object Oriented Methods, Addison-Wesley.

Harmon, P. and M. Watson (1998). Understanding UML The Developer's Guide. San Francisco, Morgan Kaufmann.

Hughes, J. (1991). Object Oriented Databases, Prentice Hall.

Hult, M. and S.-A. Lennung (1980). "Towards a definition of action research: a note and bibliography." Journal of Management Studies **17**(May): 241 - 250.

Husserl, E. (1936). The origins of geometry. Phenomenology and Sociology. T. Luckman. Harmondsworth, Penguin.

Hutt, A. T. F., Ed. (1994). Object Analysis and Design: Comparison of Methods., Wiley.

Iivari, J. (1994). "Object Oriented Information Systems Analysis: a comparison of six object oriented analysis methods." Methods and Association tools for the information Life Cycle: 85 - 110.

Jacobson, I., M. Christerson, P. Jonsson and G. Overgard (1998). Object Oriented Software Engineering - A Use Case Driven Approach, Addison Wesley.

Jayaratna, N. (1994). Understanding and Evaluating Methodologies NIMSAD A Systemic Framework, McGraw Hill.

Jayaratna, N. and B. J. Oates (1995). Evaluation of Yourdon Systems Method using the NIMSAD conceptual framework. 2nd Scandinavian Research Seminar on Information and Decision Networks.

Jenkins, M. (1985). Research methodologies and MIS research. Research Methods in Information Systems. E. Mumford, R. Hirschheim, G. Fitzgerald and A. T. Wood-Harper. Amsterdam,
North-Holland: 103 - 117.

Khoshafian, S. and R. Abnous (1990). Object Orientation: Concepts, Languages, Databases, User Interfaces, Wiley.

Kim, W. (1990). Introduction to Object Oriented Databases, MIT Press.

Klein, H. K., R. Hirschheim and H.-E. Nissen (1991). A Pluralist Perspective of the Information Systems Research Arena. Information Systems Research: Contemporary Approaches and Emergent Tradition, Copenhagen, Elsevier Science.

Klein, H. K. and K. Lyytinen (1985). The Poverty of Scientism in Information Systems. Research Methods in Information Systems. Proc. of the IFIP WG 8.2 Colloquium, Manchester.

Kroenke, D. (1995). Database Processing. Fundamentals, Design and Implementation, Prentice Hall.

Laing, R. D. (1967). The Politics of Experience and the Birds of Paradise, Penguin.

Land, F. F. (1982). Adapting to Changing User Requirements. Information Analysis: Selected Readings. R. D. Galliers, Addison-Wesley: 203 - 229.

Lewis, P. (1994). Information-Systems Development, Pitman.

Losavio, F., A. Matteo and F. Schlienger (1994). "Object Oriented Methodologies of Coad and Yourdon and Booch: Comparison of Graphical Notations." Information and Software Technology **36**(8): 503 - 514.

Lui, K. (1999). Semiotics in Information Systems Engineering.

Lui, K., A. Alderson, H. Shah, B. Sharp and A. Dix (1998). Applying Semiotic Methods to Requirements Recovery. Methodologies for Developing and Managing Emerging Technology Based Information Systems.
Sixth International Conference on Informaion Systems Methodologies., Salford, Springer.

Lyytinen, K. (1987). A Taxonomic Perspective of Information Systems Development: Theoretical Constructs and Recommendations. Critical Issues in Information Systems Research. R. J. Boland and R. A. Hirschheim, Wiley.

Maciaszek, L. A. (1990). Database Design and Implementation, Prentice Hall.

Maddison, R. N., Ed. (1983). Information System Development: A Flexible Framework. London, British Computer Society.

Maddison, R. N. (1984). A Feature Analysis of Five Information System Methodologies. Beyond Productivity: Information Systems for Organisational Effectiveness. T. H. Benelmans, North-Holland.

Martin, J. (1975). Computer Data-Base Organisation, Prentice Hall.

Martin, J. and J. Odell (1992). Object-Oriented Analysis and Design, Prentice Hall.

McFarlan, F. W. (1984). The Information Systems Research Challenge. Proc. of the Harvard Business School Research Colloquium, Boston, Harvard Business Press.

McGinnes, S. (1993). A Framework for the Comparative Evaluation of Information Systems Methodologies. Conference on the Theory, Use and Integrative Aspects of IS Methodologies, Edinburgh.

Mellor, S. and N. Lang (1997). Using UML. www.projtech.com.

Meyer, B. (1988). Object-Oriented Software Construction. New York, Prentice Hall.

Minsky, M. (1975). A framework for representing knowledge. Mind Design. J. Haugeland, MIT Press.

Morgan, G., Ed. (1983). Beyond Method: Strategies for Social Research, Sage.

Mumford, E. (1983a). Designing Participatively. Manchester, Manchester Business School.

Mumford, E. (1983b). Designing Human Systems. Manchester, Manchester Business School.

Mumford, E., R. Hirschheim, G. Fitzgerald and A. T. Wood-Harper (1984). Information Systems Research - a dubious science? Proc. of IFIP WG 8.2 Colloquium, Manchester UK.

Nielson, P. A. (1990). "Approaches to Appreciate Information Systems Methodologies." Scandinavian Journal of Information Systems **2**.

Nissen, H.-E., H. K. Klein and R. Hirschheim (1991). Information Systems Research: Contemporary Approaches and Emergent Traditions. Proc of IFIP TC8/WG/8.

Norman (1996). Object Oriented Systems Analysis and Design.

Nygaard, K and O. J. Dahl (1981) "The Development of the Simula Language" IN Wexelblat, R.L (eds) (1981) History of Programming Languages. Academic Press

Oates, B. (1995). Evaluation of the NIMSAD conceptual framework. (Normative Information Model-based Systems Analysis and Design). Information Systems Methodologies 1995 3rd Conference on information Systems Methodologies, North East Wales Institute, Wrexham.

Oates, B. J. and N. Jayaratna (1995). Evaluation of Yourdon Systems Method using the NIMSAD conceptual framework. Proceedings of Second Scandinavian Research Seminar on Information and Decision Networks.

Olle, T. W., J. Hagelstein, I. G. MacDonald, C. Rolland, H. G. Sol, F. J. M. Van Assche and A. A. Verrjin-Stuart (1991). Information Systems Methodologies - A Framework for Understanding. Wokingham, Addison-Wesley.

O'Neil, P. (1994). Database: Principles, Programming, Performance, Morgan Kaufmann.

Page-Jones, M. (1992). "Comparing Techniques by Means of Encapsulation and Connascence." Communications of ACM **35**(4): 63- 69.

Palmer, R. (1969). Hermeneutics: Interpretation Theory in Schleiermacher, Dilthey, Heidegger, and Gadamer, Northwestern University Press.

Paton, N., R. Cooper, H. Williams and P. Trinder (1996). Database Programming Languages, Prentice Hall.

Pettigrew, A. M. (1989). Issues of Time and Site Selection in Longitudinal Research on Change. The Information Systems Research Challenge: Qualitative Research Methods, Harvard Business School.

Quillian, M. R. (1967). "Word Concepts: a Theory and Simulation of Some Basic Semantic Capabilities." Behavioural Sciences **12**: 410 - 430.

Quinton, A. (1973). The Nature of Things, Routledge and Kegan Paul.

Ramachandran, M. and M. Taylor (1994). Requirements for Object Oriented Analysis Methods. Second Conference on Information Systems Methodologies.

Rational (1997). UML Notation Guide Version 1.1.

Reenskaug, T., P. Wold and A. Lehne (1996). Working with Objects: The OOram Software Engineering Method, Manning Publications.

Roethlislerger, F. J. and W. J. Dickson (1939). Management and the Worker. Mass., Harvard University Press.

Rumbaugh, J. (1996). "To Form a More Perfect Union: Unifying the OMT and Booch methods." JOOP **8**(8): 14 -18.

Rumbaugh, J., M. Blaha and W. Premerlani (1991). Object Oriented Modeling and Design. Englewood Cliffs, NJ:, Prentice Hall.

Shlaer, S. and S. Mellor (1988). Object Oriented Systems Analysis, Modelling the World in Data, Prentice Hall.

Shlaer, S. and S. J. Mellor (1991). Object Lifecycles: Modelling the World in States, Yourdon Press.

Simon, H. A. (1978). "Rationality as Process and as Product of Thought." American Economic Review **68**(2): 1-16.

Smith, J. A. (1995). Semi-Structured Interviewing and Qualitative Analysis. Rethinking Methods in PSychology. J.A. Smith, R. Harre, L.V. Langenhove. Sage.

Stamper, R. (1988). "Analysing the Cultural Impact of a System." International Journal of Information Management **8**(3).

Susman, G. (1983). Action Research: a sociotechnical system perspective. Beyond Method: Strategies for Social Research. G. Morgan, Sage**:** 95 - 113.

Susman, G. and R. Evered (1978). "An assessment of the scientific merits of action research." Administrative Science Quarterly **23**(December): 582 - 603.

Vickers, G. (1990). "Education in Systems Thinking." Journal of Applied Systems Analysis **7**: 3 - 10.

Vidgen, R. (1998). Using Multiview2 Framework for Internet-Based Information Systems Development. Sixth International Conference on Information Systems Methodologies, Salford, Springer.

Vidgen, R. and J. R. G. Wood (1995). Information Systems Development: methods, modelling and metaphors in an Object-Oriented world. UK Systems Society Fourth International Conference, Critical Issues in systems Theory and Practice.

Warmington, A. (1980). "Action Research: its method and its implications." Journal of Applied Systems Analysis **7**(April): 23 - 29.

Webb, G. (1990). Putting Praxis into Practice. Proceedings of 1st World congress on Action Research, Brisbane, Acorn.

Weick, K. E. (1984). Theoretical Assumptions and Research Methodology Selection. The Information Systems Research Challenge. F. W. McFarlan. Boston, Harvard Business School Press: 111 - 132.

Whyte, W. F. (1984). Learning from the Field: A Guide from Experience, Sage.

Wilson, B. (1984). System Concepts: Methodologies and Applications, Wiley.

Winograd, T. and F. Flores (1986). Understanding Computers and Cognition. Reading MA, Addison-Wesley.

Wirfs-Brock, R. and B. Wilkerson (1989). "Object-Oriented Design: a Responsibility-Driven approach." Sigplan Notices **24**(10).

Wirfs-Brock, R., B. Wilkerson and L. Wiener (1990). Designing Object-Oriented Software, Prentice-Hall.

Wirfs-Brock, R. J. and R. E. Johnson (1990). "Surveying Current Research in Object-Oriented Design." Communications of ACM **33**(9): 104 - 124.

Wood-Harper, A. T., Ed. (1985). Research Methodologies in Information Systems: using Action Research. Research Methodologies for Information Systems, North-Holland.

Wood-Harper, A. T. and G. Fitzgerald (1982). "A Taxonomy of Current Approaches to Systems Analysis." Computer Journal **25**(1).

Yin, R. (1989). Case study Research: Design and Methods, Sage.

Yourdon, E. (1989). Modern Structured Analysis, Prentice Hall.

Zhang, X. (1999). User Participation in Object-Oriented Contexts - From Methodological and Practical Perspectives. Department of Informatics. Lund, Lund University: 180.

Zuber-Skerritt, O., Ed. (1996). New Directions in Action Research, Falmer Press.