How Objective is Object-Oriented Analysis?

Simon McGinnes

School of Information Systems, Queensland University of Technology
PO Box 2434, Brisbane QLD 4001, Australia
email: mcginnes @snow.fit.qut.edu.au

Abstract. There are many techniques for information systems analysis. Some techniques are complementary to one another, while others address similar issues in different ways; typically, methods concentrate on a subset of the total systems development process, or embody a particular way of viewing information systems. Recently, a number of object-oriented analysis techniques have been published; this paper briefly considers where these techniques fit in the overall spectrum of methods, and asks to what extent they can be judged to be 'complete' analysis methods in the context of business information systems development. A simple set of requirements for useful information systems analysis methodologies is outlined and is used as a framework for discussion of object-oriented analysis methods generally. Finally, some suggestions are made for further work to help evaluate and improve the effectiveness of object-oriented analysis methods for business information systems analysis.

1 Introduction

A recent survey of information systems analysis approaches [26] highlighted the wide diversity of both information systems methodologies and comparative frameworks for the evaluation of methodologies. There are many conflicting views on what constitutes an acceptable approach to systems analysis. In this paper, some very high-level but important requirements are given in an attempt to establish what makes analysis methods useful. These requirements are based on the needs of the main participants in the systems development process.

The structure of the paper is as follows: first, we briefly review current thinking on information systems analysis; in section 2 a number of requirements for analysis methods are outlined; in section 3 object-oriented analysis methods are reviewed in the light of the requirements; in section 4 some observations of a general nature are made about the methods reviewed; and in section 5 a summary of the main points is given.

1.1 Limited Views of Systems Analysis

To a large extent, thinking by information systems professionals about analysis methods has been rather compartmentalised. Practicing analysts tend to stick to the small number of techniques they know best [22]. This is one reason why

CASE tools have not been taken up as rapidly as predicted, since moving to a CASE tool often means adopting new techniques. In view of the high stakes involved in business software development, and of the risks inherent in adopting new and possibly unproven techniques, this conservatism is not surprising.

A compartmentalised view is also evident in the research efforts devoted to information systems analysis. Typically, research can be classified under one of a number of headings: *formal methods* (e.g. [4]), *strategic information planning* (e.g. [3]), *human factors* (e.g. [6]) and so on. Someone with a business analysis bias is likely to view the process of systems analysis rather differently from someone with a software engineering background, and both will disagree with the view put forward by a formal methods specialist. There is nothing inherently wrong in having such a multiplicity of views; the huge range of computer applications today means that no one technique can possibly be the best under all circumstances. What is undesirable is for practitioners and researchers to be unaware of alternative viewpoints put forward in disciplines related to their own. For an interesting discussion of these issues, see [36].

1.2 Methodology as Reaction

Of the wide range of information systems methodologies available, many can be seen as reactions against perceived situations. For instance, structured analysis and design [44] can be seen as a reaction against a perceived lack of formality in systems development. Formal methods go even further in the same direction. Socio-technical approaches [32] can be seen as a reaction against the failure of systems designers to take into account the impact of information systems on people. Systems approaches [41] can be seen as a reaction against systems developers' inability to question their own assumptions about solving problems. Prototyping (and variations) [27] can be seen as a reaction against the institutionalised delays created by the 'traditional' life cycle approach and against the inability of users to deal with requirements when expressed in abstract ways. JAD [43] can be seen as a reaction to the common 'paralysis by analysis' syndrome and to the political manoeuvering which often bogs down systems development projects.

1.3 Effect of Narrow View

In fact, it seems that the authors of almost every methodology all have at least one axe to grind. Methodologies which concentrate in this way on a limited view of the systems development problem run the risk of failing to address important issues. Of course, this does not mean that systems developments using these methods will necessarily fail, although a large number of systems developments do [12]. One reason is that an experienced analyst will tend to take into account everything which seems relevant when formulating requirements, regardless of what methodology is in use [39].

Perhaps, then, it is useful to view a methodology, not as a prescription for effective analysis, design and so on, but as a way of representing and justifying a mental view of the requirements which the analyst has already assembled. That is, the methodology is used for post hoc *rationalisation* (for example, see [31]).

From my own experience of using and teaching analysis techniques such as data modelling, this view seems valid: novice data modellers often produce poor results, not because they fail to grasp the technique's rules, but because they do not have a view of the end result in their minds' eye before they start!

1.4 Systems Analysis as an Intuitive Process

This paper is not the place for a thorough discussion of the mental processes involved in systems analysis. Suffice to say that thinking by successful analysts may be more intuitive (i.e. right-brain) and less consciously analytical (i.e. left-brain) than previously thought. Therefore, an important question we should ask about each methodology is: how useful is it for documenting an intuitively-created model? If an analyst has built up a rather holistic mental model of a particular problem situation, it would be desirable to have a reasonably neutral, or objective, medium in which to express this view. Natural language is one such medium, which has been used extensively for the expression of specifications in the past. Unfortunately, natural language is perfectly capable of representing poorly structured or unstructured specifications as well as structured ones. Instead, we need techniques which allow structure to be built into a specification, but which do not impose any particular structure. That is, we need *objective* methods.

Objectivity is not the only desirable attribute of analysis methods. In the following section, several additional requirements are identified.

2 What Should an Analysis Technique Be?

Many frameworks for the evaluation and comparison of information systems methodologies have been published. Each emphasises particular aspects of the systems development process. Examples include [9], [15], [16], [17], [19], [24], [25], [30], [34] and [38], ranging from simple comparisons of methodologies to comprehensive checklists which are so all-encompassing as to approach methodology specifications themselves. In this paper, I present only a very simple and high-level view of the requirements for useful analysis methods, in an attempt to keep the discussion clear. My approach is based on the needs of the key participants: the clients and the developers.

2.1 Clients' Needs

Multiple Views of Problem Situation. Typically, requirements for information systems are gathered through consultation with a number of clients (or users). Each client is likely to hold a slightly different view of the problem situation and of its potential solutions. It is up to the analyst to use suitable techniques to elicit the different clients' views of the situation. Since there is often insufficient time or money to model each individual's own view of the world separately, a compromise view may be developed (for instance, a corporate data model). Despite this, the analysis technique must be able to capture at least the most important

variations between the different user groups, otherwise the compromise may satisfy nobody at all.

For example, it is common for different sets of users in the same organisation to view specific parts of the corporate data model in different ways. One set might have a simpler view of the data than another. The database structure must accommodate the more complex of the two views in order to avoid losing information. But the users with the simpler conception of the data can still be helped, if they are given ways of accessing the data which hide complexity (such as SQL views). This can only be done if the two views of the data structure have been described independently during analysis.

Easily Understood Method. Because this compromise model must be seen and used by a variety of people, with differing levels of knowledge and differing requirements, it must be expressed in a way which is easily understood without the need for extensive explanation. Ideally, the technique will allow the clients to participate fully in the production of the models. This means making the technique as simple as possible: perhaps simpler than many systems developers would like.

Often, graphical techniques are offered in an attempt to satisfy these needs, and there is evidence that they can help a great deal [1]. However, the mere fact that a technique involves the drawing of diagrams is no guarantee that it is any more understandable or practical.

Relating Views at Different Levels. For similar reasons, it must be possible to represent chosen key aspects of the model in a way which avoids confusion, which means that the technique must support *abstraction* or *information hiding*. It must be possible to gather high-level requirements as well as low-level ones, and where low-level requirements refine higher-level ones, the connection should be clear.

For example, the higher levels of management often have a wide-ranging but simplified view of the operations in their organisation. Their view can be very useful in helping to define a global architecture for the organisation's information systems. The high-level model can then be refined by through consultation with people who have a more detailed view of specific areas. Relating these high- and low-level views is an important advantage of abstraction.

Since there are many ways of achieving these aims, we say only that a methodology must provide useful abstraction mechanisms, and that these mechanisms must apply to every type of model, whether it be data, process or behaviour (or anything else). This does not preclude the use of object classes in analysis, but says something about how they can be used to provide abstract views of the model.

Richness. The technique must be powerful and rich enough that it can represent a reasonable amount of information in a succinct manner. The requirements for an *easily understood method* and for *richness,* are in some ways contradictory, although they are not necessarily mutually exclusive. The temptation is to load the techniques with as many information-holding features as possible. But adding more information into a diagram can overload a user. This is another reason for having an abstraction mechanism: diagrams can be made simple by hiding certain information. In other words, only certain facts need be captured initially, details

being filled in later. This is one reason why techniques which support a single level of abstraction are at a considerable disadvantage to those which allow many levels.

The kinds of information a client is interested in having recorded could differ from the information the systems developer is interested in recording. The analyst may focus directly on the information required to construct a database and programs, without explicitly considering other important issues such as politics, ergonomics, job design and so on (see 'technical content', below). There is much debate about what issues should be addressed by information systems methodologies (refer to the frameworks mentioned earlier). While that debate is important, this paper avoids discussion of particular content issues by attempting to identify more general principles of methodology use.

Recognisable Terminology. Finally, it is imperative that any model produced during analysis is firmly rooted in the clients' own terminology. The mark of an ineffective analyst is the sight of clients translating words used in a model into their own terminology. This failure of communication may sometimes be so subtle as to go unrecognised, with both parties believing they are speaking the others' language. The model must express concepts familiar to the clients, in their own terms.

2.2 Developers' Needs

The information systems professionals concerned with producing and maintaining automated information systems have needs which are distinct from those of their clients and which are, in some ways, at odds with them.

Technical Content. In view of their primary product, computerised information systems, developers require detailed and accurate specification of data and processes (whether expressed separately, as in conventional database-program architectures, or combined, as in the object-oriented approach). In general, the more formal the representation of this information, the better, since any ambiguity needs to be resolved before the information can be used to design workable systems. In an ideal world, developers would be presented with a complete statement of requirements before they embark on each development. This is rarely the case in practice, and developments where the requirements do not undergo substantial modification are even rarer.

Objectivity. If we accept the view of an analysis technique as rationalisation (discussed in the introduction), then we must ask how well each method is suited to documenting, without inherent bias, an intuitively arrived-at model. Another way of asking the same question is to see how free the technique is of implementation-specific constructs.

For example, the use of algorithms to specify processes is heavily implementation-specific, since it determines not only the effect of the processes, but also their implementations. Using pre- and post-conditions is one way of avoiding this bias.

Minimal Solution. From a practical point of view, the fewer techniques the client, analyst and designer have to learn and use the better. This is especially the case where analysis proceeds from an initial identification of requirements through to a detailed requirements statement. If it is possible to use a single technique for expression of the requirements at all stages, several benefits accrue: (a) work is saved because using more than one technique would almost inevitably lead to a degree of overlap; (b) it is easier to see the relationship between initially expressed requirements and detailed requirements.

Being able to relate high-level requirements to detailed ones can be crucial. For instance, a software house developing software on behalf of a client has to be very aware of any requirements which were not originally contracted for, and the client will be careful to ensure that all requirements originally contracted for have been addressed. Therefore, methodologies should offer, at least, excellent integration between the tools used at different stages.

Self-Checking Method. The advent of widespread use of CASE tools has led to an improvement in the internal consistency of analysis models. The laborious cross-checking required in paper-based methods was often simply not done. CASE tools can now automate much of this task. Methods should take advantage of this capability to produce better quality models.

For example, a common form of checking now performed routinely by CASE tools which support E-R and DFDs is to determine whether each data entity is used in at least one process, and whether each data entity has a complete life cycle (birth, life, death). Manually carrying out these checks in large systems can be very tedious.

2.3 Summary of Requirements

Table 1 summarises the requirements identified in section 2.

Clients' Needs	*Developers' Needs*
Multiple views of problem situation	Technical content
Easily understood method	Objectivity
Relating views at different levels	Minimal solution
Richness	Self-checking method
Recognisable terminology	

Table 1 Requirements for analysis methods

3 Object-Oriented Analysis Techniques

Several books have been published recently, detailing comprehensive approaches to object-oriented systems analysis (and, in some cases, to design and implementation). It is not the purpose of this paper to describe those methods in any detail. Instead, the methods reviewed are briefly mentioned below and

then observations are made, based on the set of requirements identified in the previous section.

3.1 Methods

Coad and Yourdon [7] present a method which is firmly based in data modelling and which includes a technique similar to program flowcharts as a process modelling tool. Odell and Martin's method [29] uses event and activity schemata, to model behavioural and procedural aspects, together with a data modelling technique. Shlaer and Mellor [35] offer a method primarily based in entity-relationship modelling, but which also suggests the use of state transition and data flow diagrams. Weiss and Page-Jones' [40] method is an attempt to combine object-orientation with the best features of structured analysis. Rumbaugh et al's [33] method is a comprehensive approach which covers analysis, design and implementation in a well-integrated way. The approach by Wirfs-Brock et al [42] is remarkable in that it uses novel concepts, in particular focusing on the responsibilities of objects rather than simply examining their behaviour.

Other approaches are too numerous to describe, but include those of Jacobson [21], Bailin [2], Henderson-Sellers and Edwards [20], Kurtz et al [23], Colbert [8] and Gibson [18].

3.2 Discussion of Methods

When the object-oriented techniques discussed above are viewed in the light of the simple set of requirements for analysis techniques presented in the previous section, a number of general observations can be made. The points below are given in the same order as those in section 2. As before, we start with the clients' needs.

Multiple Views of Problem Situation. Few, if any, of the methods provide explicit support for modelling several views of the same situation. (It would, of course, be possible to build support for multiple views into most of the methods). Like structured analysis, object-oriented analysis tends to assume that things in the world have objective existence, and doesn't concern itself too much with the fact that different users may prefer to see things in different ways.

Easily Understood Method. Are the techniques useful when it comes to communication between analyst and client? The argument on this is likely to continue for some time, and the only real proof will be experience. At this stage, it appears that most of the diagrammatic techniques proposed in the major methods work reasonably well, provided that some licence is allowed in their use, especially in the representation of simplified or abstracted views of the models. Those techniques which do not use diagrams, such as several of the formal methods, are probably at a disadvantage in this respect.

For instance, Figure 1 shows three different means of specifying models. It is clear that some would require more explanation than others.

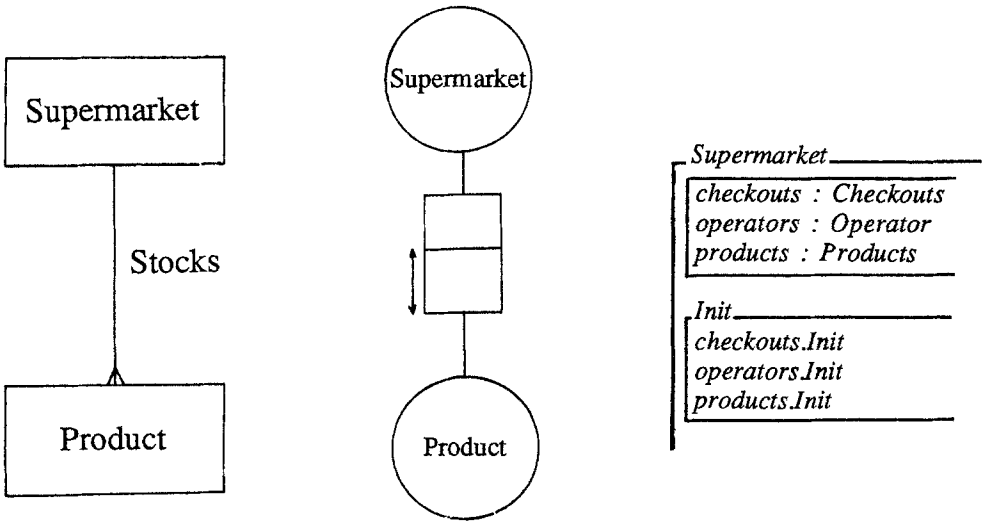

Fig. 1 Models (a) E-R (b) NIAM (c) Object-Z

Relating Views at Different Levels. Few of the methods offer much in the way of a 'big picture' view of a system. Those object-oriented analysis methods which offer a top-down approach to processes generally do it through data flow diagrams or similar constructs. Whether this is the best way is the subject of some debate [13]. Top-down functional decomposition, used with care, can be a useful tool for expressing, at a high level and in terms familiar to the clients, the business functions which a system will implement. This does not mean that a system need be viewed entirely as if it consisted of a single top function, split into lower-level functions and so on: it means simply that the concept of top-down decomposition can be a useful abstraction mechanism and ought to be available so that it can be used when appropriate. In fact, imposing a *rigid* top-down view of functions is often difficult and unnatural, so it is wise to make the use of this capability optional.

It seems that alternative ways of expressing operations at high and low levels are needed. Wirfs-Brock et al take a promising approach, using *contracts* and *subsystems* to group operations and objects. Presumably, these concepts could be extended to allow further levels of abstraction as required in larger systems. Aggregation also shows promise as a possible abstraction mechanism for object modelling.

Richness. Most of the techniques offer diagrammatic notations which are of a similar level of richness to (and in many cases are very similar to) commonly-used existing methods, such as E-R and data flow diagrams. The information captured when using object-oriented analysis is practically the same as when

using methods such as structured analysis: the repository structure for CASE tools to support each approach would be very similar. The one area where all of the object-oriented methods are in agreement, and differ from more traditional approaches, is in the association of each operation with a data object. This connection is typically not maintained by existing CASE tools. As discussed later, it is doubtful whether it always reflects a relationship which holds true in the application domain.

None of the methods offer much coverage of issues beyond the traditional data, process and behavioural perspectives.

Recognisable Terminology. Do the methods allow the use of the clients' own terminology in constructing models? There is some variation between the different approaches in this respect. Most of the major approaches do not impose any particular way of wording requirements, and therefore the analyst is free to choose the most appropriate terminology. Some of the more formal methods do impose certain restrictions: for instance, in Object-Z [14], abbreviations and short names are *de rigueur* and the use of a mathematical notation is likely to reduce the average client's ability to understand specifications.

Having discussed the clients' needs, we now consider those of the developers.

Technical Content. As observed above ('Richness'), object-oriented analysis methods do not differ markedly from structured analysis in their technical content.

Objectivity. The object-oriented approach is often described as 'natural' and 'intuitive', because it attempts to represent the application domain more closely than conventional ways of structuring information systems. This sounds ideal, but glosses over the fact that the object-oriented 'model' is simply another way of abstracting from reality. In fact, it is a rather implementation-specific way of modelling, since it assumes particular constructs (object classes, operations, messages and so on) which are stronger than those in other common modelling approaches.

For example, no-one seriously believes that the entity types depicted on ER diagrams necessarily exist in reality. Entities, like words, are simply a convenient way of capturing meaning. In contrast, objects in an object-oriented model are often equated with objects in the world.

Objects in an information system are distinct from objects in the real world, and can never be more than a representation or model of the real thing. If we make no distinction between analysis and design, as is proposed in a number of the methods, and instead view the process as one of progressive refinement, then we are stuck with the idea that the objects in the system are equivalent to those in the world. In fact this is not the case: it is often necessary to 'bend' reality to fit the object model.

For instance, the initial identification of object classes is often tricky. Is a *purchase* a class, an operation or an event? In the real world it is all of these things. Similarly, the choice of 'owner' for each operation may be difficult - some operations are not naturally the responsibility of any single object.

This lack of a conceptual model - the failure to distinguish between system objects and real world objects - is very much in evidence in the object-oriented literature.

Minimal Solution. Most of the object-oriented analysis techniques presume that analysis is preceded by a requirements definition activity, the output of which is usually assumed to be a narrative requirements specification. This activity is similar to the well-known idea of *business analysis*. Some authors, such as de Champeaux [13], refer to this process as 'domain analysis'.

However, the trend in recent years has been away from narrative specifications and towards more structured ways of representing requirements. It is generally agreed that this is desirable (for an example, see [22]). It is not immediately clear why a reversal of this trend is being proposed, unless, perhaps, the new techniques are seen as unsuited to high-level requirements analysis.

Self-Checking Methods. One of the chief distinguishing features of the object-oriented approach is that the process and data models are closely combined, in contrast with the separate models used in other approaches (such as structured analysis). However, the availability of appropriate CASE technology makes this distinction far less meaningful: CASE tools can now routinely be used to integrate the three perspectives at a very detailed level, so that there is essentially little difference between the 'integrated' (object-oriented) and the 'unintegrated' (non-object-oriented) approaches in this respect.

4 Discussion

In evaluating the object-oriented analysis methods against the set of requirements outlined in section 2, several issues of a more general nature raised themselves. These are discussed below.

Prototyping Approach. The systems development life cycle proposed for most of the methods is less rigid than the traditional 'waterfall' model. The consensus on the most appropriate life cycle for object-oriented development appears to be that either a weak life cycle, or no life cycle at all, should be used. Instead, a form of prototyping is recommended [27].

What distinguishes this approach from more conventional prototyping methods is that, in the object-oriented approach, both data and processes are developed concurrently, whereas in conventionally organised systems it is typically only the processes which are prototyped, the data structures being arrived at through some initial modelling process. It seems likely that the sort of risks inherent in allowing a database structure to be prototyped would also be present in the object-oriented equivalent: consider the great importance attached to data modelling and to the activities of data administrators and database administrators in most large organisations. This is not to say that the 'gestalt round-trip design' approach [5] is infeasible, but rather that we already know just how hard, and how risky, it is on large projects.

What Does Reuse Mean in Analysis? In the context of systems analysis, reuse means picking suitable existing classes in order to construct models. Reuse of existing models is widely cited as a possible means of reducing the effort required in analysis and design. Generally, abstraction is seen as the primary mechanism to enable reuse: if we make our classes abstract enough, they are sufficiently general to be reusable in other situations. However, as Sutcliffe [37] demonstrates, inappropriate use of abstraction can limit the potential for reuse, not increase it.

An organisation of any reasonable size may have hundreds of classes, if existing corporate data models are anything to go by. In this situation, it will be hard to know if an existing class is what is required, without looking in detail at its specification (for instance, what the effect of its operations are) and then comparing them to a well-researched requirement. In other words, you have to do some analysis *anyway*: matching the requirements to existing classes is then an additional task.

For instance, the meaning of the class *customer* in the context of a stock control application may be very different from the meaning of the class *customer* in the same organisation's credit checking system. Since it is impractical to analyse an entire organisation in one go, classes are only ever valid for the systems analysed so far.

The assumption made in the literature seems to be largely that the problem of selecting existing classes for reuse is simply one of finding an appropriate technological solution (such as class browsers, and so on). This misses the point that a detailed knowledge of the required class is necessary before a suitable existing class can be selected. The idea that there is only one way of seeing the application domain, and that it follows that any class which has already been 'discovered' will be as is required for new applications, is far too simplistic.

For these reasons, it is likely that the benefits of reuse in analysis will only be apparent in the 'core' classes of an organisation (i.e. the ones that are likely to occur repetitively). For the rest, it may well be quicker to define the classes from scratch.

Analysis or Design? If there is a general criticism in the area of terminology, it is that most of the methods require a certain 'recasting' of reality before they can represent it effectively. This really means that an element of design is allowed to creep in to what ought to be purely analysis. For example, in most of the methods, the principle of encapsulation ensures that every class contains the operations which apply to it. However, in some of the methods, the reverse is also true: every operation must be attributed to a given object class. While the first rule is useful from the point of view of analysis, the second can be very awkward: it is not difficult to think of business functions in which several 'objects' participate but which no one object owns. Some methods suggest constructing a pseudo-class to own this operation, simply so that the model does not break the rules. Others suggest breaking the operation down into smaller operations which can then be distributed amongst the existing classes. Both alternatives require some tailoring of reality to fit the model.

A second example concerns the modelling of continuous processes. In most methods, the representation of constantly-evolving processes needs to be

reduced to a succession of events, each one associated with a state change for one or more objects (sometimes referred to as ticks of an imaginary clock). While this captures the effect of the process, it is a nevertheless a distinctly artificial construction required only because the method contains no better alternative.

An analysis method which is not implementation-neutral forces the analyst to alter the world to fit the model, rather than vice-versa. The need for this mental conversion process reduces the value of the analysis model as a representation of reality. It means that, to understand the model, one has to be aware of the preconceptions embodied in it. Compare this with techniques such as NIAM [28] and (to a lesser extent) E-R, which strive to represent the world in an objective and unbiased way.

Convergence of Methods. Some of the methods mentioned above suggest the use of techniques for modelling the three information systems perspectives (data, process and behaviour). For instance, several authors propose the use of an ER-like technique for data modelling, a state transition diagram-based approach for behaviour modelling and data flow diagrams for process modelling (e.g. [33], [35]). The techniques are not necessarily used in a well-integrated way. Whether or not these techniques are ideal is the subject of some debate, which at this stage is unresolved.

However, there is nothing particularly new about this particular juxtaposition of techniques. For instance, the very popular methodology SSADM [11] has for some time recommended the use of exactly this combination. Even structured analysis [44] now includes similar data- and behaviour-modelling capabilities (although the behaviour-modelling technique is recommended only for real-time systems). This point is important when we consider the large investments in existing methods. For an organisation using a given systems analysis method, allowing a method to evolve over time is generally preferable to discarding it in favour of a new one.

5 Conclusion

It is one of the aims of this paper to move the debate away from technical issues and towards more real-world concerns such as the needs of client and analyst discussed in the preceding sections. There is by no means universal acceptance amongst the object-oriented fraternity of the need for analysis as a distinct activity. For instance, a prevalent view is that objects 'are there for the picking'. Therefore, the emergence of methods specifically for object-oriented analysis is welcome. However, some work will be necessary before these methods can be said to be suitable for general use. In summary:

- The content of some of the methods is not as revolutionary as we might be led to believe, and therefore most organisations now using structured analysis or data modelling would have little difficulty in using them.
- Further investigation is needed to see if the reuse of analysis models will save effort when applied in large organisations.

- It would be wise to adopt only those aspects of the methods which actually improve the resulting information system. This means taking a pragmatic approach to choosing techniques, and not using techniques simply because they are 'object-oriented'.
- The confusion between objects in the system and objects in the world, and the lack of an intermediate conceptual model, are cause for concern. The idea that object-oriented systems are a 'natural' representation of the world is a seductive but dangerous over-simplification. In reality, the fact that the object model seems so close to reality makes it far easier to misuse it than other modelling techniques which do not purport to represent the world so directly.
- The methods will not become widely accepted if they cannot be used from the earliest stages of analysis. The concept of a prior narrative requirements specification needs to be viewed with some caution.
- Some simplification and standardisation of the diagrammatic techniques will probably be necessary to ensure their usefulness as a vehicle for communication between analyst and client.
- Explicit support for high-level views of an application domain must be present in any method which is to have wide applicability. How this is to be achieved in object-oriented analysis is open to question and is the subject of current research efforts.
- If we recognise that the sum total of all objects owned by an organisation is effectively equivalent to the corporate database together with all applications which use the database, then the risks inherent in taking a prototyping-style approach to both data and processes, as recommended in several major methods, need to be carefully considered.
- The object-oriented methods are similar in content to structured analysis methods. To be useful in the earliest stages of analysis, and to capture more application domain knowledge, they might well benefit from augmentation with other methods.

Object-oriented analysis methods are new and relatively untested. It is important that the different methods are now critically evaluated by being put into use. Applying them in a variety of practical situations will allow their practicality to be investigated and will help their effectiveness to be improved.

References

1. Avery G, Baker E (1990) *Psychology at Work*, Prentice Hall.

2. Bailin S C (1989) 'An object-oriented requirements specification method', *Communications of the ACM*, **32**, 5, 608-623.

3. Battaglia G (1991) 'Strategic information planning: a corporate necessity', *Journal of Systems Management*, Feb. 1991, 23-26.

4. Bjorner D, Hoare C A R, Langmaack H (eds) (1990) *VDM and Z: Formal Methods in Software Development: Proceedings of the Third International*

Symposium of VDM Europe, Kiel, FRG, April 17-21 1990, Springer-Verlag.

5. Booch G (1991) *Object-Oriented Design*, Benjamin Cummings.

6. Carey J M (1988) *Human Factors in Management Information Systems*, Ablex Publishing Corp.

7. Coad P, Yourdon E (1991) *Object-Oriented Analysis*, Yourdon Press.

8. Colbert E (1989) 'The object-oriented software development method: a practical approach to object-oriented development', *TRI-Ada '89 Proceedings*, Oct. 1989.

9. Colter M (1984) 'A comparative analysis of systems analysis techniques', *MIS Quarterly*, **8**, (1), 51-66.

10. Cooper R B, Swanson E B (1979) 'Management information requirements assessment: the state of the art', *Data Base*, **11** (2), Fall 1979, 5-16.

11. Cutts G (1987) *Structured Systems Analysis and Design Methodology*, Paradigm.

12. Davies L J (1991) 'Assessing the business impact of information systems failures: a risk analysis methodology', *Computer Control Quarterly*, **9** (4), Oct. 1991.

13. de Champeaux D (1991) *A Comparative Study of Object-Oriented Analysis Models (Technical Report HPL-01-41)*, Hewlett Packard Laboratories, Palo Alto CA.

14. Duke R, King P, Rose G, Smith G (1991) *The Object-Z Specification Language, Version 1*, Technical Report 91-1, Software Verification Research Centre, University of Queensland.

15. Essink L J B (1988) 'A conceptual framework for information systems development methodologies', *Information Technology for Organisational Systems*, H J Bullinger et al (eds), North-Holland.

16. Eurinfo '89 (1989) *A Conceptual Framework for Information Systems Development Methodologies*, Eurinfo 89, 354-362, North-Holland.

17. Floyd C (1986) 'A comparative evaluation of system development methods', *Information Systems Design Methodologies: Improving the Practice*, Olle T W et al (eds), North-Holland 1986.

18. Gibson E (1990) 'Objects - Born and Bred', *Byte*, Oct. 1990, pp 245-254.

19. Hackathorn R D, Karimi J (1988) 'A framework for comparing information engineering methods', *MIS Quarterly*, June 1988, 203-220.

20. Henderson-Sellers B, Edwards J M (1990) 'The object-oriented systems life cycle', *Communications of the ACM*, **33**, 9 Sep. 1990.

21. Jacobson I (1987) 'Object-oriented development in an industrial environment', *OOPSLA '86 Proceedings*, ACM.

22. Kay S (1990) 'Demand up for experts in information engineering', *Computerworld*, Sep. 21 1990, p 53.

23. Kurtz B D et al (1991) *Object-Oriented Systems Analysis and Specification: A Model-Driven Approach*, Prentice-Hall.

24. Lyytinen, K (1987) 'A taxonomic perspective of information systems development: theoretical constructs and recommendations', *Critical Issues in Information Systems Research*, Boland R J, Hirschheim R A (eds.), Wiley 1987.

25. Maddison R N et al (1984) 'A feature analysis of five information system methodologies', *Beyond Productivity: Information Systems for Organisational Effectiveness*, Bemelmans T H (ed.), North-Holland 1984.

26. McGinnes S (1991) *A Framework for the Comparison of Requirements Analysis Methods*, internal report available from School of Information Systems, Queensland University of Technology.

27. Mullin M (1990) *Rapid Prototyping for Object-Oriented Systems*, Addison-Wesley.

28. Nijssen G M, Halpin T A (1989) *Conceptual Schema and Relational Database Design: a Fact Oriented Approach*, Prentice Hall.

29. Odell J J (1991) *Object-Oriented Analysis*, IFIP Working Conference on the Object-Oriented Approach in Information Systems.

30. Olle T W (1988) *Information Systems Development Methodologies: A Framework for Understanding*, North-Holland.

31. Parnas D, Clements P (1986) 'A rational design process: how and why to fake it', *IEEE Transactions on Software Engineering*, Feb. 1986.

32. Pasmore W A (1988) *Designing Effective Organisations: the Sociotechnical Systems Perspective*, Wiley.

33. Rumbaugh J et al (1991) *Object-Oriented Modelling and Design*, Prentice-Hall.

34. Seligmann P S, Sol, Wijers (1989) 'Analysing the structure of information systems methodologies: an alternative approach', *Proc. 1st Dutch Conf. on IS, Amsterdam Neths*, Nov 1-2 1989.

35. Shlaer S, Mellor S (1988) *Object-Oriented Systems Analysis*, Prentice Hall/Eaglewood Cliffs.

36. Sutcliffe A (1990) 'Human factors in information systems: a research agenda and some experience', *Human Factors in Information Systems Analysis and Design*, A. Finkelstein et al (eds), North-Holland.

37. Sutcliffe A (1991) 'Object-oriented systems analysis: the abstract question', *Object-Oriented Approach in Information Systems*, F. Van Assche et al (eds), North-Holland.

38. Teng J T C, Sethi V (1990) 'A comparison of information requirements analysis methods: an experimental study', *Data Base*, Winter 1990, 27-39.

39. Vitalari N P (1984) 'Critical assessment of structured analysis methods: a psychological perspective', *Beyond Productivity: Information Systems for Organisational Effectiveness*, Bemelmans T H (ed.), North-Holland 1984.

40. Weiss S, Page-Jones M (1991) *Synthesis: An Object-Oriented Analysis and Design Method*, Macmillan.

41. Wilson B (1984) *System Concepts: Methodologies and Applications*, John Wiley.

42. Wirfs-Brock R J, Wilkerson B, Wiener L (1990) *Designing Object-Oriented Software*, Prentice-Hall.

43. Wood J, Silver D (1989) *Joint Applications Design: How to Design Quality Systems in 40% Less Time*, Wiley.

44. Yourdon E (1989) *Modern Structured Analysis*, Prentice-Hall.