Entity Relationship models and Object Oriented models: a one-to-many relationship?

Christina Davies[1] , Brian Lazell[1,2], Jill Doake[1], Ishbel Duncan[1]

[1] Computer Science Department, Anglia Polytechnic University, East Road, Cambridge CB1 1PT, UK. (cdavies@csd.anglia.ac.uk). [2] Medicines Control Agency, 1 Nine Elms Lane, London SW8 5NQ,UK.

Abstract

In this study a comparative evaluation was carried out of OO and ER (Entity Relationship) based design techniques. It was hypothesised that, using ER modelling, relational data analysis and functional decomposition, an Object Oriented (OO) design could be produced which was at least as good as that produced using an explicitly OO methodology.. The study consisted of re-working a published OO derived design (for a word processing system); the ER model and Data Flow Diagrams produced in this reworking were then mapped on to an OO design and the two designs evaluated using a range of simple metrics.

It was noted that the ER model consisted of fewer classes and demonstrated lower coupling between classes. The differences were not large, but the ER derived design performed the functions of two of the sub-systems defined in the OO design. It could thus be argued that the ER derived design is likely to prove to be simpler to implement and easier to test and maintain than the OO exemplar. It is suggested that these differences are due to the fact that, although using ideas inherited from ER modelling, OO modelling does not emphasise the need to establish a minimal data model.

1. Introduction.

1.1 Background

Exponents of Object Oriented (OO) modelling vary in the extent to which they acknowledge a debt to Entity Relationship (ER) modelling and Relational Data Analysis (RDA) (see for example [2], [16], [17], [22]). The methods of design offered for OO systems have been developed by researchers from a variety of backgrounds, some of whom had an earlier background in ER modelling and relational theory and some of whom have no such prior knowledge. Proponents of the new techniques all assume, without necessarily testing their assumption experimentally, that 'conventional' data modelling techniques will be inadequate for OO modelling and that a new approach is needed. Our suggestion is that, on the contrary, ER and relational modelling together produce the minimal data model which is best used as a basis for the OO data model and that associated techniques of functional decomposition and production of Data Flow Diagrams (DFD) are adequate to assign methods to the data objects. The object of the study described in this paper was to test further this hypothesis that these techniques can be used reliably to produce an OO design which is at least as good as that produced by newer techniques specifically intended for design of OO systems. It carries on from an earlier study [8] in which we used ER and relational techniques [3], [6], to produce a rival design to one published as an instance of good OO design (Haythorn [13]).

In that study we followed Haythorn in scoring the two designs by measuring the percentage of code that must be understood in order for a programmer to make changes; this measure had been proposed by Haythorn as giving an estimate of the relative

maintainability of designs. Results of that study suggested that a design obtained through ER modelling and RDA (referred to for convenience in the rest of this paper as 'ER derived') could be mapped on to an OO design which was at least as 'maintainable', according to this measure, as one presented by an experienced OO designer as a good OO derived design.

In the study to be described here, we again compared an OO with an ER derived design, but for a much larger case study. The case study used here is one detailed in a standard text (Wirfs-Brock et al., [23]). In that text an OO approach is used to design a word processing system. We were interested in this case study since it provides a challenge to the relational and entity-based approach: it is not an Information System as most people would define one. Methodologies based on ER modelling and RDA, such as SSADM, are presented specifically as being unsuitable for real-time applications [21]. We reasoned that if we could produce a good design for this system using ER modelling and associated techniques then the case for suspecting that such techniques remain appropriate for producing OO models for Information Systems, even where there are good reasons for using an OO implementation language or DBMS, is strengthened. The word processing system is the hard case against which we test our hypothesis. Our earlier paper [8] tackled a relatively small system needing only simple data structures (a queue manager for an automated queuing system in a bank). Here we attempt the design of a relatively large system involving complex in-memory data structures.

For the purposes of this experiment, the authors concentrated on two object-oriented diagramming techniques used by Wirfs-Brock - the Object Model (called a Hierarchy Graph in the text) and the Collaborations Graph.

We must stress that we are not concerned here with the debate between advocates of OO and relational databases, but with the best design technique for arriving at a good OO design. We do not argue that OO programming languages are not enormously useful, nor offer an opinion one way or another as to the relative merits of OO and Relational Databases. Rather, this paper is concerned entirely with the question of evaluating design techniques for a system for which an OO implementation has been chosen.

2. Procedure.

2.1 The Experiment.

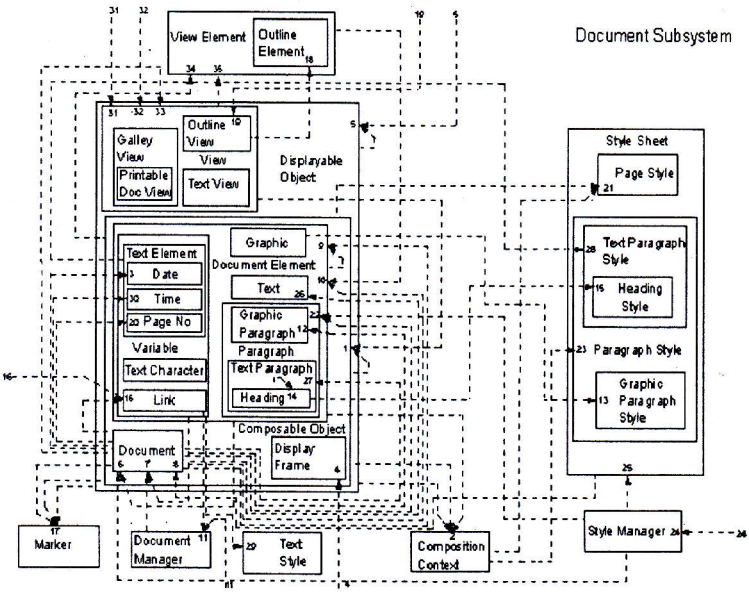
The experiment was based around the OO design for a word-processing system published in a standard text [23]. Two of the present authors attempted the task of producing a new design for part of this system, using a sub-set of the techniques of SSADM [21]. The resulting relational design was mapped by them on to an OO design in such a way as to match the diagramming technique of Wirfs-Brock. The other two authors, experienced in OO design techniques, then carried out measurements on both designs to try to arrive at a comparative evaluation.

2.2 The Target OO Design and its derivation.

The case study concerns a standard word-processing system for a windowing environment. As the system is a complex one, Wirfs-Brock starts her design process by splitting the word processing system into sub-systems. She describes sub-systems as *'groups of classes that collaborate to produce a clearly delimited unit of functionality'*

[23 p135] but sets about finding her sub-systems in the same way as she finds classes, ie by looking for nouns and noun phrases in the problem specification. Having made her decisions about sub-systems, Wirfs-Brock goes on to identify classes. The noun-phrase analysis is followed by the discarding of unsuitable nouns or noun-phrases - "*things outside the system .. noun phrases representing attributes of other things... and phrases that seem obviously spurious*" [23 p205]. Attributes and responsibilities are then allocated to classes and relationships between classes (generalization-specialization and collaborations) are established. The design is expressed in the form of a Collaboration Graph (see Figure 1). Sub-classes are shown as nested within the encompassing rectangle of their parent class. Classes have 'contracts' with each other. A contract is the list of requests that a client class can make of a server class. Both must fulfil the contract: the client must make only those requests that the contract specifies, and the server must respond appropriately. These contracts form collaborations between classes and are modelled as lines joining classes; each line is numbered with the appropriate contract number. Wirfs-Brock sees the Collaboration Graph as a description of all of the paths along which information can flow between classes covering all possible scenarios. As such it can be seen as a time lapse exposure of all the routes traversed as message passing takes place between a set of classes as each possible scenario is executed.

Figure 1. Collaboration Graph for OO derived design (after [23])



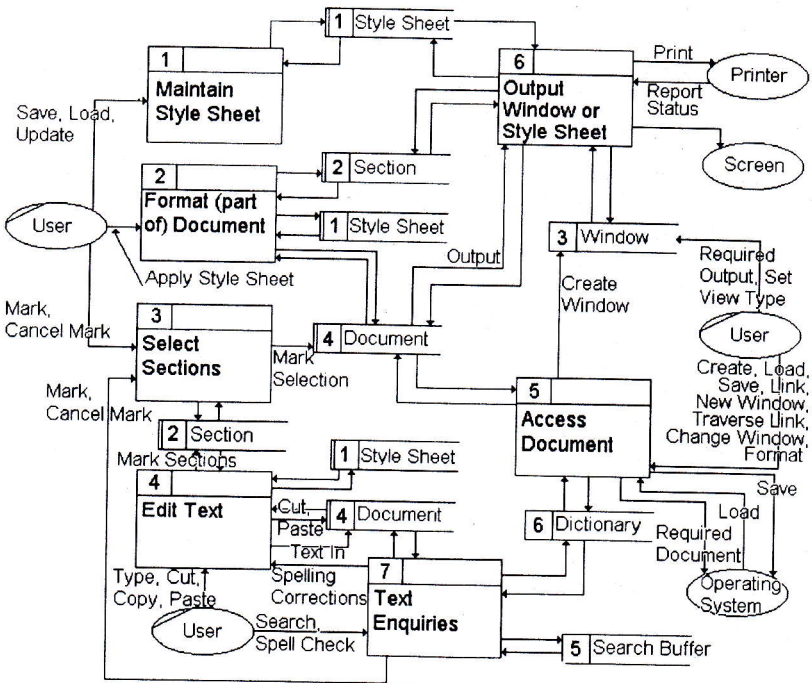
2.3 Producing the ER derived Design.

The initial intention had been to accept the OO author's division of the Word Processing system into sub-systems and to produce, as she did, a design only for the Document Sub-system. However this proved extremely difficult. The functionality of the

Document Sub-system seemed to the ER designers to be inextricably tied up with that of the Editing sub-system. For this reason, the ER designers proceeded as follows:

1. They produced an overall data model, in the shape of an Entity-Relationship diagram, from the system description given in [23].
2. A set of DFDs was produced as a means of identifying processing required and to help confirm the data model. The top level DFD is shown in Figure 2.
3. From this ER diagram, hierarchy graphs were produced. This was achieved by identifying sub or super-types on the ER diagram; super/sub-type entities were then mapped directly as parent-child classes in the OO design. The hierarchy graphs thus produced were used directly to produce the classes to be placed on a Collaboration Graph for the complete system, excluding only the services provided by or for what the ER designers regarded as external entities (eg the Disk Operating System, printers etc.). These had been identified when producing the DFDs.
4. The processes identified in the top level DFD were used as the basis for identifying necessary collaborations between classes on our own version of the Collaboration Graph . (See Figure 3).
5. The functions carried out by each of our top level processes were checked with Wirfs-Brock's collaborations to ensure that our system had equivalent functionality to hers. If necessary the DFD would have been modified at this stage to make good any shortcomings in functionality.

Figure 2: top level DFD for the Word Processing system.



2.4 Metrics used.

Once the new OO design had been produced, both it and that of Wirfs-Brock were subjected to the set of metrics detailed below. Although they are in common use they all have problems associated with them and these are discussed in Section 4.2.

Number of Classes: this is a simple count of the total number of classes used in the design.

Number of Collaborations: total number of collaborations appearing on the collaboration graph. In the diagrams shown this is simply the total number of individual lines connecting two classes on the graph.

Fan-in and fan-out: the fan-in of a module M is defined as "the number of local flows that terminate at M, plus the number of data structures from which information is retrieved by M" [12]. Fan-out is defined as "the number of local flows that emanate from M plus the number of data structures that are updated by M" [12]. We report both the maximum and mean fan-in and fan-out for each of the two designs.

Depth of inheritance: for any node in a class inheritance tree, the Depth of Inheritance (DIT) is the length of the maximum path from the node to the root of the tree. We report the maximum DIT for each of the two designs.

Number of Children: the number of direct 'successors' of a class. We report the maximum number of children for any class in each of the designs.

Number of Disparate Class Hierarchies: the design process results in a family of inheritance trees. The number of disparate class hierarchies is the number of discrete inheritance trees, ie those that do not share any classes.

Most of these metrics are in use as measures of coupling between objects. Coupling between objects, as defined in [4] is, for a given class, *'the number of other classes to which it is coupled'*. The fan-out and fan-in of a class *'refer to the number of other collaborating classes irrespective of the number of references made statically or dynamically' [14]*. Some authors actually take fan-in and coupling between objects to be synonymous [14 p115]. For non-OO systems fan-in and fan-out are typically combined to give a measure of complexity of information flow [15], [20] and there have been studies validating these measures as predictors of maintenance effort [19]. Chidamber & Kemmerer [4] also propose Depth of Inheritance as measures of coupling; DIT indicates how many ancestor classes are able to affect a given class. The number of disparate hierarchy trees and the number of collaborations can also be taken as an indicator of the degree of coupling between classes within the system.

The number of classes, on the other hand, is a crude measure of the potential size of the system. Clearly, however, if two designs differ greatly in the number of classes we might suspect that those designs were based on fundamentally different treatments of the system requirements.

3. Results.

Figures 1 and 3 show the Collaboration Graphs for each of the two designs. We follow [23] in showing sub-classes as contained within the rectangle of their parent class. Numbers on or collaboration graph refer to the processes identified in the DFD (Figure 2). Table 1 was extracted from the information shown in these two figures.

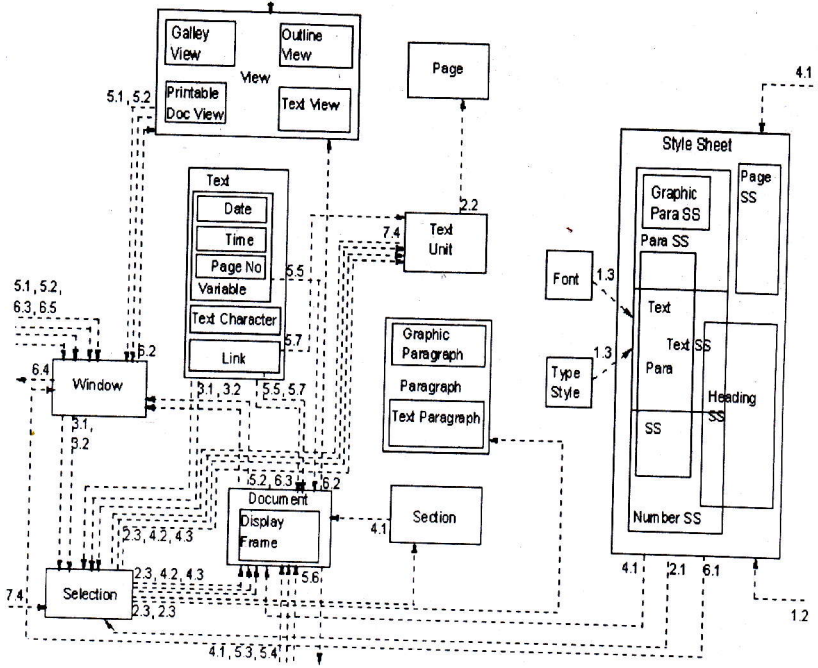


Figure 3. Collaboration Graph for ER derived design

Measure	OO derived design	ER derived design
Number of Classes (Total)	36	32
Number of Collaborations (Total)	49	41
Fan-In (Max)	5	6
Fan-In (Mean)	1.03	0.84
Fan-Out (Max)	12	4
Fan-Out (Mean)	1.06	0.66
Depth of Inheritance (Max)	5	3
Number of Children (Max)	4	4
Number of disparate class hierarchies	8	12

Table 1 : Comparison of Wirfs-Brock and Davies&Lazell models

The ER derived design showed the following characteristics:

- 10% fewer classes
- 15% fewer collaborations
- shallower depth of inheritance

- 50% more disparate hierarchies
- substantially lower fan-out
- slightly lower mean fan-in

All of these differences, although small, are in the same direction: the ER derived model shows less coupling and hence could be expected to be simpler to implement and maintain.

4. Methodological Problems.

Two types of problem arose while carrying out and analysing this experiment. Some specific problems relate to mapping from an ER model to an OO one, not in the sense that it was as difficult to produce such a design but in the sense that it was difficult to be sure that we specified our OO design in such a way as to provide a fair comparison between the two designs. The second type of problem is associated with the choice of, and interpretation of, metrics by which to evaluate the two designs.

4.1. Problems with mapping the ER derived model to a Collaboration Graph.

Wirfs-Brock distinguished in her collaborations between client and server. The processes identified by the ER modellers did not fall easily into such a classification, reflecting the fundamental difference in approach between the two design teams. In order to maintain some consistency in the direction of 'collaboration' on our graph, we showed collaboration as proceeding in the direction of data flow between the two classes involved. This was an arbitrary decision but as long as it is consistently applied has no effect on the complexity of the collaborations shown.

The decision as to what level of detail to go down to in the DFD when choosing processes to include on the Collaboration Graph was also of necessity subjective. However, in an attempt to ensure the comparability of our processes with Wirfs-Brock's collaborations we referred to the complete list of methods provided by her as a supplement to her Collaboration Graph. We checked to ensure that all of our processes together could provide the services carried out by all of her methods.

4.2 Problems with metrics

Metrics for design assessment are few and unsophisticated. Measurements such as the number of classes, fan-in and fan-out are the simplest and easiest to apply at early design stages. They are generally taken as predictors of maintenance effort [14] and this has some empirical support [20].

However, many metrics have been developed explicitly to enable one company to accumulate figures about its own productivity in order to cost new projects (eg COCOMO, [1] [10]). Less attention has been paid to the development and use of metrics to perform comparative evaluations of designs produced under different sets of conditions or by different development teams. Different development teams will be under different imperatives (whether to minimise lines of code, minimise function calls etc) and this will constrain their work. In our study, it is quite possible that the ER derived design offered here was unconsciously influenced by the required outcome of the study.

We have used here one particular interpretation of the term 'coupling'. However, there has been disagreement over a definition of the term 'coupling' and the extent to which peer-to-peer and inheritance coupling should be treated separately (eg [14, p112]). A more precise metric which takes account of the inheritance overheads of coupling is required; a simple count of the number of classes in a coupling thread might suffice. However, this is a very fine grained measure and the time taken to deliver a result would preclude its use for measuring the complexity of coupling of anything but small clusters of classes.

5. Discussion

5.1 Differences between the two designs.

In this study, an ER derived design demonstrated lower coupling than the corresponding OO design against which it was measured. The maximum fan-out for the Wirfs-Brock design was 12 and the average 1.06. In comparison our ER derived model had a maximum fan-out of 4 and average of 0.66. Thus, coupling dependencies were lower in the ER model. The number of disparate class hierarchies (8 in Wirfs-Brock compared with 12 in the ER derived design) echoes the nature of the decoupling, as does the lower total number of collaborations (41 compared with 49).

It has to be noted that there was some concern over the extent to which we were comparing like with like; classes from one methodology will describe a different type of activity, and therefore different states, from other classes developed using a different technique. However, the ER approach did result in 10% fewer classes than did the OO approach, and more than 15% fewer collaborations, in a system with greater overall functionality.. Although it is difficult to assess the effect of future alterations, this first cut design suggests that the ER approach would be easier to implement, test and maintain because of the smaller overheads of test harnesses and regression testing performed during maintenance. It is commonly accepted that fewer couplings and lower fan-out indicate that the system will be more robust under corrective or adaptive maintenance [14].

However, it is true to say that no measures of the overheads of inheritance were taken into consideration. Some collaborations are modelled in a way which make parent classes appear to collaborate when it is in fact their child classes that do. For example, one contract in the Wirfs-Brock model indicates a contract of the abstract parent class *View. Outline View*, its concrete child class, services this contract by collaborating with *Outline Element* through its parent *View Element.* A single coupling here actually involves routing through four classes. It is quite possible that such hidden complexity exists in the ER derived design.

Fan-in not only contributes to a measure of the complexity of information flow but can also be used as a measure of the re-use of a class, as it demonstrates the number of classes requiring contracts to be fulfilled by the calling class. The OO model showed a higher mean value of fan-in than the ER model, implying greater reuse of classes. Similarly the Depth of Inheritance metric (DIT) [4], which was greater in the OO model, not only measures coupling but also indicates degree of re-use. The DIT metric shows that the OO approach involves both more generalisation and more specialisation. This implies a higher level of re-use for classes within the inheritance hierarchy but it also

gives a higher overhead for testing. More information is required on the shape (especially the breadth) of the inheritance tree in order to determine likely maintenance overheads of each design.

Another interesting difference between the two designs lies in the reduction of the problem by Wirfs-Brock into one of the design of sub-systems. In ER derived design, sub-systems could be identified, if the system were complex enough for this to be desirable, by reference to the top level DFDs. One process box is expanded into successively greater levels of detail of process design and thus one top level process box could plausibly be taken as basis for a sub-system for design purposes. Thus any sub-systems identified would mirror units of functionality.

Wirfs-Brock's design technique involves first identifying sub-systems. For example, she decided that the editing functions of the WP system form a separate sub-system from the functions that concern the structure and visual representation of a document. Sub-systems are identified in the same way as classes, ie by looking for nouns and noun phrases in the problem specification. These, however, seem inappropriate; in this case their use has not produced sub-systems of cohesive functionality. The particular choice of sub-systems is largely responsible for extra message passing in the OO design.

6.2 Conclusions.

In spite of the problems identified with comparing designs derived from different methodologies, and in comparing designs at all, some conclusions can be offered.

The ER derived model can be accepted as at least as good as the OO model, on a range of metrics of design quality, and there are suggestions that the ER derived design may be slightly superior. For example, the ER derived design consisted of fewer classes and fewer collaborations between classes, for a design that had greater functionality than that of the OO design. It may well be that the ER design would need added classes when implemented (there are no abstract classes in our design) but there was nothing in the design process suggesting the need for such classes. Although some of the measured differences between the two systems are small, they are all in the same direction: together they suggest that the ER derived model could be implemented to give a system that is easier to test and maintain than could the OO derived design.

Equally interesting, however, are the similarities of the designs. The two designs have many classes in common; inspection of these classes at a high level of abstraction suggests that they have largely the same attributes and serve broadly similar functions. Exact functionality can only be evaluated by black-box testing at code level.

There are several possible reasons for the perceived design similarities. We introduced this paper by pointing out that OO authors differ in the extent to which they explicitly use techniques from ER modelling and relational analysis. It is possible that some who claim to use design techniques that have no link with ER modelling nevertheless retain habits of thought which are conditioned by knowledge of ER modelling and/or the relational database model. As an example, Coad and Yourdon [5] acknowledge that their Object-Oriented Analysis (OOA) builds on the *'best concepts'* from several different methods including Information Modelling, which they define as the use of Entity-Relationship Diagrams and Semantic Data Modelling. Their advice when trying

to identify *'pertinent Class-&-Objects'* is to look for useful real-world abstractions in the problem domain. In common with many other OO writers, they then offer a list of criteria to use in deciding which classes should be discarded and which should be expanded to more than one class. Many of these criteria have their roots in relational data analysis and normalisation theory. However, without an explicit foundation in the theory underlying relational analysis, such criteria are likely to prove difficult to apply. None of the rigour of ER modelling or normalisation can be harnessed since Coad and Yourdon explicitly exclude formal modelling.

Rumbaugh et al. [18] also acknowledge that they borrow many concepts from relational analysis. However, their methodology also debars the software developer from carrying out formal data modelling; it explicitly rules out use of the concept of unique identifiers. It is asserted that all objects have unique identity and are distinguishable by *'their inherent existence'* and not by the value of any of their attributes. Two objects could theoretically have identical attribute values and still have unique identity. They claim that most object-oriented languages automatically generate implicit identifiers with which to reference objects, there is no need to invent unique identifiers as in relational analysis, *'explicit object identifiers are not required in an object model .. they are computer artefacts and have no intrinsic meaning'* [18 p24].

Rumbaugh's methodology requires designers to identify associations between classes which are essentially equivalent to ER relationships. Just as in ER modelling, designers are asked to identify association classes, i.e. classes which break up many to many associations, thus avoiding the loss of information which characterises many to many relationships. In our experience of undergraduate teaching, students learning OO design find these concepts extremely hard to apply without reference to ER concepts. Unpublished studies in this Department of system designs produced by amateurs (eg research students in subjects other than computer science) with no knowledge of ER or OO design also suggest that relations (or classes) whose function is to resolve many-to-many relationships are those which these amateurs find most difficult to identify.

It seems that many OO authors acknowledge the desirability of the goals of ER design, but seem deliberately to avoid offering a rigorous set of steps to take to achieve these goals. One possible explanation, therefore, for the similarity between the two designs is that one implicitly uses techniques used explicitly by the other.

From this and our previous study we offer the hypothesis that for the one model based on ER modelling and relational analysis that would be produced by the majority of practitioners experienced in these techniques, there is a set of variants (which, we are tempted to suggest, are likely to be ill-advised variants) produced via OO modelling techniques which deliberately eschew the benefits of older forms of data modelling and function design.

There is one important caveat to offer: if, as we suspect, relational and ER modelling when carried out correctly will tend to result in more efficient designs, this superiority may be offset by ease of use of OO techniques. It may turn out that, although OO design methods lack the precision of RDA and ER modelling, they result in more people being able to produce more acceptable designs in a shorter time. This possibility is still to be fully investigated.

6. Summary and Further Work.

In this comparison between an OO and ER derived design, the following methodological problems were identified:

1. The particular OO design technique used in [23] resulted in fundamentally different sub-systems from ours. This made it difficult for the investigators to adhere to the OO specification for the particular sub-system to be modelled.
2. Although a mapping could be found from processes to collaborations it is not possible to defend this mapping rigorously and any such mapping relies on subjective judgement.
3. Although we used some metrics applicable at the design stage, validation of these few metrics must await implementation of both designs, or at least more detailed design.
4. Some metrics used at the design stage, such as depth of inheritance, will be heavily influenced by the development environment in which the designers work. This means that they must be interpreted with caution when used between, rather than within, individuals or groups. No one measure on its own will be sufficient to detect differences in designs.

The two designs, derived from two different design methods, were qualitatively substantially the same but showed small but consistent quantitative differences. Although the differences between the two designs were small, the ER model scored better on all measures. It had fewer classes and lower coupling and it had substantially greater functionality, encompassing the functionality of two of the sub-systems defined in 23]. On the basis of these measures, it seems likely that the ER derived design will be simpler to implement and maintain than the OO design. The difference in organisation of the designs into sub-systems seemed to result from aspects of the OO designer's design technique. The OO solution resulted in unnecessary message passing between classes which contributed to the quantitative differences between the ER and OO derived designs..

To follow up our study, both designs will be at least partially implemented and the predictions of the metrics reported here tested. More detailed measurements, especially Function Point Analysis or a measure derived from it [11] will be carried out on the designs discussed here and on lower level designs to be produced as part of the implementation phase. We also hope to tackle the much more difficult problem of evaluating designs for OO systems which have been produced by practitioners in "real life" environments. Field trials comparing design methodologies are extremely difficult to set up, but the importance of determining the relative efficacy of Information System Design techniques (what one might call "evidence-based computing") is important enough for considerable effort to be justified.

References

1. Boehm, B.W. (1981) *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ.

2. Booch, G. (1994) *Object-oriented Analysis and Design*, Benjamin / Cummings.

3. Chen, P.P. (1976) The Entity-Relationship Model - Toward a Unified View of Data, *ACM Transactions on Database Systems*, 1 (1).

4. Chidamber, S. & Kemerer, C. (1994) A metric suite for object-oriented design, *IEEE Transactions Software Engineering,* 20(6).

5. Coad, P. & Yourdon, E. (1991) *Object-Oriented Design*, Prentice Hall, Englewood Cliffs, NJ.

6. Codd, E. (1970) Relational Data Model, *Communications of the ACM*, 13(6).

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. & Jeremaes,

7. P. (1994) *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Englewood Cliffs, NJ.

8. Davies, C., Curtis, A., Doake, J. & Lazell, B. (1996) Object Oriented Design and Maintainable Code: a Relational Perspective, *Proceedings of 4th ECIS Conference*, Lisbon, pp659-672.

9. Davis, A. (1993) Software Lemmingineering, *IEEE Software* 10, pp79-84.

10. De Marco, T. (1982) *Controlling Software Projects*, Yourdon Press, New York.

11. Dreger, J.B. (1989) *Function Point Analysis*, Prentice-Hall, Englewood Cliffs, NJ.

12. Fenton, N.E. & Pfleeger, S.L. (1996) *Software Metrics, A rigorous and practical approach,* Thomson, London.

13. Haythorn,W. (1994) What is object-oriented?, *Journal of Object Oriented Programming,* March-April, pp67-78.

14. Henderson-Sellers, B. (1996) *Object-oriented metrics: measures of complexity*, Prentice Hall, Englewood Cliffs, NJ.

15. Henry, S. and Kafura, D. (1981) Software structure metrics based on information flow, *IEEE Transactions Software Engineering,* 7(5), pp510-518.

16. Jacobson, I. (1993) *Object-Oriented Software Engineering - A Use Case Driven Approach*, ACM Press.

17. Meyer, B. (1988) *Object-oriented Software Construction*, Prentice-Hall, Englewood Cliffs, NJ.

18. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorensen, W. (1991) *Object-Oriented Modelling and Design,* Prentice-Hall, Englewood Cliffs, NJ.

19. Shepperd, M.J. and Ince, D.C., (1990) The use of metrics in the early detection of design errors, *Proceedings of the European Software Engineering Conference*, pp67-85.

20. Shepperd, M.J. and Ince, D.C., (1993) *Derivation and Validation of Software Metrics*, Clarendon Press, Oxford.

21. SSADM (1992) *SSADM Version 4 reference manuals*, NCC Blackwell Ltd.

22. Shlaer, S., Mellor, S.J. (1988) *Object-Oriented Systems Analysis: Modeling the World in Data,* Prentice Hall, Englewood Cliffs, NJ.

23. Wirfs-Brock, R., Wilkerson, B. & Wiener, L. (1990) *Designing Object-Oriented Software,* Prentice-Hall, Englewood Cliffs, NJ.

24. Yourdon, E. & Argila, C. (1996) *Case Studies in Object Oriented Analysis and Design,* Yourdon Press, New York.