THERE IS NO OBJECT-ORIENTED ANALYSIS

David C. Hay
Essential Strategies, Inc.

The world's bookstores are now full of books concerned with "object-oriented analysis". Is it possible that the world of object-orientation has completely changed the nature of systems analysis? To listen to some of its aficionados, this must indeed be true. Information Engineering is passé, and has been completely replaced by object-oriented analysis.[1]

James Rumbaugh and his colleagues wrote one of the seminal books on analysis in the object-oriented world.[2] To their credit, they never used the expression "object-oriented analysis". They simply made analysis an important step in the object-oriented development process – as well they should have. The problem was that they then asserted that it was object-orientation that first made analysis data-centric[3], which is not the case. (Entity modeling has been around since 1977.) This attitude has had the unfortunate effect of permitting the object-oriented world to assume that any analysis that uses models of data structure is inherently "object-oriented". This then forms the basis for asserting that the old way of doing things is now completely invalid.

For an entire body of knowledge that has developed over twenty years or more to be summarily overturned by a new approach seems unlikely. This has never happened before, and it is difficult to believe that it has happened now.

More likely is the possibility that the object-oriented folks have added something to the body of knowledge. Perhaps, in addition to its contributions to design and programming, the object-oriented world has something to contribute to the field of analysis. Or perhaps not.

When data modelers refer to entities as the "things of significance to a business" (as they have done for some fifteen years, now), the argument could well be made that these are the "object classes" of the business. Occurrences of these entities are the business' "objects". When we address a business as object-oriented analysts, if we limit ourselves to business object classes and their associations, we are in fact being information engineers.

---

1 Oracle Education Services, "Oracle Education Mini-lesson: Object Database Designer for Oracle8 Objects", Oracle Development Tools User Group, (Palm Springs, CA:1998), p. 3.

2 James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen, *Object-oriented Modeling and Design,* Prentice Hall (Englewood Cliffs, NJ:1991), PP. 153-156.

3 Rumbaugh, *et al., ibid,* p. 146.

But creation of the expression "object-oriented analysis" itself suggests that more than the business objects we have been looking at are involved. The object-oriented analysis described by Mr. Rumbaugh and his colleagues is nothing other than that which we information engineers have been doing for a long time. For them to appropriate credit for these techniques is relatively harmless. More dangerous is the argument often made that object-oriented analysis is somehow *more* than regular analysis. The problem is that this "more" isn't really analysis.

Some of the books your author has encountered (including Rumbaugh's) do limit their view of the analysis of data to the objects of significance to the business.[4,5] Other authors, however, do not limit themselves to business objects, nor do they even seem to be oriented toward identifying business needs.

First of all, they are not truly oriented toward the business. This is evident in the language used. As a minor example, consider the names of entities and attributes. In UML and other "object modeling" notations, spaces are not permitted between words in entity and attribute names. If, however, this is to be a model of the business – validated by human beings – then conventional English must be used. The expression most business people will recognize is "purchase order", not "PurchaseOrder" or "Purchase_order". The statement being made by this convention is that the model is being prepared, not in service to the people of the business, but in service to the computer.

(It is true that some practitioners do use spaces between words,[6] and the original specification for UML does not specify a rule either way[7], but the prevalence of those who don't allow spaces says something profound about the state of the industry.)

The problem in fact goes beyond spaces between words in names: in at least one of the books your author has read, object classes (definitions of things of significance to the business, remember) are called things like "CustInfo" instead of "Customer".[8]

Another author at least spelled out "StudentInformation", but she then defined this class as "Information needed to register and bill students." As an afterthought she acknowledged that a "student is someone who is registered to take classes at the University",[9] but this wasn't the definition of the class.

---

4 Rumbaugh, *et al., ibid.*

5 Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design,* Prentice-Hall PTR (Upper Saddle River, NJ:1998).

6 Martin Fowler, *UML Distilled,* Addison-Wesley (Reading, MA:1997).

7 Rational Software Corporation, *Unified Modeling Language Notation Guide*, Rational Software Corporation, (Santa Clara, CA:1997), pp. 42-44.

8 Paul Harmon and Mark Watson, *Understanding UML: The Developer s Guide*, Morgan Kaufmann Publishers, Inc., (San Francisco: 1998).

9 Terry Quatrani, *Visual Modeling with Rational Rose and UML,* Addison-Wesley, (Reading, Massachusetts:1998).

The business object classes are *not* "student information" and "customer information" (even spelled out with spaces). The business object classes are "students" and "customers".

Grady Booch, in his seminal work on *Object-oriented Analysis and Design,* was unable to define his principle concepts without using program code. This made it a little tough for those of us who do not know C++ to understand what he was talking about. For example, he defined the sub-type/super-type relationship with the following bit of pithy text:

```
// Yield type
typedef unsigned int Yield;

class FruitGrowingPlan : public GrowingPlan {
public:

   FruitGrowingPlan(char* name);
   virtual "FuitGrowingPlan();

   virtual void establish(Day, Hour, Condition&)
   void scheduleHarvest(Day, Hour);

   Boolean isHarvested() const;
   unsigned daysUntilHarvest() const;
   Yield estimatedYield() const;

protected
   Boolean repHarvested;
   Yield repYield;
};[10]
```

Apparently this identifies a "FruitGrowingPlan to be a sub-type of "GrowingPlan". Please forgive your author if he does not use this definition when next he tries to introduce the business community to data modeling.

Second, these books add object classes that are not oriented toward identifying business needs. There are two categories of such classes.

First, there are object classes that constitute what are simply bad models:

In one case, for example, the additional object classes "CustInfoMulti" and "CustInfo1" were defined as separate sub-types of the unfortunately named CustInfo entity referred to above. These sub-types were defined to distinguish customers with just one address from those with multiple addresses.[11] This is nonsense, since the underlying *object class* is still "customer". The number of addressees it has should be shown by the association between Customer and Address. The association "Each Customer may be at one or more

---

[10] Grady Booch, *Object-oriented Analysis and Design,* Benjamin/Cummings Publishing Company, Inc., (Redwood City, CA:1994), p. 61.

[11] Paul Harmon and Mark Watson, *ibid.* p. 150-152

Addresses," asserts that if a Customer may have one or more addresses, it certainly may have one. Or it may have more. There is absolutely no reason for there to be two sub-types.

Now bad models should not be held against object-orientation. After all, information engineers have produced their share of bad models as well. Your author has found only a few object-oriented books,[12,13,14] however, where the models were not perverse in the manner of the above example. The issue is not what could be done, but what object modelers are taught to do. The fact of the matter is that rarely are they taught to respect the underlying structure of the information.

A second category of object-oriented "extensions" to the data model are classes that in fact belong to the domain of function modeling and physical design. These come from the object-oriented developer's emphasis on behavior. To describe this, though, it is worth a minute to discuss entities and behavior:

OBJECTS AND BEHAVIOR

The most profound difference between the object-oriented approach to modeling and the information engineering approach is the inclusion of behavior in the class definitions. As an approach to software *design* this has been a significant addition to the body of knowledge in our industry. The extension of the idea to *analysis* models, however, is more problematic.

In the case of basic classes, it is simple enough to add documentation of behavior, without affecting the underlying structure of the model. The issues determining what classes should be there in the first place would not be affected. Indeed, the idea of adding behavior to an entity definition is not a bad one, and could result in useful insights. As Bob Brown has said, "entities are just objects that don't know how to behave."[15] To do so during analysis, however, the behavior involved would have to be true business behavior, and not the behavior anticipated to be designed into a system.

The examples found in most texts don't describe business behavior. They are written in a kind of structured English that looks suspiciously like program code.

Indeed, Grady Booch says, "We must emphasize that analysis focuses upon behavior, not form." While your author, a data bigot, wouldn't be happy about it, this is not be an unreasonable position to take. The problem is that he then goes on to say, "It is inappropriate to pursue issues of class design, representation, or other tactical decisions

---

12 James Rumbaugh, *op.cit.*

13 Martin Fowler, *op. cit.*

14 Craig Larman, *op. cit.*

15 Bob Brown, "Extended Modeling Language", GUIDE Proceedings, November 1990.

during this phase. Rather *analysis* must yield a statement of what *the system* does, not how it does it."[16] [Emphasis added.]

In other words, his version of object-oriented analysis is not concerned with object data models at all. (So much for Rumbaugh's assertion that object-oriented invented data-centric analysis.) In Mr. Booch's view, object-oriented analysis isn't about the articulation of the business requirements for a system. He as much as said that object-oriented analysis is about system design. Indeed, even to the extent that analysis is concerned with functions, they are the functions of a yet-to-be-designed system, not of the business.

Things get stranger when a logical conclusion of the object model approach is that you can have "objects" with behavior but no data. In particular, there are two categories of object classes like this: "Control classes" are used to control processing in a system, and "boundary" or "interface classes", that "interact with things outside the application".[17] Neither of these describe business data structure, and indeed, following Mr. Booch's definition above, they don't describe business function, either. They essentially programming constructs, describing a hypothetical system.

For example, one text specified "ProfessorCourseManager" and "AddACourseOffering" as classes.[18] The first is neither data nor function, but merely the fact that a professor can control a course. If this were presented as an associative class between Professor and Course (and named something like "CourseManagement"), where an occurrence of the class is the role of a Professor's managing a Course, it would be a legitimate class. But as presented, it is only related to Course, and represents control of the course, not a fact about it.

The second is simply a function in disguise. In fact even an object-oriented analyst would recognize it as a method that should be captured *inside* the CourseOffering class.

Your author once sat in on a presentation of an object model which consisted of an object class that performed some function, then handed off data to another object that did another function, which in turn handed other data off to yet a third object that did something else.

Excuse me, but how is this different from information engineering's data flow diagram? (Except that it is being presented as a data structure diagram.)

Yes, it is true that information engineering produces process models. At this point the argument could be made that it is an extension of the object-oriented philosophy that function boxes and control boxes should appear along with data boxes on object models – but adding behavior to an object class is not the same thing as adding function boxes to an entity/class diagram. They really are different things, and should be the subject of different models.

---

16 Grady Booch, *op. cit.*, p. 252.

17 Paul Harmon & Mark Watson, *op. cit.* pp. 123-125.

18 Terry Quatrani, *op. cit.* p. 54.

As mentioned above, this doesn't mean that it wouldn't be useful, in a business context, to describe the life cycle of an entity/object class. What functions create occurrences of it? What update them? What delete occurrences? These are legitimate extensions of the entity/relationship modeling technique. A technique called "entity life histories" (which predates UML and other object-oriented analysis techniques by many years), addresses exactly this issue.[19] It was originally described by Michael Jackson (no, not the popular singer) in his 1983 book, *System Development*,[20] and it has been incorporated into the SSADM methodology widely used in Europe.[21]

While there is a good argument for adding the behavior of an entity to the entity's description, cobbling together object and function *models* simply confuses the issue.

In short, there is no such thing as "object-oriented analysis". There is only "analysis", to which the object-oriented folks have added a dollop of object-oriented design. While object-oriented analysis has been offered as a fundamentally new way of doing analysis, what is really new is that this version of analysis contains elements specifically oriented toward the *implementation* of systems using object-oriented technology.

To understand why this is misguided, it is necessary to understand John Zachman's "Framework for Information System Architecture".

THE ZACHMAN FRAMEWORK

By 1989, even though the industry was awash with methodologies, modeling notations, and different ways to "communicate" with each other, we in fact rarely communicated very well at all – with each other or with our clients. In response, John Zachman came up with some important insights, and from them developed his "Framework for Information Systems Architecture".[22] Among other things, Mr. Zachman realized that a source of our communications problems was that each of us views the problem of information system development from a quite different point of view. Mr. Zachman's recognition of these points of view was important, because it finally clarified why it is that we have such difficulty talking to each other. In particular, he identified six different perspectives in any system development project:

---

19 David C. Hay, "Object Oriented Data Modeling: Entity Life Histories", Oracle CASE Special Interest Group, 1993. Also available at *http://www.essentialstrategies.com/publications/objects/elh.htm.*

20 Michael Jackson, *System Development,* Prentice Hall, (Englewood Cliffs, NJ: 1983).

21 Ed Downs, Peter Clare, and Ian Coe *Structured Systems Analysis and Design Method: Application and Context,* Prentice Hall INternational (UK) Ltd, (Hemel Hempstead, Hertfordshire:1988).

22 Zachman, John, "A Framework for Information Systems Architecture", *IBM Systems Journal*, Vol 26, No 3, 1987.

- ***Scope*** – The understanding of why the organization exists, how it is like other organizations in the same industry, and what makes it distinctive.
- ***Business owner s view*** – This refers to the perspective not of the stockholders, but of the people who operate the business. This encompasses all the jargon of the business, as well as an understanding of how everything actually works.
- ***Architect s view*** – Recognition that there are fundamental, technologically-independent structures present, and representation of the business in terms of these structures.
- ***Designer s view*** – The application of technology to address any information system requirements discovered in the views above. The perspective here is technological: relational data base systems, object orientation, network protocols, and so forth.
- ***Builder s view*** – This is the view of the inside of the programs and technology. The builder knows the finer points of the programming language, or the communications technology, or whatever.
- ***Production view*** – The view of the completed system.

If two people are coming at the problem from different perspectives, their language and terms of reference will be different, and they will be working at cross purposes, unless they recognize the differences in their points of view – and translate. With the translation it becomes clear to both why they believe what they do – and it becomes clear that, as long as they recognize and respect each other's positions, this doesn't matter. Each has a different job to do, and each's perspective is appropriate to the carrying out of those jobs.

Note that this is not an argument for the "waterfall approach" to developing systems. The waterfall addresses the perspectives in sequence: first, strategy addresses scope; then, analysis deals with the business owner's and architect's views; later, design captures the designer's view; and so forth. But this is not necessary. If you choose to start with design, you may do so. The Framework simply tells you what it is you don't know when you do that. That is, you don't know the business owner's perspective, so you will have to guess at what the system is to do. You don't know the architect's perspective, so you will have to cobble the system together as best you can, without benefit of any understanding of fundamental structures.

Are you willing to risk the consequences of not knowing those things? If so, go right ahead.

If you decide to do a little bit of analysis, followed by a little bit of design, and then return to analysis, that's fine. Again, it's your choice. You can now understand the shortcuts you are taking and the potential costs of those shortcuts. If you decide to do analysis and design at the same time, the Zachman view simply requires that you recognize you are dealing with two perspectives at once, and you must be willing to accept the implications of this.

WHAT OO MISSES

The Zachman Framework now makes it possible to describe what is wrong with the object-oriented world's approach to analysis. Specifically, object-oriented analysis begins with a cross between the business owner's view and the designer's view, moves directly to the designer's view, and never addresses the architect's view at all.

Yes, Grady Booch specifically describes the importance of architecture, saying that "A system with sound architecture is one that has conceptual integrity and, as Brooks firmly states 'conceptual integrity is *the* most important consideration in system design.'"[23,24] And yet we saw above just how much architecture Mr. Booch puts in analysis. It is true that Mr. Rumbaugh and his colleagues devote a large part of their book to analysis, but if you look at the details of how object-orientated analysis is too often carried out, you can see that architecture – based on the architecture of the enterprise to be served – gets pretty short shrift, on both the function and data side of the effort.

Functions

A popular technique in the object-oriented world is "use cases". These are represented as diagrams of specific anecdotes offered by people in the business being examined. They are supposed to describe the specific activities carried out by an individual. As such they should be effective at capturing the business owner's view.

As it happens, however, they don't actually do that.

"A use case diagram provides a functional description of a *system* and its major processes and places a boundary on the problem to be solved. It also provides a graphic description of who *will* use the system and what kinds of interactions they *can expect to* have with the *system*."[25] [Emphasis added.]

Note that a use case describes a future "system", not the existing enterprise. Moreover, it is about a future world when a new system will make all the business person's problems magically disappear. A use case does not actually describe the physical world presently experienced by the user. ("There are five copies of a purchase order and the gold one goes to accounting".) Instead, it describes a job in terms of the technology anticipated for the system (complete – in 1998 – with lap-top computers and web access[26]). Rather than describing the current business, it describes an imagined system – one that hasn't even been designed yet.

It is the beginning of a user-interface design. That's all.

As it happens, many of the business processes observed by the analyst are not really intended to carry out the functions of the business at all. They are only there to accommodate inadequacies in current systems, and it is a waste of time for us to try to

---

23 F. Brooks, *The Mythical Man Month,* Addison-Wesley, (Reading, MA:1975), p. 42.

24 Grady Booch, *op. cit.,* p. 230.

25 Paul Harmon and Mark Watson, *op. cit.* p. 112.

26 Paul Harmon and Mark Watson, *op. cit.* p. 116.

automate them. Instead of trying to do that, we should be asking, what is the business trying to accomplish with these processes the analyst sees? What are the underlying functions to be carried out? What are the "essential" processes (without regard for the technology that might be used) required to carry out the functions the business wants to do? How can technology bring us closer to providing the true functions of the business?

These questions are never asked.

Even if the use cases could be used to infer something about the current nature of business systems, developers then typically go from this supposedly business owner's view directly to the designer's view. There is often no attempt to examine the set of use cases either to determine which should be automated or to identify general patterns of behavior or underlying functional structures. Nor is there provision for analyzing a set of use cases to make sure that they neither overlap nor leave out important activities.

Data

If the object-oriented approach doesn't come to grips with the underlying *functions* of an enterprise, what about its *data*? We have already seen Mr. Booch's view of the importance of understanding data structure at all during analysis. Indeed, to the extent that it is done, it typically only addresses the objects seen by the users, without trying to understand their inherent structure.

First of all, since object-oriented technology does not require normalization, this step is often skipped, precluding development of a sound understanding of the underlying structure of the data involved.

When Dr. Codd invented the relational model, and with it "normalization", the idea was that the normalization techniques were required to make relational databases work. With the appearance of object-oriented database technology, presumably normalization is no longer required.

This would be true, except for the fact that the advent of normalization had a much more profound effect on the industry than the mere facilitation of relational databases: it has helped us systematically to eliminate data redundancy, and, more importantly, to better *understand* the *nature* of data. Each datum in an enterprise is *about* something – and only one thing. Our job as data analysts is to determine what that thing is. Moreover, there is only one datum that is that fact about that thing, and we must determine where to put that fact so it can be found again.

If we don't correctly identify the relationship between data and the things they describe, we haven't done our job.

A second problem with the object-oriented approach to data is that it tends to identify object classes simply from the use cases, as the objects seen by the potential system users. This approach fails to ask an important question: What are the *fundamental* things of significance to the business, of which the things most people see are but examples? The business objects described in use cases are those that the individual sees in his own area of activity. Different individuals may be perceiving what are really the same objects, but with different names and characteristics. One department sees "customers"; another sees "vendors"; another sees "clients"; yet another deals only with relationships between departments in the same company.

Are not all these things simply instances of "parties" (people and organizations) in different roles? Indeed will not many parties play several of these roles?

Going beyond the boundaries of the enterprise itself, are there not similarities between the fundamental structure of this enterprise and those of other enterprises in the same or different industries?

Object-orientation is supposed to promote re-use: the place to start should be in the identification of standard entity/object models for standard business situations.[27]

In fairness, many object-oriented people have acknowledged the importance of understanding common structures while doing requirements analysis,[28] and at least one book has now taken up the issue of doing a proper conceptual architecture[29,30], but these views are not reflected in a large part of the literature.

To summarize, in ignoring the architectural view, many object-oriented practitioners fail to ensure that they are addressing the entire enterprise. They fail to examine the organization as a whole and to understand fundamental structures and processes – those which cut across the views of the individuals who run the business. There is no attempt to understand those structures which will exist regardless of any technology employed to deal with them.

Yes, it seems to be true that object-oriented programming makes it easy to change a design when something turns out to be wrong. But this approach to analysis will guarantee that there are a lot more wrong things to fix than would be the case if some architectural thought were given to the system before it was designed.

CONCLUSIONS

It has been difficult to write this article. The criticisms of object-oriented analysis presuppose that we know what object-oriented analysis is. As with any technique it is in fact practiced very differently by different practitioners. Many who claim they are object-oriented are in fact following the best of information engineering techniques. Indeed, many information engineers are guilty of modeling sins far more egregious than those presented here. Your author is the first to acknowledge both of those points.

The problem is language. Our industry is highly susceptible to fads, especially in terminology. The fact of the matter is that the subject at hand is – as it has always been – nothing other than, how do we do a better job of building systems? Each year people come up with new techniques to further the craft. Unfortunately, new sets of techniques

---

27 See David Hay, *Data Model Patterns: Conventions of Thought*, Dorset House, (New York: 1997)

28 Fowler, M, *Analysis Patterns: Reusable Object Models*, Addison-Wesley (Reading, MA:1997).

29 James Rumbaugh, *et al., op. cit.*

30 Craig Larman, *op. cit.*

invariably get packaged with a new name and are presented as if they have completely overturned everything that went before. This is never true. The new techniques are always incremental additions to the body of knowledge and only that – fancy names notwithstanding.

This is the problem with the way object-oriented analysis has been presented. Of course object-orientation has contributed greatly to the programmer's art.

The assignment for requirements analysis, however, has not changed. It remains that of understanding an enterprise in technologically neutral terms. The objective is to determine the data, function, and other requirements for information processing – expressed in terms that are both clear to the business customer of that processing, and which enable *any* technology to be applied in addressing them. Nothing in the object-oriented arsenal of tools has changed that. Requirements analysis is fundamentally a dialogue between the system architect and the potential system user.

Many of its practitioners in fact do recognize that if object-oriented development is to succeed, its practitioners must acknowledge and adopt the valuable techniques which preceded the advent of object-oriented programming. A proper business analysis is still required. Disciplined understanding of the nature of the data that describe objects remains important. We have known how to do these things for a long time. We can't afford to throw this knowledge out.

In short, there is no such thing as "object-oriented" analysis.

ABOUT THE AUTHOR . . .

A thirty year veteran of the Information Industry, Dave Hay has been producing data models to support strategic information planning and requirements planning for nearly twelve years. He has worked in a variety of industries, including, among others, power generation, clinical pharmaceutical research, oil refining, forestry, and broadcast. He is President of Essential Strategies, Inc., a consulting firm dedicated to helping clients define corporate information architecture, identify requirements, and plan strategies for the implementation of new systems. He is the author of the book, *Data Model Patterns: Conventions of Thought*, recently published by Dorset House.

He may be reached at davehay@essentialstrategies.com,
http://www.essentialstrategies.com, or (713) 464-8316.