Martin Fowler ≋ fowler@acm.org
Independent consultant in Boston, Massachusetts

Is There Such a Thing as Object-Oriented Analysis?

Recently, I was running a workshop on object-oriented analysis. At one point, a participant exclaimed: "Is this analysis or design—and what's the difference anyway?" It's a thorny question, one that bugs me from time to time, and always a good candidate for a flame war on comp.object.

People imagine that it's easy to ask someone to build a computer system. You find a programmer, tell them what you want, and they go away, get some pizzas, and build it. Yet as we all know, it is devilishly difficult to actually do this for anything but the smallest system. For a start, there is a lot to tell. Just writing it all down is a lengthy exercise in itself—hence the term *Victorian Novel Specification.* The other awkward problem is that people will tell you what they *want,* but you actually have to build them what they *need*—not necessarily the same thing.

In theory, analysis provides a statement that describes the problem domain, without saying anything about a solution. Such a statement should not construct anything artificial about the problem domain and it should be technology independent. The software people can then just design a solution to this statement. Pity things don't work that way.

In this theory, the term object-oriented analysis is an oxymoron. Objects are a design idea—such things as encapsulation, polymorphism, and the like make sense in developing maintainable software; but if you think of me as a lump of encapsulated data with an interface of operations, you need more than an optician. Having said this, when you see object people do analysis, the models they come up with are clearly different in nature than what a relational data modeler would come up with. To see this, compare the patterns books of David Hay and myself (Fowler, M., *Analysis Patterns: Reusable Object Models,* Addison Wesley, 1997; Hay, D., *Data Model Patterns: Conventions of Thought,* Dorset House, 1996); we both claim we are doing analysis modeling, but our style of modeling is very different and clearly influenced by the technology.

Is analysis about just recording how the world is, or is it about designing something new? Few modelers will actually model what happens in the business. There are frequent contradictions and inconsistencies between users and departments, and words are used in different ways. We try to abstract, and thus simplify our analysis models, yet such abstractions are *constructed*—you can't really say they are *in the world.* Can we, should we, be passive describers when we analyze? And if we are not, are we really doing design rather than analysis?

The key to understanding this question is to remember that all we are doing in analysis is coming up with a description of our understanding of the domain. This description could be prose, or could use UML. Whether we grace this description with the term *model* or not, it is still a description that we construct, not some universal reality we pluck out of the problem domain; we are building an artificial description of it. As Jim Odell famously said, we do not model the world, we model our perception of the world. To do this we need some language to state our description. To argue that one language is more natural than another is to argue that a wooden bridge is more natural than a steel bridge. Either way, it's a bridge, and it's constructed, not natural.

So, to argue that one modeling language is more natural than another is unimportant. Naturalness is usually in the eye of the beholder. The real question is whether one language is more *useful* than another.

To answer this new question of usefulness, we must understand what we are building the description for. For most practitioners the purpose is to build software. A good description is one that helps us build good software—that is, software that helps the users in their task for a reasonable cost. The description that comes from analysis is only as good as its usefulness in this task.

Is it more useful to be technology independent? Is it better that my analysis statement should be equally amenable to objects, relational databases, functional programming, or any other design paradigm? Let's assume for a moment that such a thing is possible. Eventually we have to implement a solution. At that point, we have to transform it to the technology we eventually build with. Such a transformation carries a cost, and if we want to keep the analysis picture up to date we will pay an ongoing cost. Is this cost outweighed by the advantages of a technologically independent model?

My opinion is that it is perfectly reasonable for your analysis model to lean toward the implementation technology you intend it to use if, as a result, the model is more helpful in understanding the problem you are working on. For me, the essence of the difference between analysis and

continued on page 41

design is that analysis is primarily about *understanding the domain,* while design is about *understanding the software that supports that domain.* Clearly the two are closely connected, and the boundary between them can often become pretty blurred. But the boundary need not be sharp. We shouldn't let purity come before usefulness. In theory, having a model that is both analysis and design is some hybrid that isn't good at either, yet I believe that this kind of hybrid makes the best model.

Why? My view is that the key to the usefulness of an analysis model is that it's a communication medium between the software experts and the business experts. The software experts need to understand what the model means in terms of building software, and the business experts need to understand that the business process is properly described so the software experts are understanding the right thing.

So, what is an analysis model, and what makes it different from a design model? For me the issue is one of emphasis. Remember, analysis is about understanding how the domain works. I build this understanding by building models, either in my mind, in UML, or in a programming language. I can be working out the details of some charging algorithm by programming it, or in a discussion with users about what they mean by the word *asset*. I wouldn't say that figuring out database interactions or a distributed computing architecture was analysis. But my analysis models are still driven by the same heuristics as any other object-oriented software. Analysis is essentially the design of the business objects and business logic. This isn't everybody's definition of analysis, but I find it useful.

So object-oriented analysis is a reasonable notion. It is building a description of the domain that increases our understanding of that domain, in order to help us build an object-oriented software system. Judging what is a good analysis is judging its usefulness for that purpose. This is not a definition that lends itself to a clean separation between analysis and design, but then such a separation isn't really useful anyway. As in engineering, we are faced with blurred boundaries and trade-offs. That's just the way life is.