# An open letter to object technology newcomers

**From AC**

<< previous article | next article >>

**Alistair A.R. Cockburn** (http://alistair.cockburn.us/) **Humans and Technology**

This report is an html draft of HaT.TR.96.02, which may be submitted for external publication . This early version is being made available for professional peer communication. Since this file was auto-converted and touched up, some html marks may be bad. Please let me know if your browser objects.

## Contents



# Intro

It used to be the habit of authors to write an apology for adding another piece of writing to the face of the earth, and a justification. The need for apology still holds, the justification for this article is this: Over the last 6 years, I have spent numerous and pleasant long evenings in restaurants, bars, and hotel lobbies, discussing the nature of object orientation with other interested seekers of truth, light and wisdom. Over the years, we have been able to answer many of the questions we asked at the beginning. The other night I spent such an evening in a restaurant discussing those same questions asked by a new colleague. The next day, his office-mate wanted the same questions answered, and suggested I capture the discussion for his other colleagues to read. Considering the number of like-minded colleagues he has around the world, asking the same questions, it seems sensible to write it for them all to read.

The key questions were:

- "Why is it that some OO experts get really upset when I do or say anything that is not strictly and purely object-oriented, but use cases are so popular, while not being object-oriented, and object modeling looks so much like data modeling?"
- "Would the model produced by a data modeler look the same or different as the structural part of an object model, and would the model produced by a process modeler look different or the same as the behavioral part of an object model?"
- "Why do we use use cases and scenarios, instead of just modeling the structure of the business?"

The questions are asked repeatedly by new arrivals to object orientation, and cannot be answered without having certain experiences. They come right to the heart of working with object orientation on a daily basis. So they are worth receiving consideration and serious response.

---

### "Why is it that some OO people get really upset when I do or say anything that is not strictly and purely object-oriented, and yet use cases are so popular, while not being object-oriented, and object modeling looks so much like data modeling?"

My answer comes in 3 steps.

"When Bob <invented name for generic OO expert> and I get together to talk about OO, there is a shared understanding we have of each other, that we are experienced and convinced users of object technology. We understand and expect to use object identity, polymorphism, combined data and behavior, instance responsibilities, inheritance. So when I say, 'Tomorrow we'll do data modeling on the application form,', Bob does not think that I have abandoned objects in favor of relational

tables, but expects that we are discussing the *structural* aspects of the situation, which will show up in the object model. He forgives my use or misuse of the term because he has associated certain expectations with my speech.

When Bob visits your organization, just learning object technology, he sees a strict separation of data from behavior, and an absence or neglect of object identity and polymorphism. So when you say 'data modeling', he comes down on you like a ton of bricks. He wants to make sure that you get clear about the shifts you need to make. You will notice that over the last months, your model (and modeling) have slowly acquired object identity, polymorphism, and combined behavior with data. It is perhaps now less dangerous to use the word 'data modeling', but there is the chance that he will think you are sliding back into previous work styles. In other words, he does not have those safe expectations with respect to your speech, and so you are obliged to speak carefully.

In the end, even an object model has *structural* aspects and *behavioral* aspects. We use those words to signal that we are not making the mistake of doing 'just' data modeling or 'just' process modeling. In point of fact, modeling the *structural* aspects is just a particular form of data modeling, being a bit careful with the meaning of the term data modeling. It is not that we are modeling the tables on the disk, but modeling the structure of the information that will be captured. Data modelers refer to this as the 'conceptual data model', as opposed to the logical or physical data model. It is something they regularly produce.

On the other hand, if two OO people are talking, and one starts talking about, 'process modeling', the other does not have a safe interpretation to give. Does the person really mean process modeling with standard data-flow diagram? These have been shown over the years to give great difficulty when going to an OO implementation, and so the conversation is about to enter a difficult period. Does the person mean the *behavioral* aspects of the model? Does the person mean modeling a process as an object in itself? This is interesting because it is not done that often. So 'process model' is not safe to insert into the conversation because the intent is too ambiguous.

Getting finally to use cases and scenarios, they are primarily requirements-gathering techniques, and are neutral to the implementation technology. They contribute a value in guiding the conversation during design, giving it context and scope. The fact that they are not 'object-oriented' is neither good nor bad. The fact that they resemble functional decomposition scares some people, but is also neither good nor bad. The important consideration is that they contribute value to the design process. When they are used to describe internal designs, they present a picture of the behavior of the system. Procedural code, flow charts, interaction diagrams, Petri nets, data-flow diagrams, use cases, all show aspects of behavior, and have different uses, advantages and disadvantages. As long as you know that objects have behavior as well as data, you can work open the discussion about how best to capture or describe the behavior within and across objects."

---

## "Would the model produced by a data modeler look the same or different as the structural part of an object model - would the model produced by a process modeler look the same or different as the behavioral part of an object model?"

"In my experience, there are two groups of data modelers - those who model the data on the disk, and those who model the information in the system or in the organization. There is a world of difference between the models they produce and the discussions one can have with them.

Those who model the data on the disk talk and think in concrete terms, often in terms of tables. Their models are often quite different from OO models, and they are unwilling to see the similarities that exist across the technologies. In the data modeling community, it is legitimate to say that these people produce logical or physical data models, without being insulting.

Those who model the information in the system produce the conceptual data model. My experience is that the models they produce are very similar to the models produced by experienced OO modelers. These people typically work a lot closer to the business people than do the logical data modelers, which perhaps explains the difference in their results.

Also, there seems to be a body of knowledge, lore, training, or examples that let these people come to their results faster than OO people do. I have time and time again seen OO people go through three or four design iterations, ending up with the same model that the (conceptual) data modelers produced on the first round. So I have a lot of respect for the data modeling community. In fact, when I get stuck in the object design, I sometimes run and snatch a copy of what the data modelers produced, to look at where we are likely to end up.

I had the experience of facilitating a session between warring data and object modelers. We decided to do 'CRC sessions for an audience'. The four experienced OO designers sat around one end of a long table and were the only active participants. The

business experts sat next along the table. They were there to answer questions and correct misstatements about the business. After them were the data modelers. At the far end of the table were other interested people. All told, there were over a dozen people in the room, but only four active people, talking mainly among themselves.

After about an hour of work, a section of the object design had emerged. We asked the data modelers what they had produced in their previous work, whether it was essentially the same or quite different. They said, 'Essentially the same.' We repeated this for two more segments of the design and got response from the data modelers each time that we had produced essentially the same design as they had, apart from certain obvious differences in notational technology. They had note used inheritance, but had the same placeholders in the same places. After that we were able to split into groups containing one business expert, one data modeling expert and one OO expert, and they quickly reconciled the rest of the design, identifying where there was a simple difference in technology, and sorting out minor differences. Where there was a difference, it sometimes happened that the data modelers had a better rationale, and sometimes the OO models had a better rationale, and the groups were able to resynchronize their designs.

So there have been at least a few experiences showing the similarity between results of conceptual data modeling and OO modeling with CRC cards.

There were also experiences showing the difference between logical (information-on-disk) based relational modeling and OO modeling. For the most part, the differences were due simply to technology, free use of inheritance and many-to-many relations in the OO model. The difficulties came with the inability of the participants to communicate across those technological differences.

So much for the data modeling part of the question.

The same cannot be said at this time for process modeling. Numerous methodologies, plans, conversions and projects were tried in the early 1990's to adapt data-flow diagram models to objects. A few projects may have laid claim to success, but in the end there seems to have been quiet consensus that they are difficult to map to object models. Most data-flow-diagrammers-become-OO-designers have dropped them (the Martin-Odell and Ptech groups being notable exceptions). My answer at this time is that process modelers do not produce a model comparable to what OO modelers produce.

The answer is not closed, though. OO people are beginning to model process *per se*. It is not clear how this will evolve. Will the processes start looking like steps in a procedure (procedural programming reincarnated), will they look like data-flow diagrams, or will they look like encapsulated objects?

---

## "Why do we use use cases and scenarios, instead of just modeling the structure of the business?"

"Use case and scenarios ...

... guide the conversation, giving it scope and context,

... indicate what to include, exclude, how wide, how deep to go, and when to stop,

... provide variations to stress-test the design.

I have watched a number of groups do a random exploration of the domain when they model without working from scenarios. How does the leader know what questions to ask next? From intuition guided from some previous experience. How does the group know if they are capturing the information they will eventually need? They often don't, and don't know it, or do know it and use intuition and guesswork.

Several really expert OO designers have confessed in the late of the night that they do not have a straightforward formula for getting to the beautiful objects. One said, "We have really wild discussions at times", another agreed to the notion that, "Sometimes we just beat our heads against the walls until eventually some decent objects show up."

Using scenarios does not prevent wild discussions and beating heads against walls until decent objects show up. Using scenarios provides a context and boundaries for those discussions so that they stay within the area of interest. Without the scenarios, I watch groups go mad as they wander through an arbitrarily large modeling space for days or longer. So the first reason to use scenarios and use cases is to channel the effort.

The second use of scenarios is to determine when you are done.

Consider that you have been given the job of modeling a trucking company. What do you model? Would you include the new purchase value of the trucks... the actual purchase value... the resale value... the make... the color of the interior... the number of speeding tickets issued against it... the number and destinations of its trips? If you try to include everything, you will never finish. Where do you start, when do you stop, what do you include, exclude, and how much detail do you want?

An answer is provided by the scenarios. You need to have sufficient information to deliver all the questions posed by the scenarios. On the structural or full object model, every box (line, phrase) addresses some need of a scenario. If you color the items on the model red as you walk through a scenario, the red items will all be connected in some way (there will be no islands). If you do this for all scenarios, all items will be colored at the end. There are no boxes (lines, phrases) extra, there are no boxes (lines, phrases) missing. Then you are done. During the modeling work, the conversation may go wild from time to time, but there is a way to tell if you have strayed out of the bounds of the current need, or wandered too deep.

Finally, scenarios provide variations to stress test the design and establish its quality.

How do you distinguish the better between two models? Saying, 'It matches the business' is necessary, but not sufficient. There are many possible models that 'match the business.' What is the measure of quality?

Software has a very high cost function associated with making changes. For a change request, the cost grows rapidly with the area of damage (the 'trajectory of change'). The goal for software, then, is to change exactly one module. This will not always be possible, but as Kent Beck says, 'There is a noticeable difference in the quality of the software when you can change just one class.' For a business model, it is less clear what the cost function attaches to, but minimizing area of damage is a reasonable place to start.

To test the trajectory of change, you need variations. The scenarios provide the variations. 'What happens when the user wants to <something-or-the-other>?' How many items on the model do you have to touch? Some techniques, such as the CRC ('class-responsibility-collaborator') technique encourage you to invent *impromptu* scenarios rapidly, to uncover likely, but unstated future changes. At the end, you review the likely changes, and see how many items you have to alter to make those changes. For two models that match the business, that one is better which has fewer (Kent says, 'One') touched points for each scenario.

You will also find variations in other places, such as the structure of related products. Can you change products touching just one place? Both sources of variations should be used.

There are other uses of scenarios, such as checking requirements with users, and providing functional test cases for the system. These are the three reasons to use scenarios during the modeling activity itself."

---

<< previous article | next article >>



Categories: Articles | Use cases | OO design | Articles by date | Year, 1996