

Marc J. Balcer *Chief Scientist, ModelCompilers.com*

In the last column I introduced the concept of a solution model—a model of an application that is complete and testable but that is not tied to a particular implementation technology. But how do you get started creating a solution model?

The Problems

When first starting on a project, it is very easy to fall into the trap of fixating on a small number of obvious classes. For our online bookstore, we might focus on the tangible objects in the problem such as Product, Customer, and Order. While easy to create initially, a model consisting of just tangible things can become very complex when we try to do the dynamic (state) models.

A second problem is a tendency to include classes that are not properly part of the application domain. Classes such as EntityBean and QueryScreen belong to the realization of the solution, but not the application domain (the bookstore) itself.

Getting started on a modeling project requires starting with a good collection of candidate classes of all kinds—not just tangible things like Customers and Products, but roles, incidents, interactions, and specifications.

Many of us in the “write about how to create models” community have written about different techniques for identifying classes. I’m sure you’ve read everything from, “underline the nouns in the requirements document” to “be a <class>. Now, who are your friends?” Some are practical, some make for good show, and some are even effective. One technique that’s been effective for me over many years is a community brainstorming session commonly known as an “object blitz.”

Preparations

Effective meetings don’t have to take a lot of time. For an object blitz, schedule a conference room for a two-hour period and divide the time into three 40-minute segments with scheduled breaks in between. (Breaks are a necessary evil. They keep the participants sharp but can also be terrible distractions. Know your team).

Appoint a group moderator and a scribe. The moderator’s job is to control the group so as to keep the discussions fruitful. The scribe’s job is to record the output from the meeting. While the moderator can be one of the analysts, the job of the scribe should not be assigned to one of the analysts because team members should be focused upon analysis or note-taking, but not both. When I am consulting to a group, I will usually take on the moderator/scribe job the first time we do an object blitz, but then I will give the responsibilities to client team members on subsequent iterations.

Defining the Scope

The first step (just 10 minutes or so) is to ensure that everyone understands the scope of the system. Go around the room and ask each team member, “In one sentence, describe the purpose of the domain.” Record these sentences on a flip chart or whiteboard. Then merge the sentences together into a single mission statement for the domain and attain consensus.

Pay very special attention to the assumptions implicit in the sentences. If one individual assumes that there will be a separate service domain to manage inventory while another does not, you will certainly end up with very different sets of objects!

Brainstorming

Then ask the group to think of objects in the domain, paying particular attention to the mission statement defined in the first step. It’s important here to brainstorm: Don’t discuss the candidate classes and don’t argue whether one is appropriate or not. The goal is to maintain a rapid pacing and to stay focused on task. A strong moderator is required if the group tends to want to discuss every class.

Years ago I made the analogy between brainstorming and popping popcorn: It takes a while to get warmed up, then everyone gets going fast and furious for a while, and finally the popping slows down. When this happens, you’re done. Don’t try to find the very last class—that’s the job of the formal analysis process.

Rough Classification

In this final phase, the goal is to weed out irrelevant classes that don't belong in the domain. For example, in the bookstore application, "Unix Task" belongs to the architecture or operating system domain; it is not part of the application domain.

Sort through the candidate classes one by one, sorting them into three piles: relevant, irrelevant, and maybe. When there are disagreements, place the class into the "maybe" pile. Sort all the classes first, then discuss them.

Once the classes have been sorted, go through each of the classes in the "maybe" pile. Ask the person who nominated the class to provide a two-sentence description of the class and how it relates to some of the other classes in the "relevant" pile. Note the description on the back of the card for future reference. Rely on the domain experts for guidance here: if they say, "what's that?" the candidate class is likely in trouble!

When to Start

"That's nice," you say, "but we don't have complete requirements yet." What are you waiting for? Complete, consistent requirements? In your dreams! Here are some reassuring rules you can use to start incrementally developing models.

Rule #1: Trust what you know

No matter how unusual a problem may be, there are some ideas you can rely on: ideas like "a thing can't be in more than one place at a time" and "customers may be placing several orders concurrently."

The result of this first rule is to answer the questions that should be answered by the application experts in the most reasonable way possible. Playing this game quickly identifies additional questions and builds your own confidence in the subject matter.

Don't be afraid to be wrong. I learn a lot more by being told I'm wrong than to be confirmed right. Show your in-progress models. Get feedback. Laugh at your silliness.

Sometimes the experts aren't so expert and you wind up providing real insights. As a consultant, I'm the outsider and expected to bring a fresh, unsullied perspective to the problem.

Rule #2: You can't model what you can't sketch.

I have to credit Leon Starr[2] for that insight. Basically it means that you need to be able to understand the concept—not just create box-and-line diagrams. While models are abstractions of the real world, they should not be so abstract that the subject-matter experts can't recognize their subject matter in the models.

While models are abstractions of the real world, they should not be so abstract that the subject matter experts can't figure them out.

Rule #3: Become your own expert

Sometimes the experts aren't so expert and you wind up providing real insights. To build effective models, you must base those models on real facts about the domain. Analysis is a learning process and effective analysts must learn the subject matter as well as, if not better than, the business's subject matter experts. I call this "cramming the domain." Amaze the so-called real application experts with your knowledge of the domain.

Rule #4: Work incrementally. Model the normal first.

Analysis paralysis is a fear of commitment. It's easy to get overwhelmed by "but what ifs." Any model, no matter how comprehensive, can be considered incomplete because some condition hasn't been included. But there comes a point where the desire for completeness is outweighed by the need to deliver something.

Sketch out the simplest complete model by making reasonable simplifying assumptions.

An easy rule is to use "one" whenever reasonable. For example, start by assuming only "one warehouse" and "one shipment" (don't split an order). But don't follow this rule blindly—if, for example, splitting the order into multiple shipments is easier to model, then let *that* be the simplifying assumption.

This rule is likely the most important of all. "Complete" is relative to the needs of the moment. Make the simplifying assumptions and build models that are "complete" in that they meet the needs of the assumptions.

Conclusion

"A journey of a thousand miles begins with a single step." Approach application modeling incrementally as well. Stay focused on the real-world problem. Start by modeling normal behavior. Get frequent feedback.

Next month we'll look at some of the problems you can encounter when developing solution models.

References

- [1] Marc J. Balcer, "How to Conduct an Object Blitz." *Objects and Projects*. 1:1 (Autumn 1993).
- [2] Leon Starr, *Executable UML: How to Build Class Models*. Prentice-Hall, 2001.