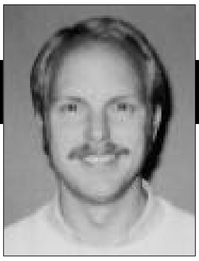

Farewell and a wood pile

Kent Beck

IT'S THE OBJECTS, STUPID

SOMETIMES IT TAKES me awhile to see the obvious. Sometimes even longer than that. Three or four times in the last month I've been confronted by problems I had a hard time solving. In each case, the answer became clear when I asked myself the simple question, "How can I make an object to solve this problem for me?" You think I'd have figured it out by now: got a problem? make an object for it.

Here's an example: I had to write an editor for a tree structure. There were several ways of viewing and editing the tree. On the left was a hierarchical list. On the top right was a text editor on the currently selected node of the tree. On the bottom right was a list of text editors on the subnodes of the currently selected node (see Fig. 1).

Figure 2 shows the domain objects that live behind this view. How is the editor going to work? Let's say I have an editor on the value 5 (Fig. 3). How are we going to write the code to parse and install a new function? The first part is simple enough:

```
FunctionEditor>>parse: aString
    | new |
    new := FunctionParse parse: aString.
```

But now we're stuck. If we just say:

```
function := new
```

then the "right" instance variable of the *BinaryFunction* (which the editor knows nothing about) won't be updated.

"Make an object for it," that's the ticket. The object is an *EditingWrapper*. When you go to edit a function, you first wrap every node in the function tree, as shown in Figure 4. Now the editor looks like Figure 5. And we can write the parsing method like this:

```
FunctionEditor>>parse: aString
    | new |
    new := FunctionParse parse: aString.
    function function: new
```

Kent Beck has been discovering Smalltalk idioms for ten years at Tektronix, Apple Computer, and MasPar Computer. He is the founder of First Class Software, which develops and distributes developer tools for Smalltalk. He can be reached at First Class Software, P.O. Box 226, Boulder Creek, CA 95006-0226, 408.338.4649 (voice), 408.338.3666 (fax), or by email at 70761,1216 (Compuserve).

If we parsed the string "*@years*", the resulting picture would look like Figure 6. When the *BinaryFunction* unwraps its children, the right function will be in place.

As I said, several times in the last month I've faced baffling problems that became easy when I asked myself the question, "How could I make an object to solve this problem for me?" Sometimes it was a method that just didn't want to be simplified, so I created an object just for that method. Sometimes it was a question of adding features to an object for a particular purpose without cluttering the object (as in the editing example). I recommend that the next time you run into a problem that just doesn't seem like it has a simple solution, try making an object for it. It won't always work, but when it does it's sweet.

THE PARABLE OF THE WOOD PILE

The following is really about software. Really.

I live in the redwood forest. Fall in the forest has its own set of smells, distinct and different from the smells of every other season. Crushed dry ferns have a sharp, dusty

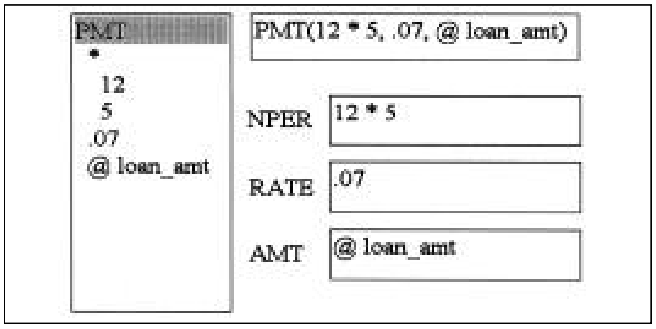

Figure 1.

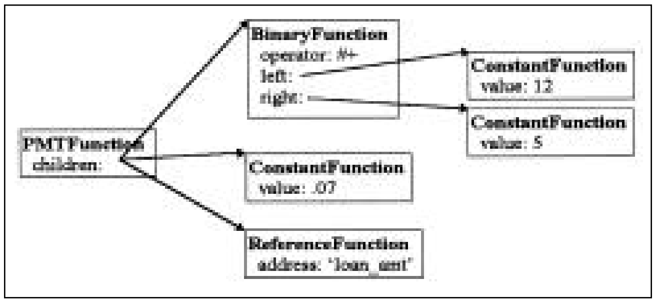

Figure 2.

smell. Rotting bay nuts are like psychedelic bay leaves. When we get our wood delivered, the smells of freshly split oak and madrone add to the mix.

My house is down by the creek, maybe 25 feet below the level of the driveway. There is a sheer cliff off to one side and stone steps directly in front of the house. When we get our customary two cords of wood delivered (for you city folk, that's a pretty damn big pile of wood, takes most of a 2-ton truck to carry it), the easiest way to get it down near the house is to throw it over the cliff, one stick at a time, then go down later and stack it.

Wood chucking time has become something of a ritual for me. The smells of the fall forest, the filtered fall light through the surrounding redwoods, the ache of my generally-desk-bound body, the knowledge that I'm keeping my family warm for the rest of the winter, all combine for a satisfying couple of days.

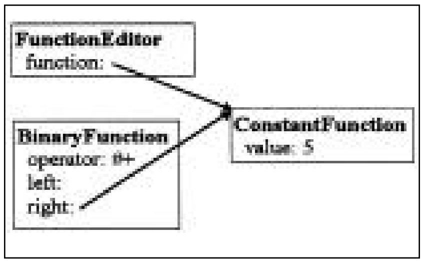

Figure 3.

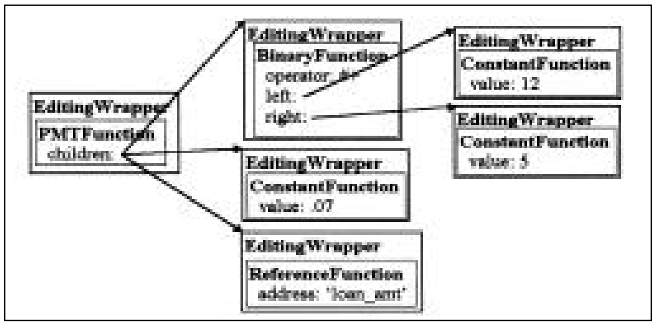

Figure 4.

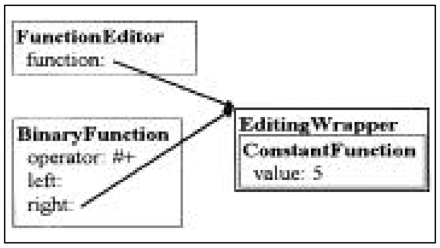

Figure 5.

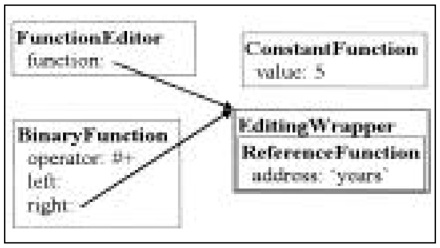

Figure 6.

My driveway is long and narrow, so when the truck delivers the wood it makes a long pile, maybe 25 feet long and eight or nine feet wide. The end of the pile is right at the top of the cliff, so the first hour or so is easy—turn, pick up a stick, turn, throw. Once I get settled into a rhythm, I probably throw a stick every five seconds.

This year we had a dinner party to attend, and I didn't want to have to walk over pile of firewood all dressed up, so I wanted to at least get a path cleared quickly. Once I got the sticks close to the top of the cliff thrown, I noticed that my progress slowed down. Instead of "turn, grab, turn, throw" I was doing "walk, grab, walk, throw," where I was having to walk a few steps to get to the front of the pile. It may not seem like much, but it slowed down my throwing rate by half. The more progress I made, the further I had to walk, the slower I went, the further my goal of walking to the car without scuffing my shiny shoes receded.

I'm an engineer at heart, and repetitive manual labor leaves me plenty of time to think, so I wasn't about to let this state of affairs continue without at least trying to bring my productivity back up. I discovered I could throw light sticks down with one hand. On every trip to the front of the wood pile I began picking up two sticks, a heavy one in my left hand and a light one in my right. I'd throw the light one one handed first, then heave the heavy one with both. This let me amortize my walking over two sticks. The pace picked up.

Pretty soon, though, I noticed I was still going slow. The front of the pile kept receding as I worked, so my time spent walking kept increasing. What I really needed was a way to get back to working like I had worked at first, just turning and throwing with no walking at all.

You've probably guessed the solution. I went to the pile and tossed sticks the 10 or 15 feet to the top of the cliff. I tossed 30–40 sticks, walked over, threw them down the hill, then walked back. This way my walking was amortized over so many sticks it didn't even count. I had to handle each stick twice, so my productivity was half of what it was at the beginning, but I could sustain the pace through the rest of the pile. No matter how far back the front of the pile got, it was always easy to quickly toss a little stack to the top of the cliff.

My wife and I made it to our party—shoes, suit, and dress unscathed.

When an experienced team starts a project in Smalltalk, the first few months go smoothly. The first cut at the domain model slides right in and away you go. Pretty soon, though, the team starts to bog down. Decisions they made without complete information begin to take their toll. The easy progress of the early days is soon but a fond memory. New functionality, rather than sliding in, has to be shoved in with a pile driver. Quality and predictability go out the window, because the team doesn't know if the next feature will fit with what's there, in which case all will be well, or it won't fit, in which case who knows how long it will take to shoehorn it in.

I have seen two unproductive reactions to this situation and one reasonable one. The first are the teams that keep walking back and forth to the wood pile, no matter how far it recedes. I call this "Smalltalk is more rope." These teams ignore the increasing risk and decreasing productivity, but Smalltalk is forgiving enough that they can keep their application sort of running while they add new features. Throw in enough *nil* checks and *isKindOf*s and you can make almost anything work. The result is disaster deferred. Eventually the team is asked to do something that just can't be shoved into the system.

The shell-shocked veterans of "more rope" failures often turn the other way. Ignoring the sticks right there in front of them, they try to toss the whole pile close before they start throwing down the hill. They insist on creating the frameworks first. The application is divided into strict layers and developers are only allowed to work on their own layer. The layers don't precisely fit, because they are developed in isolation, but developers have no choice but to carry on as best they can. The result is again disaster deferred. The system gets big, because layers provide services no one needs and because there is no view of the whole system that would allow large-scale simplifications.

The sustainable solution is to find a balance between moving the pile and tossing the logs. Toss some, move some, toss some, move some, starting with tossing. (Jeff McKenna had a great article about this years ago, and Ward Cunningham has a pattern language called Checks about the same idea: http://c2.com/ppr). Take advantage of the quick wins to give you concrete information from which to generalize. Make it run, make it right, make it run, make it right.

FAREWELL

This is my last column, at least this go around, for THE SMALLTALK REPORT. It's been quite a ride, programming in Smalltalk and trying to write about it. When I started, when THE SMALLTALK REPORT started, we were the wild-eyed purveyors of what many people saw as a crazy language. Since then, Smalltalk has become the language of choice for many kinds of applications. Recently the Smalltalk market has been thrown into turmoil by the merger of ParcPlace and Digitalk and their subsequent disappointing financial performance.

From that standpoint, it seems like a strange time to quit. I'd like to go out on a high note, with noble Smalltalk standing proudly head and shoulders above the crowd. However, when I saw that I wasn't putting the thought or care into these columns that they, that you, deserve, I knew the time had come.

I'll still be involved in the Smalltalk world, in fact, more than ever. You won't get rid of me that easily! I'll be unveiling a one-day Smalltalk patterns course at Smalltalk Solutions in March. I'm working on a book, THE SMALLTALK BEST PRACTICE PATTERNS: VOLUME 1, CODING, due out in the first quarter next year. I'm scrambling to keep up with my products. I'm working on some fascinating contract programs. To top it off, consulting has picked up since OOPSLA. The only way you'll be rid of me is if I drop dead of exhaustion.

Thanks

I'd like to thank all the people who helped me during the last few years. In particular:

- Rick Friedman, for giving Smalltalkers a forum for our voices when we were far out in the wilderness.
- John Pugh and Paul White, for all their work making the Smalltalk Report work well.
- Elizabeth Upp and the production team at SIGS, for dealing with late submissions, raw ASCII, and requests for odd graphics.
- Liz St. Pierre, for hassling me in the gentlest possible way consistent with results.
- Ward Cunningham, for help refining many of my best column ideas.
- You, THE SMALLTALK REPORT readers, for support, encouragement, email, and ideas. Without you I could have written all the columns I liked, but no one would have read them.

So long. I hope I see you Smalltalkin' down the road.