Design and Research

January 2003

(This article is derived from a keynote talk at the fall 2002 meeting of NEPLS.)

Visitors to this country are often surprised to find that Americans like to begin a conversation by asking "what do you do?" I've never liked this question. I've rarely had a neat answer to it. But I think I have finally solved the problem. Now, when someone asks me what I do, I look them straight in the eye and say "I'm designing a new dialect of Lisp." I recommend this answer to anyone who doesn't like being asked what they do. The conversation will turn immediately to other topics.

I don't consider myself to be doing research on programming languages. I'm just designing one, in the same way that someone might design a building or a chair or a new typeface. I'm not trying to discover anything new. I just want to make a language that will be good to program in. In some ways, this assumption makes life a lot easier.

The difference between design and research seems to be a question of new versus good. Design doesn't have to be new, but it has to be good. Research doesn't have to be good, but it has to be new. I think these two paths converge at the top: the best design surpasses its predecessors by using new ideas, and the best research solves problems that are not only new, but actually worth solving. So ultimately we're aiming for the same destination, just approaching it from different directions.

What I'm going to talk about today is what your target looks like from the back. What do you do differently when you treat programming languages as a design problem instead of a research topic?

The biggest difference is that you focus more on the user. Design begins by asking, who is this for and what do they need from it? A good architect, for example, does not begin by creating a design that he then imposes on the users, but by studying the intended users and figuring out what they need.

Notice I said "what they need," not "what they want." I don't mean to give the impression that working as a designer means working as a sort of short-order cook, making whatever the client tells you to. This varies from field to field in the arts, but I don't think there is any field in which the best work is done by the people who just make exactly what the customers tell them to.

The customer *is* always right in the sense that the measure of good design is how well it works for the user. If you make a novel that bores everyone, or a chair that's horribly uncomfortable to sit in, then you've done a bad job, period. It's no defense to say that the novel or the chair is designed according to the most advanced theoretical principles.

And yet, making what works for the user doesn't mean simply making what the user tells you to. Users don't know what all the choices are, and are often mistaken about what they really want.

The answer to the paradox, I think, is that you have to design for the user, but you have to design what the user needs, not simply what he says he wants. It's much like being a doctor. You can't just treat a patient's symptoms. When a patient tells you his symptoms, you have to figure out what's actually wrong with him, and treat that.

This focus on the user is a kind of axiom from which most of the practice of good design can be derived, and around which most design issues center.

If good design must do what the user needs, who is the user? When I say that design must be for users, I don't mean to imply that good design aims at some kind of lowest common denominator. You can pick any group of users you want. If you're designing a tool, for example, you can design it for anyone from beginners to experts, and what's good design for one group might be bad for another. The point is, you have to pick some group of users. I don't think you can even talk about good or bad design except with reference to some intended user.

You're most likely to get good design if the intended users include the designer himself. When you design something for a group that doesn't include you, it tends to be for people you consider to be less sophisticated than you, not more sophisticated.

That's a problem, because looking down on the user, however benevolently, seems inevitably to corrupt the designer. I suspect that very few housing projects in the US were designed by architects who expected to live in them. You can see the same thing in programming languages. C, Lisp, and Smalltalk were created for their own designers to use. Cobol, Ada, and Java, were created for other people to use.

If you think you're designing something for idiots, the odds are that you're not designing something good, even for idiots.

Even if you're designing something for the most sophisticated users, though, you're still designing for humans. It's different in research. In math you don't choose abstractions because they're easy for humans to understand; you choose whichever make the proof shorter. I think this is true for the sciences generally.

Scientific ideas are not meant to be ergonomic.

Over in the arts, things are very different. Design is all about people. The human body is a strange thing, but when you're designing a chair, that's what you're designing for, and there's no way around it. All the arts have to pander to the interests and limitations of humans. In painting, for example, all other things being equal a painting with people in it will be more interesting than one without. It is not merely an accident of history that the great paintings of the Renaissance are all full of people. If they hadn't been, painting as a medium wouldn't have the prestige that it does.

Like it or not, programming languages are also for people, and I suspect the human brain is just as lumpy and idiosyncratic as the human body. Some ideas are easy for people to grasp and some aren't. For example, we seem to have a very limited capacity for dealing with detail. It's this fact that makes programing languages a good idea in the first place; if we could handle the detail, we could just program in machine language.

Remember, too, that languages are not primarily a form for finished programs, but something that programs have to be developed in. Anyone in the arts could tell you that you might want different mediums for the two situations. Marble, for example, is a nice, durable medium for finished ideas, but a hopelessly inflexible one for developing new ideas.

A program, like a proof, is a pruned version of a tree that in the past has had false starts branching off all over it. So the test of a language is not simply how clean the finished program looks in it, but how clean the path to the finished program was. A design choice that gives you elegant finished programs may not give you an elegant design process. For example, I've written a few macro-defining macros full of nested backquotes that look now like little gems, but writing them took hours of the ugliest trial and error, and frankly, I'm still not entirely sure they're correct.

We often act as if the test of a language were how good finished programs look in it. It seems so convincing when you see the same program written in two languages, and one version is much shorter. When you approach the problem from the direction of the arts, you're less likely to depend on this sort of test. You don't want to end up with a programming language like marble.

For example, it is a huge win in developing software to have an interactive toplevel, what in Lisp is called a read-eval-print loop. And when you have one this has real effects on the design of the language. It would not work well for a language where you have to declare variables before using them, for example. When you're just typing expressions into the toplevel, you want to be able to set x to some value and then start doing things to x. You don't want to have to declare the type of x first. You may dispute either of the premises, but if a language has to have a toplevel to be convenient, and mandatory type declarations are incompatible with a toplevel, then no language that makes type declarations mandatory could be convenient to program in.

In practice, to get good design you have to get close, and stay close, to your users. You have to calibrate your ideas on actual users constantly, especially in the beginning. One of the reasons Jane Austen's novels are so good is that she read them out loud to her family. That's why she never sinks into self-indulgently arty descriptions of landscapes, or pretentious philosophizing. (The philosophy's there, but it's woven into the story instead of being pasted onto it like a label.) If you open an average "literary" novel and imagine reading it out loud to your friends as something you'd written, you'll feel all too keenly what an imposition that kind of thing is upon the reader.

In the software world, this idea is known as Worse is Better. Actually, there are several ideas mixed together in the concept of Worse is Better, which is why people are still arguing about whether worse is actually better or not. But one of the main ideas in that mix is that if you're building something new, you should get a prototype in front of users as soon as possible.

The alternative approach might be called the Hail Mary strategy. Instead of getting a prototype out quickly and gradually refining it, you try to create the complete, finished, product in one long touchdown pass. As far as I know, this is a recipe for disaster. Countless startups destroyed themselves this way during the Internet bubble. I've never heard of a case where it worked.

What people outside the software world may not realize is that Worse is Better is found throughout the arts. In drawing, for example, the idea was discovered during the Renaissance. Now almost every drawing teacher will tell you that the right way to get an accurate drawing is not to work your way slowly around the contour of an object, because errors will accumulate and you'll find at the end that the lines don't meet. Instead you should draw a few quick lines in roughly the right place, and then gradually refine this initial sketch.

In most fields, prototypes have traditionally been made out of different materials. Typefaces to be cut in metal were initially designed with a brush on paper. Statues to be cast in bronze were modelled in wax. Patterns to be embroidered on tapestries were drawn on paper with ink wash. Buildings to be constructed from stone were tested on a smaller scale in wood.

What made oil paint so exciting, when it first became popular in the fifteenth century, was that you could actually make the finished work *from* the prototype. You could make a preliminary drawing if you wanted to, but you weren't held to it; you could work out all the details, and even make major changes, as you finished the painting.

You can do this in software too. A prototype doesn't have to be just a model; you can refine it into the finished product. I think you should always do this when you can. It lets you take advantage of new insights you have along the way. But perhaps

even more important, it's good for morale.

Morale is key in design. I'm surprised people don't talk more about it. One of my first drawing teachers told me: if you're bored when you're drawing something, the drawing will look boring. For example, suppose you have to draw a building, and you decide to draw each brick individually. You can do this if you want, but if you get bored halfway through and start making the bricks mechanically instead of observing each one, the drawing will look worse than if you had merely suggested the bricks.

Building something by gradually refining a prototype is good for morale because it keeps you engaged. In software, my rule is: always have working code. If you're writing something that you'll be able to test in an hour, then you have the prospect of an immediate reward to motivate you. The same is true in the arts, and particularly in oil painting. Most painters start with a blurry sketch and gradually refine it. If you work this way, then in principle you never have to end the day with something that actually looks unfinished. Indeed, there is even a saying among painters: "A painting is never finished, you just stop working on it." This idea will be familiar to anyone who has worked on software.

Morale is another reason that it's hard to design something for an unsophisticated user. It's hard to stay interested in something you don't like yourself. To make something good, you have to be thinking, "wow, this is really great," not "what a piece of shit; those fools will love it."

Design means making things for humans. But it's not just the user who's human. The designer is human too.

Notice all this time I've been talking about "the designer." Design usually has to be under the control of a single person to be any good. And yet it seems to be possible for several people to collaborate on a research project. This seems to me one of the most interesting differences between research and design.

There have been famous instances of collaboration in the arts, but most of them seem to have been cases of molecular bonding rather than nuclear fusion. In an opera it's common for one person to write the libretto and another to write the music. And during the Renaissance, journeymen from northern Europe were often employed to do the landscapes in the backgrounds of Italian paintings. But these aren't true collaborations. They're more like examples of Robert Frost's "good fences make good neighbors." You can stick instances of good design together, but within each individual project, one person has to be in control.

I'm not saying that good design requires that one person think of everything. There's nothing more valuable than the advice of someone whose judgement you trust. But after the talking is

done, the decision about what to do has to rest with one person.

Why is it that research can be done by collaborators and design can't? This is an interesting question. I don't know the answer. Perhaps, if design and research converge, the best research is also good design, and in fact can't be done by collaborators. A lot of the most famous scientists seem to have worked alone. But I don't know enough to say whether there is a pattern here. It could be simply that many famous scientists worked when collaboration was less common.

Whatever the story is in the sciences, true collaboration seems to be vanishingly rare in the arts. Design by committee is a synonym for bad design. Why is that so? Is there some way to beat this limitation?

I'm inclined to think there isn't-- that good design requires a dictator. One reason is that good design has to be all of a piece. Design is not just for humans, but for individual humans. If a design represents an idea that fits in one person's head, then the idea will fit in the user's head too.

Related:

- Japanese Translation
- Taste for Makers
- Romanian Translation
- Spanish Translation
