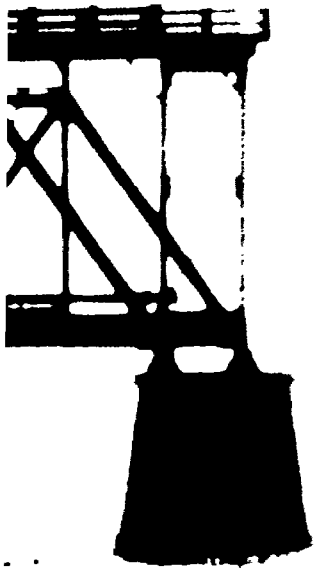
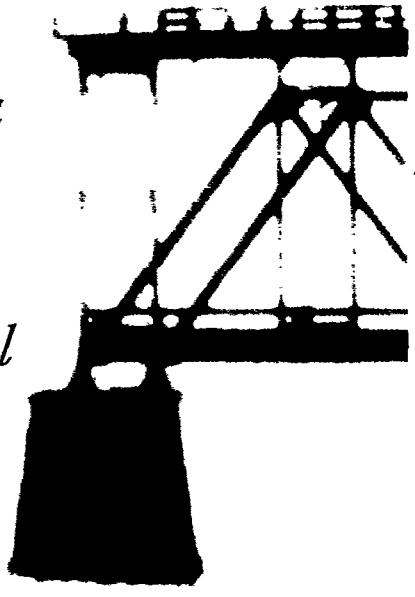
Gaps Marc Rettig

I've been talking to accomplished software designers, trying to learn how they do what they do. This is more difficult than I expected. Learning about design as it is practiced by working designers is hard because the practice is so diverse, and because so many excellent designers aren't writing or talking about what they do. Books and classes are one thing, but there is almost always a difference between doctrine and practice. When you ask designers, "how did you make that wonderful thing?" you may or may not be able to get a straight answer.





I've also been trying to find ways to communicate the value of design to business people. This is more difficult than I expected. So much of the necessary language is already in use, and so much of the necessary perspective is new to the audience.

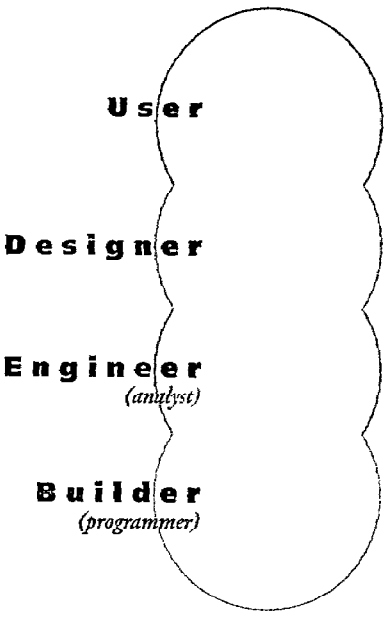
Figures 1 and 2 have helped me explain the role of design in a software project. Figure 1 shows the continuum of concerns between the concrete world of the product's intended user and the programmer's world of algorithms and data structures. The circles overlap, representing the common ground shared by people in each area of responsibility. Engineers (a.k.a. "systems analysts" in some corners of the corporate world) have to

* The image Bank/Grant V Laro

understand programming well enough to work with the implementors, and they have to understand the design well enough to specify the right features. It's the old, "Do the right thing, and do it right."

I'm interested in the "designer" circle, because excellence in design almost always comes from the consistently-applied vision of a small team who jealously defend the practice of whole-product, user-centered design. They aren't designing "applications," windows, or "GUI"s. They are concerned with the whole experience of the people who will be using the product, and with its whole context of use. Furthermore, they have a role in the organization that lets them influence all the necessary aspects of the design. They have to know enough about software engineering to avoid ridiculous specifications, and to work closely with others on the team, but their main objective is to create comfortable, functional worlds that satisfy the needs and intentions of the users. (These figures are based on Meredith Bricken's notions in "Virtual Worlds: No Interface to Design," in Michael Benedikt's *Cyberspace: First Steps.*)

Figure 1



The rise in concern about product usability has resulted in common practice that looks more like figure 2. Instead of creating a role for designers to practice their craft, we are expanding the engineer's world to include a new set of concerns. The engineers are learning to work with the users, and to shape their engineering designs according lessons learned in this dialog. This is a great step forward, but it places an unreasonable burden on the software engineer, whose home discipline already creates tremendous demands without the added skill requirements of user-centered design. Other design-centered fields, such as building architecture and industrial design, use a model more like figure 1.

I set out to learn about design so I could speak clearly about its value and practice. But mostly I have discovered gaps; gaps waiting for bridges, and lonely bridge-builders slinging rope walkways. We need to find ways to build sturdy bridges across these gaps. One, the credibility gap, is closing as the field learns enough about its work to make books like *Cost-Justifying Usability* and *The Trouble With Computers.* We are learning how to argue our case. But there are still wide gaps between the average software designer and open-armed acceptance into the process as peers with systems analysts, software engineers, and programmers. Mitch Kapor's 1990 *Software Design Manifesto* still rings true.

The language gap

When I talk about design with development managers at a large data processing shop, I immediately bump into a language problem. They already use the word "design" all the time. Software engineers design data structures, networks, object spaces, and systems architectures. Programmers design their classes, modules, and algorithms. So if someone walks in and starts talking about "design," there is an instant communication problem; the worse kind, since people think they are communicating (after all, they're using the same vocabulary), but really they aren't.

In other design-oriented fields, people say "engineer" where software developers say "design." Or they put adjectives in front: "product design," "systems design," "engineering design." I get some mileage in my company by talking about "Design with a capital D'" as a way of distinguishing user-centered whole-product design from all the other kinds of design.

The language gap makes it difficult to speak clearly about design.

The example gap

I've been looking for examples of good design, which will help provide a compelling argument for an emphasis on design in business systems. There are a few examples of well-designed shrink-wrapped software. But it's hard to point to even something like say, Intuit's *Quicken,* and tell a manager, "With the right attention to user-centered design, the

right process, and the right mix of skills on our team, we could build the *Quicken* of manufacturing inventory systems."

Most of the identifiably excellent work in software design is being done in retail software, most notably in entertainment. Internal business systems—customer service, inventory, manufacturing control, resource allocation, and so on and on and on—are typically built with little attention to user-centered product design. As Jakob Nielsen pointed out in *interactions,* I.2, p. 55 ("GUI Panic is Good Panic"), the usability movement is doing a lot to raise design awareness in corporations. And there are a few design firms and departments who consistently deliver outstanding work. Unfortunately it is very difficult to learn about the few resulting examples of excellence.

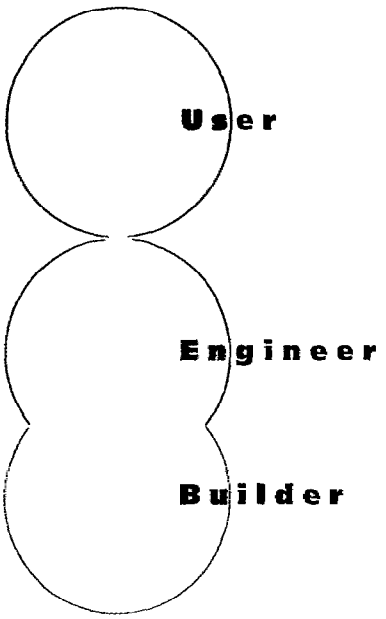
The example gap makes it difficult to demonstrate the value of design.

The point of view gap

This could also be called, "the clique problem." Over the last few years "the CHI community" (with 'CHI' meaning "computer-human interaction," not just ACM SIGCHI) has gained a lot of visibility in the business world. "Usability" is now a top priority on many development teams, and it's an important measure of quality in product reviews and corporate quality assessments. But relatively few of us are doing and teaching the kind of whole-product design that goes on in figure 1. Our literature covers wide territory in usability engineering: human factors, prototyping, testing, and iterative processes. We're also saying a lot about graphic design; so much so that some percentage of corporate developers still think that good usability comes from clear window layouts and beautiful bitmaps.

We have participatory design, we have books about integrating usability into the software development process, and we have books about the business case for paying attention to design. But so many are focused on these things that the field's overall message has an unhealthy imbalance towards our own point of view. We spend most of our time writing to each other and not enough time writing to the people who need our services. Worse, for all of our worry about technology-centered approaches in software development, we get caught up in the same trap ourselves. We focus on the technology of usability—the techniques, processes, tools, and gadgets—more than the craft: the gritty daily contact with other people, the basic design skills, and the curious artist-engineer-psychologist mental process that makes up the world of the practicing designer. There's something about putting this all into a "methodology" or a set of "guidelines" that takes the life out of design. What reprints do you give to development teams and their managers to explain whole-product user-centered design? Where do you go to hire people who can fill the "design" circle in figure 1? The point-of-view gap makes it hard for management to take designers seriously.

Figure 2



A brick in the gap: the *interactions* awards

This Spring, ACM and *interactions* magazine will publish the call for nominations for the first annual ACM interactions Design Awards, a peer-recognition program for excellence in interaction design. This represents an important effort to bridge some of the gaps in our field and an important point in the craft's growth to maturity.

The ACM interactions Design Award is modeled on awards sponsored by publishers in other design-oriented fields, such as Communication Arts' Design Annuals and the IDEA awards for industrial design. As a peer-recognition program, this award differs from other ACM awards in several ways:

- It is not a cash award, nor is it given in recognition of lifetime achievement or a dramatic influence on the industry.
- Winners receive a certificate, plaque or other token of recognition. Their work will be displayed in a special issue of interactions, and described in the award's

publicity material. Winning works may also appear in interactions expositions at conferences and museums.

- The process is designed to encourage growth in the field. Top-placing entries will receive feedback from the jury, discussing strong points and areas for improvement (similar to say, the Malcolm Baldridge Award for Quality).

The call for nominations will be issued this Spring, with a final deadline of September 30th. The winners will be showcased next Spring in a special issue of *interactions*. Critiquing and granting awards to interactive products is a dauntingly difficult undertaking, since the jury will have to take into account the intended context of use, constraints on the design, and the profile of the intended users. The committee will need help and support from the *interactions* readership and CHI community.

However difficult the job might be, I can't wait for the results.

Marc Rettig, 100 S. Wacker, Chicago, IL 60606; +1-312-507-9013; 76703.1037@compuserve.com

References

[1] Bias, Randolph G and Mayhew, Deborah J, *Cost-justifying Usability*, 1994, Academic Press.

[2] Kapor, Mitch, *The Software Design Manifesto*, http://www.kei.com/homepages/mkapor/

[3] Landauer, Thomas K. *The Trouble with Computers: Usefulness, Usability, and Productivity*, 1995, MIT Press.

[4] Meredith Bricken, "Virtual Worlds: No Interface to Design," in Benedikt, Michael, *Cyberspace, First Steps*, 1993, MIT Press, pp. 363-382.