Dealers in Lightning: Xerox PARC and the Dawning of the Computer Age, Michael Hiltzik

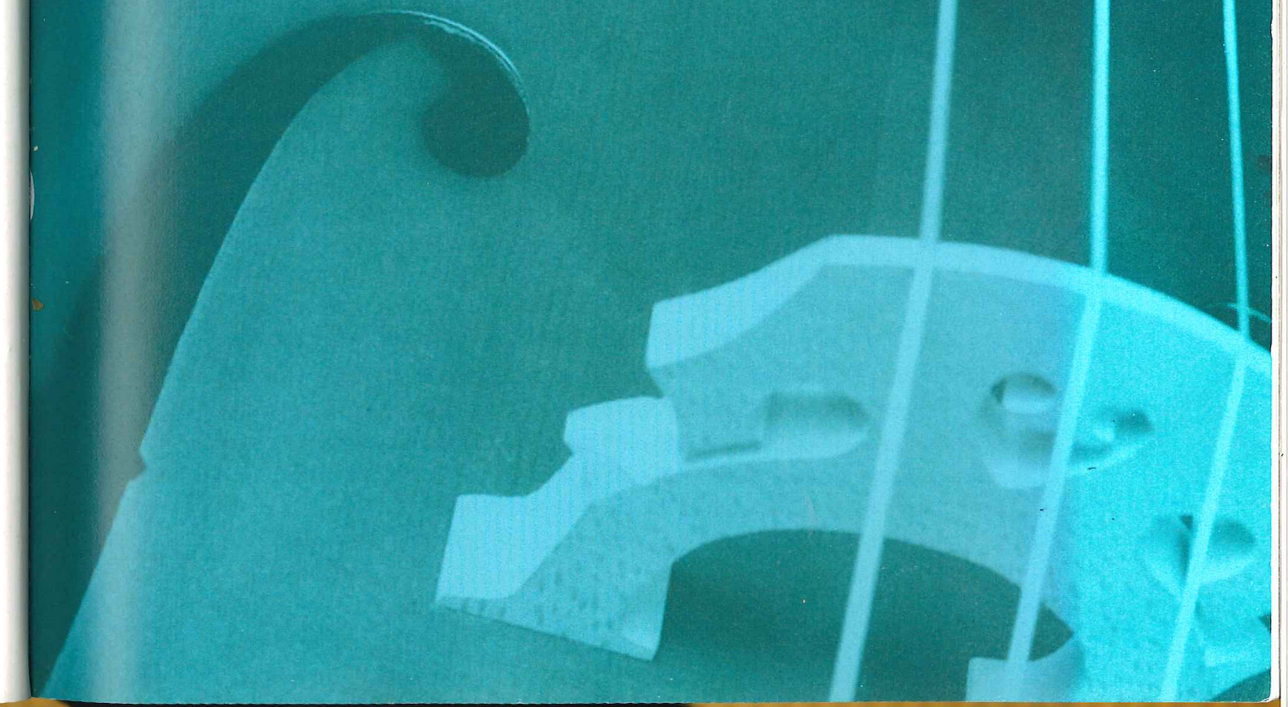
The Victorian Internet, Tom Standage

Geeks Bearing Gifts: How the Computer World Got This Way, Ted Nelson

The Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal, M. Mitchell Waldrop

2

The Four Approaches to Interaction Design



In the field of interaction design, not only are there different schools of thought about what interaction design is (see Chapter 1), but there are also different styles of working that should be examined.

There are four major approaches to tackling interaction design projects:

- User-centered design (UCD)
- Activity-centered design
- Systems design
- Genius design

All four have been used to create successful products, and it is typically up to designers to select (sometimes unconsciously) the way that works best for them. A few assertions apply to all these approaches:

- They can be used in many different situations to create vastly different products and services, from Web sites to consumer electronics to interactive environments.
- Most problematic situations can be improved by deploying at least one of these approaches to solving the problem.
- The best designers are those who can move between approaches, applying the best approach to the situation, sometimes applying multiple approaches even within a single project.
- An individual designer will probably gravitate toward one of these approaches more than others. Some of these approaches may feel simply wrong. Designers will probably generally work with the approaches they feel most comfortable employing. At different times, however, another approach may be the best way to solve a design problem, so it is important that interaction designers know all four approaches.

Table 2.1 provides a quick comparison of the four approaches.

We'll look in detail at each of these approaches, outlining in broad strokes the philosophy behind each one in its purest form (which is unlikely to be found in practice). We'll start with the approach that is currently the most popular: user-centered design.

TABLE 2.1 Four Approaches to Design

Approach	*Overview*	*Users*	*Designer*
User-centered design	Focus on user needs and goals	The guides of design	Translator of user needs and goals
Activity-centered design	Focus on the tasks and activities that need to be accomplished	Performers of the activities	Creates tools for actions
Systems design	Focus on the components of a system	Set the goals of the system	Makes sure all the parts of the system are in place
Genius design	Skill and wisdom of designers used to make products	Source of validation	The source of inspiration

User-Centered Design

The philosophy behind user-centered design is simply this: users know best. The people who will be using a product or service know what their needs, goals, and preferences are, and it is up to the designer to find out those things and design for them. One shouldn't design a service for selling coffee without first talking to coffee drinkers. Designers, however well-meaning, aren't the users. Designers are involved simply to facilitate the achievement of the users' goals. Participation from users is sought (ideally) at every stage of the design process. Indeed, some designers view users as co-creators.

The concept of user-centered design has been around for a long time; its roots are in industrial design and ergonomics and in the belief that designers should try to fit products to people instead of the other way around. Industrial designer Henry Dreyfuss, who designed the iconic 500 series telephone for Bell Telephones, first popularized the method with his 1955 book *Designing for People*. But while industrial designers remembered this legacy, software engineers were blissfully unaware of it, and for decades they churned out software that made sense in terms of the way computers work, but not in terms of the way that people work. To be fair, this focus was not all the engineers' fault; with the limited processing speed and memory of computers for the first 40 years of their existence, it's sometimes astounding that engineers

could make computers useful at all. The constraints of the system were huge. There was little concern for the user because it took so much effort and development time simply to get the computer to work correctly.

In the 1980s, designers and computer scientists working in the new field of human-computer interaction began questioning the practice of letting engineers design the interface for computer systems. Increased memory, processing speed, and color monitors now made, different types of interfaces possible, and a movement began to focus the design of computer software around users, not around computers. This movement became known as user-centered design (UCD).

Goals are really important in UCD; designers focus on what the user ultimately wants to accomplish. The designer then determines the tasks and means necessary to achieve those goals, but always with the users' needs and preferences in mind.

In the best (or at least most thorough) UCD approach, designers involve users in every stage of the project. Designers consult users (and potential users) at the beginning of the project to see if the proposed project will even address the users' needs. Designers conduct extensive research (see Chapter 4) up front to determine what the users' goals are in the current situation. Then, as designers begin ideation (see Chapter 6), users are brought in to help generate concepts (which is known as **participatory design**). Designers (often alongside usability professionals) evaluate and test prototypes with users as well (see Chapter 8).

Simply put, throughout the project, user data is the determining factor in making design decisions. When a question arises as to how something should be done, the users' wants and needs determine the response. For example, if during user research for an e-commerce Web site, users say they want the shopping cart in the upper-right corner of the page, when the shopping cart is ultimately positioned on the page, that's likely where the shopping cart will be.

User goals—the real target of UCD—are notoriously slippery and often hard to define, especially long-term goals. Or else they are so vague, it is nearly impossible to design for them. Let's say a designer is creating an application to help college students manage their schedules. What's the goal there? To help students do better in school? But why? So they can graduate?

What's the goal there? To get a good job? To become educated? User goals can quickly become like Russian dolls, with goals nested inside goals.

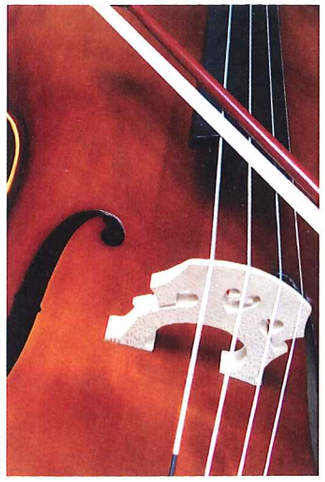
That being said, what UCD is best at is getting designers to move away from their own preferences and instead to focus on the needs and goals of the users, and this result should not be undervalued. Designers, like everyone else, carry around their own experiences and prejudices, and those can conflict with what users require in a product or service. A UCD approach removes designers from that trap. "You are not the user" is a mantra often espoused by UCD designers.

UCD doesn't always work, however. Relying on users for all design insights can sometimes result in a product or service that is too narrowly focused. Designers may, for instance, be basing their work on the wrong set or type of users. For products that will be used by millions of people, UCD may simply be impractical because the audience is too segmented. UCD is a valuable approach, but it is only one approach to designing.

Activity-Centered Design

Activity-centered design (ACD) doesn't focus on the goals and preferences of users, but instead on behavior surrounding particular tasks. Activities can be loosely defined as a cluster of actions and decisions that are done for a purpose. Activities can be brief and simple (making a sandwich) or time consuming and involved (learning a foreign language). Activities can take moments or years. You can do them alone or with others, as is the case, for example, when you sing a song. Some activities, such as withdrawing money from an ATM, have a set ending—in this case, getting the money. Others, such as listening to music, have no fixed ending. The activity simply stops when the actor (or some outside force) decides it is over.

ACD has its roots in activity theory, which is a psychological framework from the first half of the 20th century. Activity theory posits that people create tools as a result of "exteriorized" mental processes. Decision-making and interior life of individuals are de-emphasized in favor of *what people do* and the tools they collectively create in order to make (and to communicate). This philosophy translates well into activity-centered design, where the activity and the tools to support it—not the user—are at the center of the design process.



COURTESY ISTOCKPHOTO

Figure 2.1

A cello is a product that was definitely designed using activity-centered design. A UCD designer would have likely thought it far too hard to learn.

Many of the products we use today were designed using activity-centered design, especially functional tools like appliances and cars. Activity-centered design allows designers to tightly focus on the work at hand and create support for the activity itself, instead of for more distant goals (**Figure 2.1**). Thus, it's well-suited for complicated actions or for products with varied and large amounts of users.

The *purpose* of an activity is not necessarily a *goal*. Purposes are generally more focused and tangible than goals. Take the activity of raking leaves, for example. The gardener might have a goal (to have a tidy yard) but the purpose of raking is simple: to collect leaves. ACD would focus on collecting leaves.

Of course, sometimes goals and purposes can be the same or similar. For example, in the activity of making tea, the goal and the purpose are pretty much the same: to drink tea. Few people have as a goal to become a master tea brewer.

Activities are made up of actions and decisions, otherwise known as *tasks*. Tasks can be as discrete as pushing a button or as complicated as performing all the steps necessary to launch a nuclear missile. The purpose of tasks is to engage in (and possibly complete) an activity. Each task is a moment in the life of the activity, and many of those moments can be aided by design. For example, a button can be provided to turn a device on, and a label or instructions may aid a user in making a decision.

The difference between a task and an activity can be fairly minor. Some tasks have enough parts to them to be considered subactivities unto themselves. For example, in making a phone call, one of the tasks is finding the right number to dial. There are quite a few ways to find a phone number: call a service for assistance, look up the number in the phone or online, recall it from memory, and so on. Each of these solutions to the task of finding a number is itself a task. So is finding a phone number a task or an activity?

For designers, the difference is usually academic; it has to be designed for no matter what it's called.

Like user-centered design, activity-centered design relies on research as the basis for its insights, albeit differently. Designers observe and interview users for insights about their behavior more than about their goals and motivations. Designers catalog users' activities and tasks, perhaps add some missing tasks, and then design solutions to help users accomplish the task, not achieve a goal per se.

Ultimately, activity-centered design allows designers to focus narrowly on the tasks at hand and design products and tools that support those tasks. The task "submit form" will probably require a button. The task "turn device on" will probably require a switch or button. And so on. The activity, not necessarily the people doing the activity, guides the design.

Activity-centered design can be ethically tricky. Some tasks require skill—sometimes great skill—and designers shouldn't ignore this in designing alternatives. Removing or automating people's valuable skills is morally troubling. It may take weeks to learn a call-center's software, for instance, and removing the need for that skill can devalue employees. But then again, perhaps the reason the software takes weeks to learn is because it's poorly designed. Designers should be especially careful of the tasks they automate: it is very easy to de-skill users, to remove tasks that may be tedious or difficult to learn, but are also pleasurable to perform. Imagine being asked to design a piano that was easier to learn and play!

Another danger in activity-centered design is that by fixating on tasks, designers won't look for solutions for the problem as a whole. They won't see the forest for the trees. There's an old design adage: you'll get a different result if you set out to design a vase instead of something to hold flowers. By focusing on small tasks, designers can find themselves designing vase after vase and never a hanging garden.

Systems Design

Systems design is a very analytical way of approaching design problems; it uses an established arrangement of components to create design solutions. Whereas in user-centered design, the user is at the center of the design process, here a system—a set of entities that act upon each other—is. A system

isn't necessarily a computer, although it can be. Systems can also consist of people, devices, machines, and objects. Systems can range from the simple (the heating system in your house) to enormously complex (whole governments). Systems design is a structured, rigorous design approach that is excellent for tackling complex problems and offers a holistic approach to designing. Systems design doesn't discount user goals and needs—they can be used to set the goal of the system. But in this approach, users are de-emphasized in favor of *context*. Designers using system design focus on the whole context of use, not just individual objects or devices. Systems design can be thought of as a rigorous look at the broad context in which a product or service will be used.

Systems design outlines the components that systems should have: a goal, a sensor, a comparator, and an actuator. The job of the designer, then, becomes that of designing those components. In this way, systems design removes the guesswork and fuzziness of the other approaches and provides a clear roadmap for designers to follow.

Let's use the classic example of a heating system to illustrate the main parts of any system (**Figure 2.2**):

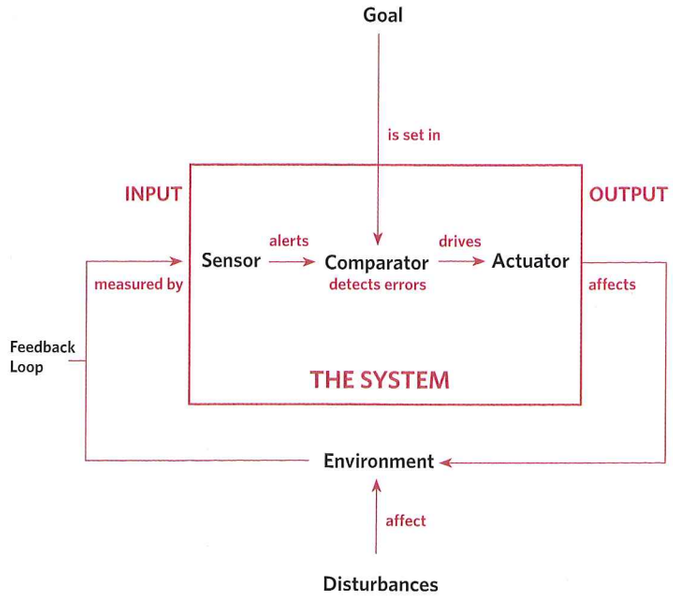
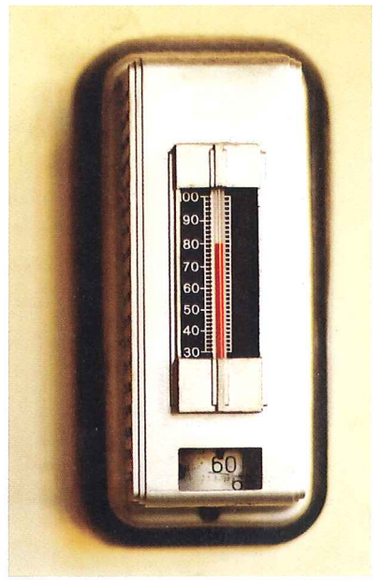


Figure 2.2

A system, based on a diagram by Hugh Dubberly and Paul Pangaro, 2003.

- **Goal.** This is not the users' goal, but rather the goal of the system as a whole, which can be drawn from user goals. The goal states the ideal relationship between the system and the environment it lives in. In a heating system, an example of a goal may be keeping your house at 72 degrees Fahrenheit.
- **Environment.** Where does the system "live"? Is it digital or analog or both? The environment in the heating system example is the house itself.
- **Sensors.** How does the system detect changes in the environment? A heating system has a thermostat with a thermometer (**Figure 2.3**) to detect temperature changes.
- **Disturbances.** Changes are called disturbances; these are elements in the environment that change the environment in both expected and unexpected ways. In the heating system example, a disturbance is a drop or rise in temperature.
- **Comparator.** The comparator embodies the goal within the system. It compares the current state (the environment) to the desired state (the goal). Any difference between the two is seen by the system as an error, which the system seeks to correct. In the heating system example, the comparator can be a tiny computer or a mercury switch that compares what the sensor tells it about the environment (for example, "72 degrees…72 degrees…72 degrees…71 degrees…71 degrees") to the goal ("Keep the house at 72 degrees").



COURTESY ISTOCKPHOTO

Figure 2.3

A thermostat contains the sensor, comparator, actuator, and controls of a heating system.

- **Actuator.** If the comparator says, ah, something is different (an "error") by examining the data coming from the sensor, it sends a command to the actuator (in this case, the boiler). Actuators are a means of making changes (output) to the environment. In this case, the actuator makes heat.
- **Feedback.** With output comes feedback. Feedback is a message about whether or not a goal was achieved or maintained—whether or not an error was detected. In this example, feedback would report either that the house is still at 71 degrees or that is now at 72 degrees and the heater can be turned off.

- **Controls.** Controls are means of manually manipulating the parts of the system (except the environment). In this example, you use a control to set the temperature you want the house to be. Another control might trigger the actuator and turn the heat on.

There are two types of disturbances to the environment that might affect our heating system. The first consists of expected disturbances, such as the temperature dropping. The second type consists of unexpected disturbances—things that would fall outside of the expected range of input. These types of disturbances typically cause the system to crash or behave in odd ways. In our heating example, such an event might be a sudden 100-degree drop in temperature.

To make most unexpected disturbances expected (and thus make the system more stable), systems need what's called **requisite variety**. The system needs an assortment of responses to deal with a range of situations. These responses can be anything from error messages ("You are being sent 1 million e-mail messages!") to workarounds ("You are being sent 1 million e-mail messages. Should I delete them or deliver them in chunks of 10,000?") to mechanisms to prevent the system from failing (deleting all incoming e-mails over a certain number). Systems without requisite variety crash often, which may be fine for a prototype, but not so great for, say, an air-traffic control system.

Feedback is output from the system that reports that something has just happened: input was received from the environment, the comparator was changed, and so on. You get feedback from your computer almost every time you press a key. We'll discuss feedback for users more in Chapter 7, but we'll simply note here that systems without feedback either will not work or will be bewildering.

Systems design isn't only about digital products, of course. Most services, for example, are systems consisting of digital and analog components. Your local coffee shop is filled with sensors, comparators, and actuators, only you probably know them as the shop employees. However, the objections and distaste many designers have about systems design spring from examples just such as these. Many designers feel that systems design is dehumanizing, turning people into robotic components in a very synthetic arrangement. And indeed, systems design is a very logical, analytical approach to interaction design. Emotions, passion, and whim have very little place in this sort of design, except as disturbances in the environment that need to be countered. Someone screaming angrily in the coffee shop is a major disturbance!

The greatest strength of systems design, however, is that it is useful for seeing the big picture—for its holistic view of a project. No product exists in a vacuum, after all, and systems design forces designers to take into account the environment that the product or service inhabits. By focusing on the broad context of use and the interplay of the components, designers gain a better understanding of the circumstances surrounding a product or service.

Hugh Dubberly on Systems Design

Hugh Dubberly is founder and principal at Dubberly Design Office (DDO), an interaction design consultancy in San Francisco. Before forming DDO, he served as vice president for design at AOL/Netscape and as creative director at Apple Computer, Inc. He has also taught at San Jose State University and Stanford University.

What is systems design?

Systems design is simply the design of systems. It implies a systematic and rigorous approach to design—an approach demanded by the scale and complexity of many systems problems.

Where did systems design come from?

Systems design first appeared shortly before World War II as engineers grappled with complex communications and control problems. They formalized their work in the new disciplines of information theory, operations research, and cybernetics. In the 1960s, members of the design methods movement (especially Horst Rittel and others at Ulm and Berkeley) transferred this knowledge to the design world. Systems design continues to flourish at schools interested in design planning and within the world of computer science. Among its most important legacies is a research field known as design rationale, which concerns systems for making and documenting design decisions.

What can designers learn from systems design?

Today, ideas from design methods and systems design may be more relevant to designers than ever before—as more and more designers collaborate on designing software and complex information spaces. Frameworks suggested by systems design are especially useful in modeling interaction and conversation. They are also useful in modeling the design process itself.

Hugh Dubberly on Systems Design *(continued)*

What is the most important thing to be aware of in systems design?

A systems approach to design asks:

- For this situation, what is the system?
- What is the environment?
- What goal does the system have in relation to its environment?
- What is the feedback loop by which the system corrects its actions?
- How does the system measure whether it has achieved its goal?
- Who defines the system, environment, goal, etc.—and monitors it?
- What resources does the system have for maintaining the relationship it desires?
- Are its resources sufficient to meet its purpose?

Is systems design incompatible with user-centered design?

A systems approach to design is entirely compatible with a user-centered approach. Indeed, the core of both approaches is understanding user goals. A systems approach looks at users in relation to a context and in terms of their interaction with devices, with each other, and with themselves.

What is the relationship between systems design and cybernetics?

Cybernetics (the science of feedback) provides an approach to systems and a set of frameworks and tools. Among the most important ideas for designers:

- Definition of a system depends on point of view (subjectivity).
- We are responsible for our actions (ethical stance).
- All interaction is a form of conversation.
- All conversation involves goals, understandings, and agreements.

Are there times when systems design isn't appropriate?

A systems approach to design is most appropriate for projects involving large systems or systems of systems. Such projects typically involve many people, from many disciplines, working together over an extended period of time. They need tools to cope with their project's complexity: to define goals, facilitate communications, and manage processes. Solo designers working on small projects may find the same tools a bit cumbersome for their needs.

Genius Design

The fourth major design approach is something I named **genius design**. Genius design relies almost solely on the wisdom and experience of the designer to make design decisions. Designers use their best judgment as to what users want and then design the product based on that judgment. User involvement, if it occurs at all, comes at the end of the process, when users test what the designers have made to make sure it really works as the designer has predicted.

Compared to the rigor of the other three approaches, genius design seems almost cavalier. Yet this is how most interaction design is done today, either by choice—Apple, supposedly for privacy reasons, does very little user research or testing at all—or by necessity. Many designers work in organizations that don't provide funding or time for research, so the designers are left to their own devices.

This is not to say that designers who practice genius design don't consider users—they do. It's simply that the designers either don't have the resources or the inclination to involve users in the design process. Designers use their personal knowledge (and frequently the knowledge of the organization they're working for and research from others) to determine users' wants, needs, and expectations.

Genius design can create some impressive designs, such as Apple's iPod (**Figure 2.4**).

It can also create some impressive failures, such as Apple's first handheld device, the Newton, which was too bulky and difficult to use for most users. Aside from market forces (not an inconsiderable factor), the success of genius-designed products rests heavily on the skill of the designer. Thus, genius design is probably best practiced by experienced designers, who have encountered many different types of problems and can draw upon solutions from many past projects.

COURTESY APPLE

Figure 2.4
Apple's iPod was created using genius design by designers such as Jonathan Ive.

It probably also works best when the designer is one of the potential users, although this can be a serious trap as well. The designers who created the Windows 95 operating system probably considered themselves the users, but while they understood perfectly well how the OS worked, ordinary users suffered. Because of their intimate understanding of how the product or service they designed was created and their inside knowledge of the decisions behind it, the designers will know much more about the functionality of the product or service than will most end users.

Unfortunately, while genius design is best practiced by experienced designers, it's often attempted by inexperienced designers. Many designers use this approach because it is, frankly, easier than the other three. It requires a lot less effort to noodle on a whiteboard than it does to research users or artfully assemble the components of a system. Designers, especially new designers, should practice genius design with care, for instincts can be dead wrong.

Genius design has many strengths, however, especially for an experienced designer. It's a fast and personal way to work, and the final design, perhaps more than with the other approaches, reflects the designer's own sensibilities. It is also the most flexible approach, allowing designers to focus their efforts where they see fit. By following their own muses, designers may be able to think more broadly and innovate more freely.

Jim Leftwich on Rapid Expert Design

James Leftwich, IDSA, is Chief Experience Officer at SeeqPod, and founder of Orbit Interaction, a pioneering interaction design consultancy located in Palo Alto, California. He has over 25 years of broad consulting and professional experience in the area of Human Interface development and related intellectual property strategies.

You don't like the term "genius design" very much. How come?

I find it difficult to believe that [with that name] it's an approach that many people would aspire to. Young designers may simply think, "Well, hey, I'm not a genius, so I guess this approach isn't for me." Second, designers of any type are likely to cringe at the term, and would rather die before announcing to the world that they practice "genius design."

Jim Leftwich on Rapid Expert Design *(continued)*

I refer to it as Rapid Expert Design, rather than "genius" (even if some talent and inherent capabilities may be a definite plus just like in sports).

Given that I believe Rapid Expert Design is a valid and largely missing and under-examined approach to interactive product development (as well as re-development, improvement, turnarounds, etc.), I think it's important how the "framing" is carried out. I'm certain you're familiar with the concept of framing [see Chapter 3], and the importance it plays in all forms of communication and debate. It's very similar to how political issues are framed, and how when one partisan side (or anyone, for that matter) defines an issue in a particular way, it affects all the subsequent discussion and perception.

That's why I find the term you used very unhelpful in propagating an incredibly needed and valid approach.

Why is this approach so important?

We're slogging through a world with a nearly uncountable number of undesigned and unaddressed user interface and functional problems and inadequacies. Long, drawn-out, and process-oriented approaches to the development of interactive products and services have left us with precious few clear wins. Very few interactive products and designs stem from singular or whole visions and architectural guidance. They are, instead, the result of groupthink, rearview mirror timid incrementalism, bureaucratic and disempowered designer-watering-down, and a whole host of other threats and obstacles.

How can we teach designers to practice Rapid Expert Design?

This is where the catch is, and why it's so important to start with an acknowledgement of the validity of the approach and realization of how Rapid Expert Designers are trained and exercised. The only way to become proficient at the R.E.D. approach is through apprenticing and incrementally using the approach on projects of increasing scale and complexity. A young designer that ambitiously bites off an entire consumer product may indeed fail. However, it's important that they begin learning (along with more experienced designers) how to approach things in this manner, in smaller steps, so that they can eventually become more proficient at R.E.D.

Summary

Most designers feel more comfortable with one design approach than the others, although most designers mix approaches as they work. A designer's temperament, personal philosophy, and view of work and of a project's users will help determine which approach the designer prefers. But the best designers are those who can move between these different approaches as the situation warrants, so it's good to know them all.

For Further Reading

About Face 3: The Essentials of Interaction Design, Alan Cooper, Robert Reimann, and David Cronin

Designing for People, Henry Dreyfuss

Cybernetics, Second Edition: or the Control and Communication in the Animal and the Machine, Norbert Wiener

General System Theory: Foundations, Development, Applications, Ludwig Von Bertalanffy

Activity-Centered Design: An Ecological Approach to Designing Smart Tools and Usable Systems, Geri Gay and Helene Hembrooke

Acting with Technology: Activity Theory and Interaction Design, Victor Kaptelinin and Bonnie Nardi

3

Design Strategy