



Summit '98

Guerilla Design Techniques

by David Adams and Dan Beckett

Guerilla Design Techniques

Optimize Yourself

Computers get faster every month, but we don't. The smartest "optimization" for most of us is to use our time more effectively, not rewrite some code to run faster. If you want your code to run faster buy a faster machine. In this session we present three "guerrilla design techniques" that:

- ✓ Are cheap and easy to use.
- ✓ Optimize the design process.
- ✓ Improve communication with clients and users, and within a design group.
- ✓ Simplify trying several approaches, which improves the final result.
- ✓ Support ongoing system improvement.
- ✓ Can be fun (you don't have to admit this to anyone else.)

The three techniques we discuss are:

Sketching a System

Quickly give order to a new system. The system sketching process rapidly identifies possible solutions, and areas you (or the user) do not understand well enough to implement as a system.

Designing Screens On Paper

Optimize researching, documenting, designing, prototyping, and implementing screens.

Prototyping and User Testing

Early user testing and prototyping can make the difference between a system that is delivered on time and within budget, and a system that is late and too expensive.

Before we discuss the individual techniques let's review the overall purpose of these techniques: optimizing the design process.

Optimizing The Design Process

Catch Defects Early

Over the last thirty years numerous studies have shown that the earlier in the development process you find an error the cheaper it is to correct. This is common sense. As Benjamin Franklin said, "a stitch in time saves nine." If you find an error during analysis it costs almost nothing to correct because there is no code to modify, no screens to rebuild, no data to correct, no users to retrain, and little or no documentation to update. If you find an error during the design phase you have to update your models, but that's not too costly. If you find a problem in the implementation phase it can be expensive to correct. The problem may involve changing the underlying data model, and may affect other program areas. The rip-

ple effect can devastate a schedule and a budget. If the error is not found until the system is tested or installed then even small corrections can be prohibitively expensive.

Think of a software project from start to finish as a piece of clay that slowly hardens. Early in the process the clay is soft and easily shaped and reshaped. By the time the system is installed it is rigid. Changes may mean working with a chisel and breaking the piece. Here is a table that expresses how the cost of implementing changes increases progressively as the development cycle advances:

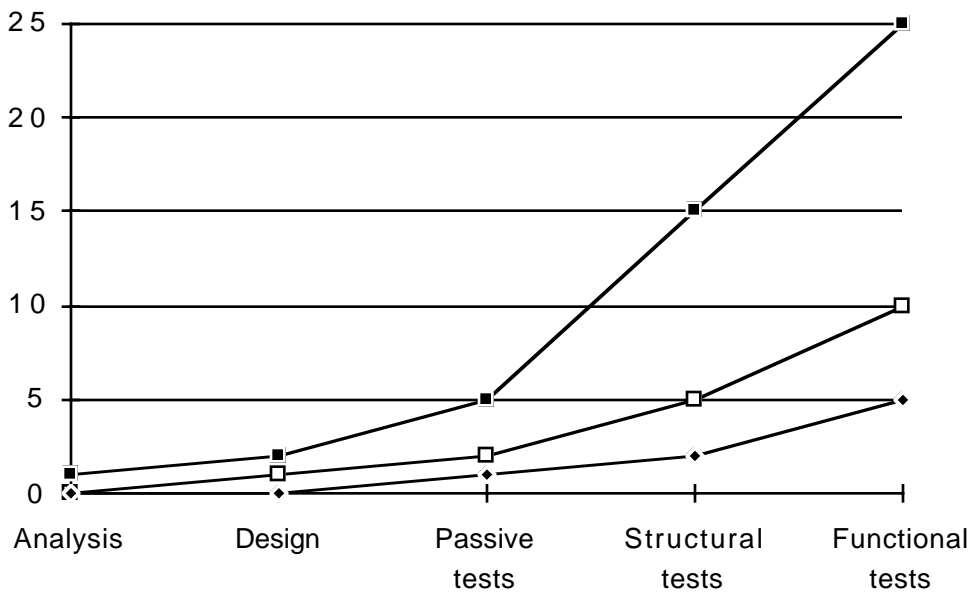
Time Detected	Requirements	Design	Code
Analysis	1	0	0
Design	2	1	0
Passive tests	5	2	1
Structural tests	15	5	2
Functional tests	25	10	5

Adapted from McConnell 1993, citing Dunn 1984.

You read this table like this:

A requirements defect found in the Analysis phase costs a unit of 1 to correct. The same defect found in the Design phase costs twice as much to correct, and 25 times as much to correct when in final testing.

The key point to observe from this table is that the increase in cost rises *sharply* as the phases progress. The *same* problem is increasingly expensive to correct the later into the project you find it. A \$100 defect caught in the requirements phase costs \$2,500 to correct if found during final testing. Here is a graphical presentation of the same information:



Costs increase sharply as the project progresses.

Ideally your analysis and design are so perfect that you don't have to make any corrections in the program later on. Well, that's nice in theory, but good luck achieving that in the real world. The point of using techniques to identify errors early is to reduce the number of late-phase errors you have to deal with. If you can reduce late-phase errors by 10% you've made a big difference in the cost and delivery date of the final project. If you use the techniques described in this session you should be able to achieve better than a 10% improvement.

Iteration Is Good

Design, and programming, are *by nature* iterative. It is very unlikely you'll get everything right on the first try. This is normal, natural, and never goes away. Iteration is arguably the most important benefit of quick and dirty design tools. The more designs you try out in a short time, the better. The more fluid your design tools, the easier to experiment with. The technique we're describing in this session make experimenting *easy*.

Improving The Conversation

People are smart. The more ideas that you can get out of the people on your design team, the better. When we say "design team" we mean you, any other designers/programmers you work with, *and* your clients or users. Your users are part of your team. They are the experts in their requirements and current procedures. (They're also the ones who ultimately decide if you've done a good job.) Ideally everyone on the team should contribute their knowledge and ideas to the conversation. It is all too easy to do team design with one person working on the

computer. This reduces the quality of the conversation. The person with the mouse is ultimately in charge, and the conversation has to stop while they create and position objects. Our guerilla design techniques are deliberately non-technical and minimally-computerized to make it possible for everyone to participate.

Parallel Development

Sometimes the best way for a group to work is alone. If that sounds like a contradiction it's because it's a contradiction; a contradiction we can all understand. In many cases group work doesn't draw out the best in all participants. Some people are shy, some people tend to dominate, some people think quickly on their feet, others need time to reflect. Our guerilla design techniques can be used in a group, or alone. The ideal approach can be to have each group member use the techniques alone and then share their work at a group session. The end result can integrate several people's ideas into a new solution that is better than any individual's original concept.

Now let's look at the techniques.

Sketching a System

sketch *n.* 1. A hasty or undetailed drawing or painting often made as a preliminary study. 2. A brief, general account or presentation; an outline.

The American Heritage Dictionary of the English Language

What Comes First?

When you're researching a new system it's hard to know where to start. If you're building a business system you need to find the things and activities the system uses and produces. If you're working on a database design you need a list of candidate fields and tables. If you're designing a Web site you need to figure out what the site includes. Building the first raw lists of items and events for a system is easy; you can review existing forms, reports, and literature, conduct interview, and hold brainstorming sessions. You're more likely to be overwhelmed by the volume of detail than anything else. What is difficult is figuring out the *relationships* amongst all of these things. Once you have a sense of what's involved in a system you end up facing questions like these:

Are these two things the same?

Should this thing be broken down into more detail?

Are these two things related somehow? Is one higher or lower in some kind of hierarchy?

What comes first?

As system designers and builders we need cheap and easy to use methods that improve analysis, design, and communication. In this chapter we describe a way of "sketching" systems that meets these goals. You and your users should be able to master this technique in about a minute.

An Example

You could be sketching any system for any purpose: data flow diagrams, a data model, a feature list, candidate field lists, class hierarchies, linked Web pages, whatever you need. We'll illustrate system sketching with an example Web site for a small company that sells coffee and coffee paraphernalia wholesale and retail. The company wants a Web site to increase brand awareness and sales. After looking at product brochures and interviewing management we've come up with a preliminary list of the items and features the Web site should include:

Annual report	Price list	Testimonials
Statement of purpose	Order form	Welcome message
Contact information	Feedback page	News
Coffee making instructions	Recipes that use coffee	Specials
Quotes about coffee	Address	Information about roasting
Product descriptions	Company logo	Coffee paraphernalia
Company history	Retail order form	Employee pictures
Customer service	Copyright information	Seasonal blends
T-Shirts	Mission statement	Coffee throughout history
Search for variety and roast	Hours	Contacting Webmaster

This list is shorter than you'll build for even the most basic systems, yet it already raises numerous questions. Some terms look redundant, some terms are unclear, and the relationships among terms is missing. Refining and organizing the list is already necessary with thirty items, and becomes more critical as the features and contents of the system expand. Here's how to quickly try out any number of "sketches" of how the system should be organized:

- 1) Start with a stack of blank index cards.
- 2) Write every key term you are trying to organize on its own card.
- 3) Move the cards around into groups until you've found an order that makes sense.
- 4) Write new cards, or use sticky pads, for the terms that describe groups of cards. (Index cards of a different color are convenient for category terms.)
- 5) If you find redundant terms, consolidate them.
- 6) If you find terms that need splitting, break them apart onto new cards.
- 7) If you find a term that needs renaming, rename it.
- 8) If some things don't make sense put them aside and ask the user.
- 9) Repeat as needed.

That's it. You can use this technique alone, in your design team, or with the user. This technique is not a full design methodology in itself, it is a tool for accelerating analysis and research in support of whatever design methodology you use.

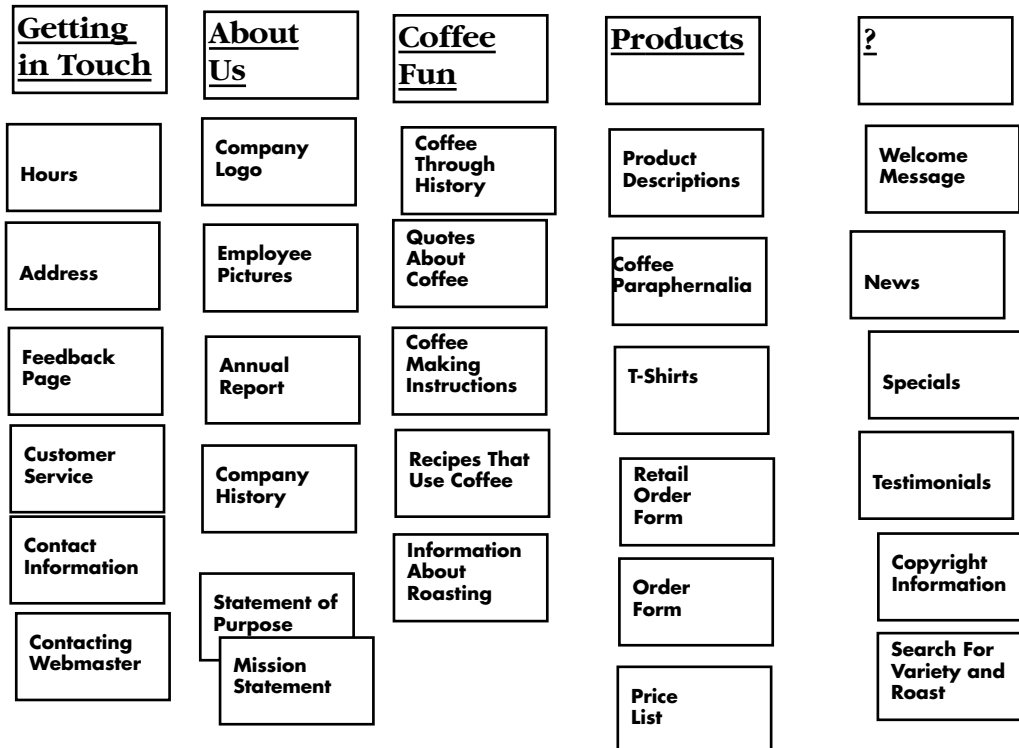
That's Too Easy!

The simplicity of this technique can lead people to dismiss it. Don't. This technique was invented by anthropologists, but is now used in virtually any discipline you can name. Different groups use system sketches in slightly different ways, and with different names (affinity diagramming, PICTIVE), but the technique itself remains nearly unchanged. If your boss needs a fancier name before agreeing to use this technique tell them its a CASE (Card Aided System Engineering) tool called STOO-

ICTST (Sorting Things Out On Index Cards To Save Time). If you don't like this acronym, feel free to make up your own. Now let's look at applying this technique to our coffee company Web site.

Example

Here is one way of organizing the cards listed earlier. Category terms are underlined and placed at the top.



Building this sketch took a few minutes. This is a helpful exercise as it reveals redundancies and questions. Here are a few of the questions that came up while building this sketch:

Aren't "Statement of Purpose" and "Mission Statement" the same thing?

A bunch of stuff doesn't fit in neatly. Where does "news" belong? What about "specials"? Are we missing a category?

Are "testimonials" about the company or the coffee? Both? Where do they go? "Products" or "About us"?

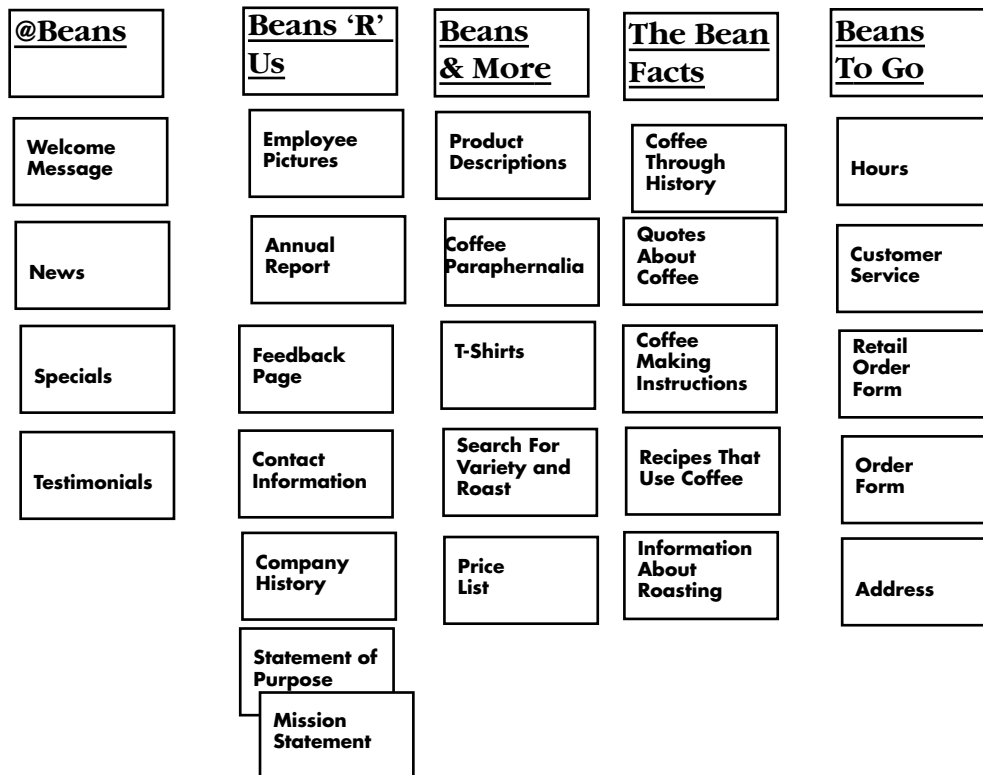
Where does the "Search for Variety and Roast" belong? Do we need other kinds of searches?

How do "Retail Order Form" and "Order Form" differ? How do they fit in with the "Price List"?

The point is that sketching allows you or your design team to:

- ✓ Quickly categorize items.
- ✓ Identify possible redundancies, ambiguities, and questions.
- ✓ Experiment with alternative approaches almost instantly.

This simple exercise leaves you well prepared to ask the user the next round of questions. If you want to get the greatest benefit out of this technique try several designs alone or in a group. As an example, here's another approach to the system we've been talking about:



Include these on every page:



This second designer's system sketch is different from the first in several ways. Notice that basic assumptions differ about what the sketch is meant to show. This version sketches out information categories, and a rough strategy for how to present the information as a Web site. A few common page elements are identified, the "@Bean" section groups topics that the company wants to emphasize, there is more humor, and product ordering is made into its own group to increase its visibility. This is a much more user and task oriented sketch. The sketch raises and

seeks to answer some of the same questions as the first sketch, misses some of the questions raised by the first sketch, and adds a few new ones. Here are some new questions:

What elements belong on each page?

What information does the company want to highlight?

What is the flow through the system like for a user? Where do they start?

Each design you produce leads to new questions and a better understanding of the system as a whole.

Working With Users

We've talked about using system sketching as a tool for you or a design team. You should also use this technique with users. After all, they're the experts on their business and data. People are *naturally* good at organizing items into hierarchies, and need little instruction. They do not need to know *anything* about computers to work with the cards.

Level Of Design

When you're building system sketches remember your current *level of design*. The sketches you've seen so far, particularly the second one, are beginning to propose system solutions when you do not even know the full extent of the system. A fundamental goal of system design is to determine the full scope of the system, and all of its meaningful details, *before* proposing a solution. This distinction can be difficult for users who are so deeply immersed in the details that they can see nothing else. System sketches like these can help you to elicit good information from users, but do not misuse them by trying to prototype systems too quickly. The card system allows you to experiment with different levels of design for *purposes of communication*. This technique in no way changes the importance of designing and specifying a system completely before choosing an implementation.

Benefits Summary

As a recap, let's review the benefits of the system sketching technique we've just discussed:

- ✓ It is almost free.
- ✓ It is flexible. Adding and removing terms involves almost no work.
- ✓ You can use this at several different stages of the design process.
- ✓ This technique is an analysis and interview tool not a design methodology. This tool is compatible with any design methodology.
- ✓ This technique can help identify underlying sources of confusion in the client's existing policies and procedures.
- ✓ Iteration improves design. You can try several approaches in a short time.
- ✓ There is no barrier to freely creating new cards, or consolidating existing ones.

- ✓ Communication improves. Unlike a traditional computer based approach several people can work on this system simultaneously.
- ✓ The user does not need to learn a new language or notation.
- ✓ You can train nearly anyone in this technique in about a minute.
- ✓ You can write definitions of terms on the backs of the cards for integration in your system design documentation.
- ✓ It never crashes ;-)

Designing Screens On Paper

Overview

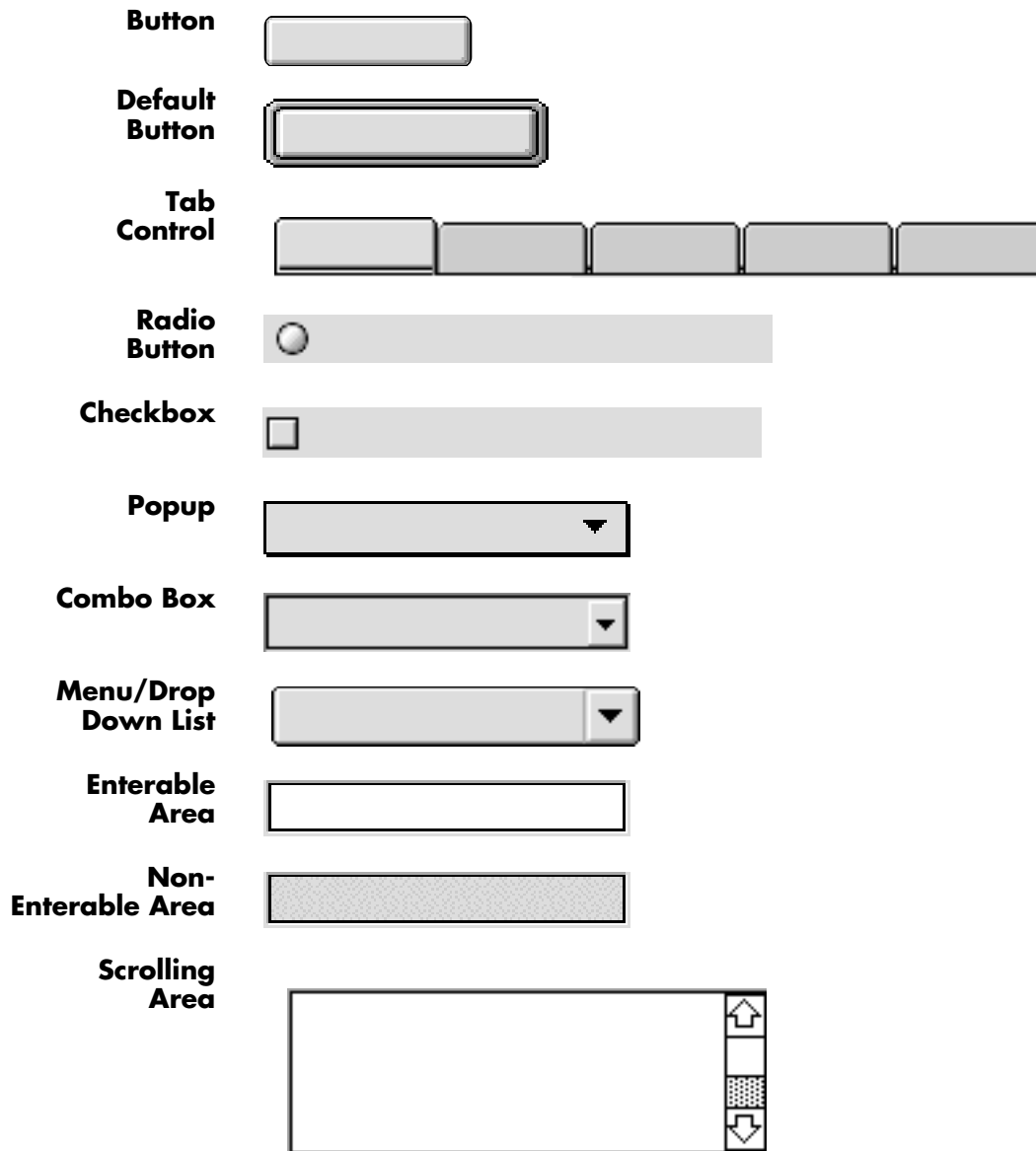
The simplest form of paper screen design is a hand drawn sketch. Many designers and design teams sketch proposed screens on a piece of paper or white board. This is a good way to quickly get some rough ideas about how a screen should look and work. Sketches work particularly well when you have someone on staff who is a quick and skillful sketch artist. The paper design technique we use goes a step beyond hand sketching. It takes a little more time to set up, but gives you paper screen designs suitable for review with end users. Here are the steps:

- 1) Print pictures of key screen design elements, like buttons, scrollable areas, and so on.
- 2) Cut the pictures up so that you have several copies of each object.
- 3) Get a blank piece of paper.
- 4) As you design the screen move interface objects around experimenting with different designs until you are satisfied.
- 5) Glue the objects into position.
- 6) Make whatever notes you need directly on the page.
- 7) When you're happy with the design, glue everything in place.
- 8) Make copies as needed.

That's it! This technique sounds trivial until you try it. In our case studies we've found that in 5-15 minutes we can build several versions of a screen design in a group. It typically takes 5-10 times longer to implement a prototype of the screen in 4D. By prototype we mean a screen that displays the basic objects and behavior, without performing all necessary data accesses and updates. Not only does this technique accelerate the screen design process, it is fun! (Remember: you don't need to tell anyone that it is fun. You can wear a lab coat and carry a clipboard if you need to protect your image.) Let's look at the tools of this technique more closely, and then go into the advantages of this technique in depth.

Tools Of The Trade: Object Templates

The hardest part of paper screen building is creating the sample objects and cutting them up. (If you have some kids around the house, put them to work!) You can build sheets of templates in about an hour. Here are some sample objects:



How Specific Should The Objects Be?

There are a few ways to approach paper screen design. You can use general or highly specific objects. For example, you can use sample objects as displayed on a particular OS, or use one set as a general tool. You can distinguish between similar objects such as Popups, Combo Boxes and Menu/Drop Down Lists, or use one object to represent any of these during design. What's the right approach? It depends on what you're trying to do. If you are trying to sketch out the basic behavior of a screen then the differences between display on different OS's is not

relevant. If, on the other hand, you are trying to determine how to present the same screen on both platforms with the least amount of change then you'll want to select very specific objects as displayed on a specific platform.

Example: A Simple Problem...

To give you a sense of how this technique works we'll look at an example. Here are the user's requirements for the screen:

We need a customer lookup screen that lets us find customers by first name, last name, or company name quickly and easily.

This sounds easy enough, and is typical of how users express their requirements. What should the screen look like? When you sit down to build it the "simple" description proves more complex.

Should the user be able to search on multiple fields?

What kinds of search comparisons are supported?

If multiple conditions are allowed, how can they be combined?

What kind of interface will make the search options easiest for the user to understand and work with?

Asking the user these questions is helpful, but often not enough. They know the result they want, but may not be able to readily visualize different interface options. As designers we should present them with choices that are easy to work with. You can build a number of test screens yourself, or in a group, to work out a few high quality proposals. Let's look at a few different versions of two different approaches to the "simple" search screen: the first allows only a single field search, the second allows multiple search fields. It took us about ten minutes to build the five paper designs we discuss.

A Single Field Search Interface

Here is the first effort at a single field interface.

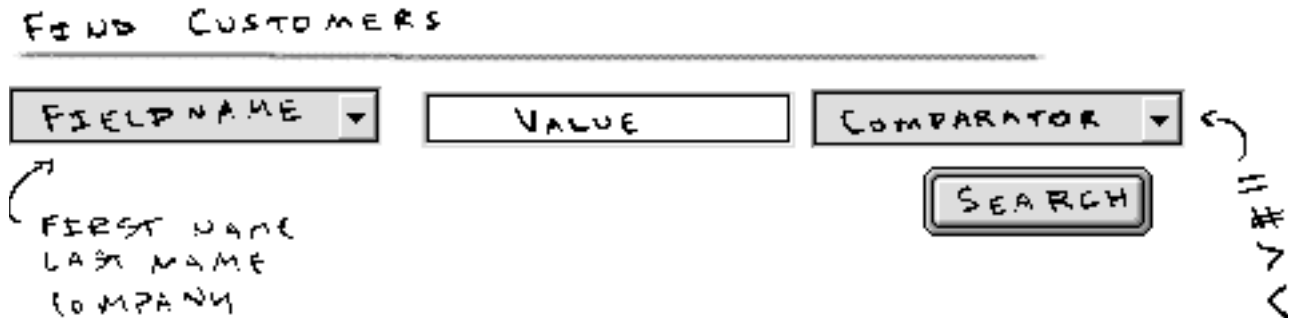
FIND CUSTOMERS

FIELDNAME ▾ VALUE SEARCH

↖
FIRST NAME
LAST NAME
COMPANY

The user can search on one field at a time.

This design lets the user select one search field and enter a value. This design appears easy to use, and takes very little space. It certainly satisfies the client's request for a "quick and easy" search. Unfortunately, the user cannot specify the kind of search comparison to use. The second design gives the user more control over how the search is performed:

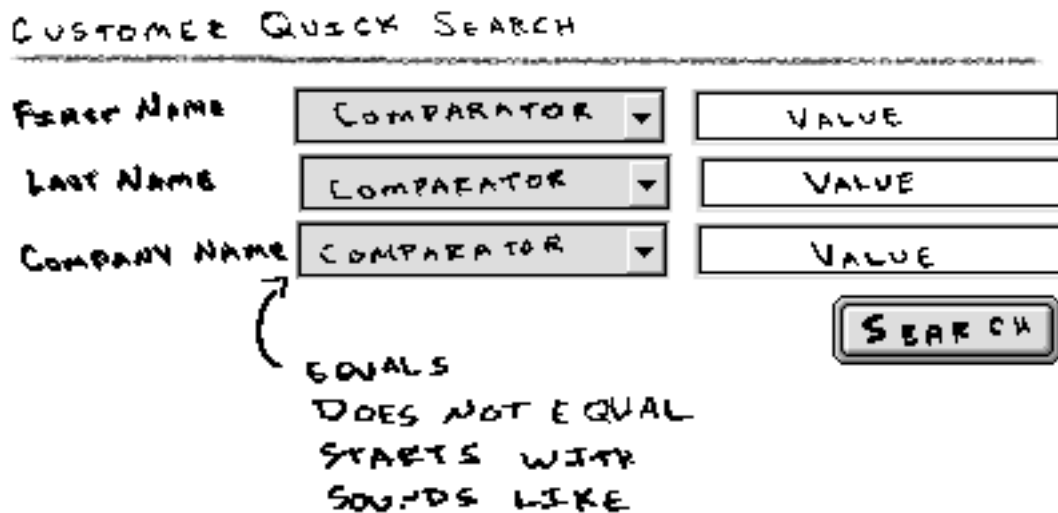


The "comparator" popup lets the user decide how to search.

The second design adds a new feature to the first design. Now the search allows the user to specify a comparator for the search: =, #, > or <. This design remains "quick and easy," yet offers greater flexibility.

A Multiple Field Search Interface

The single field search design is definitely easy for the user, but it may prove inadequate. Let's look at a few versions of a multi-field search. The first approach lets the user specify search values and comparators for all three fields:



The user can see and search on all three fields.

This design lets the user search on all three fields. An advantage of this design over the earlier “simple” screen is that the choices are fully exposed. Rather than hiding the searchable field names in a popup they are visible at a glance.

This design allows the user to enter search values for more than one field, but how would they look for “Customers whose last name # empty”? The current design has no way of knowing what it means when you don’t enter a search condition. Does it mean “search for empty” or does it mean “ignore this field in the search.” The next design offers a solution to this problem.

CUSTOMER QUICK SEARCH

FIRST NAME	COMPARATOR ▾	VALUE	<input type="checkbox"/>
LAST NAME	COMPARATOR ▾	VALUE	<input type="checkbox"/>
COMPANY NAME	COMPARATOR ▾	VALUE	<input type="checkbox"/>

→
EQUALS
DOES NOT EQUAL
STARTS WITH
SOUNDS LIKE

SEARCH

The checkboxes next to each field let the user specify which fields to include in the search.

Here small checkboxes allow you to indicate if the search value is included in the search or not. A script behind the value and comparator objects could turn the check boxes on if they are manipulated to partially automate selecting these check boxes.

This design still leaves it ambiguous how the fields should be combined. Does the user want to find matches on *any* (logical OR) field or only on *all* (logical AND) fields? You can assume one or the other, or let the user make the choice:

CUSTOMER QUICK SEARCH

FIRST NAME	COMPARATOR	VALUE	<input type="checkbox"/>
LAST NAME	COMPARATOR	VALUE	<input type="checkbox"/>
COMPANY NAME	COMPARATOR	VALUE	<input type="checkbox"/>

MATCH ANY MATCH ALL

← "AND" EQUALS DOES NOT EQUAL STARTS WITH SOUNDS LIKE ← "OR"

SEARCH

The radio buttons allow the user to control how the search conditions are combined.

Five Designs In Ten Minutes

It took about ten minutes to build these five designs. Building the resulting screens takes considerably longer as variables need to be named, defaults set, arrays populated, and objects arranged. Even without implementing the search, the screen building time is significantly longer. In our experience it takes about five to ten times longer to build a screen in 4D than on paper.

Which Design Is Best?

Which of the five sample designs is best? We don't know. Chances are that none of them is the ideal design. The first approach may not be powerful enough, and the final design may be too complicated. As computer professionals and software designers we suffer from an enormous deficiency: *we understand computers too well*. The interface makes sense to you since you know how it works. It is not safe to assume that even a "simple" search interface is easy for users, or meets their needs. The only way to meaningfully test an interface is by trying it with users. The earlier in the development process you perform user interface and usability tests the cheaper it is to refine the interface. Now let's talk about prototyping and user testing in more detail.

Prototyping and User Testing

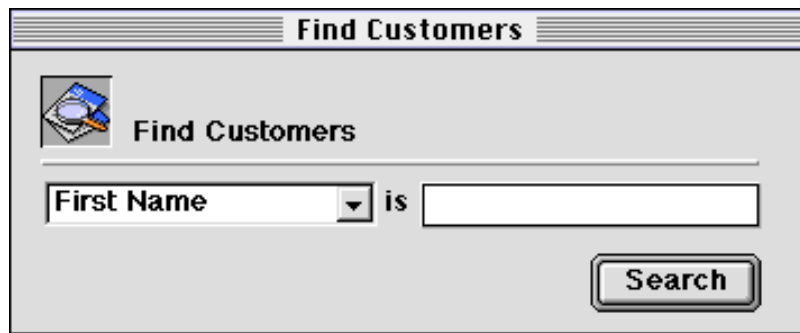
Prototyping Saves Time and Money

The key benefits of RAD tools are that they make prototyping, user testing, and system implementation quicker and cheaper. If you neglect prototyping and user testing you are throwing away two of the key benefits of a RAD tool like 4D. You will do prototyping and user testing, even if you don't call it that. If you deliver a "finished" system your users will test it. When user's test a prototype you get sug-

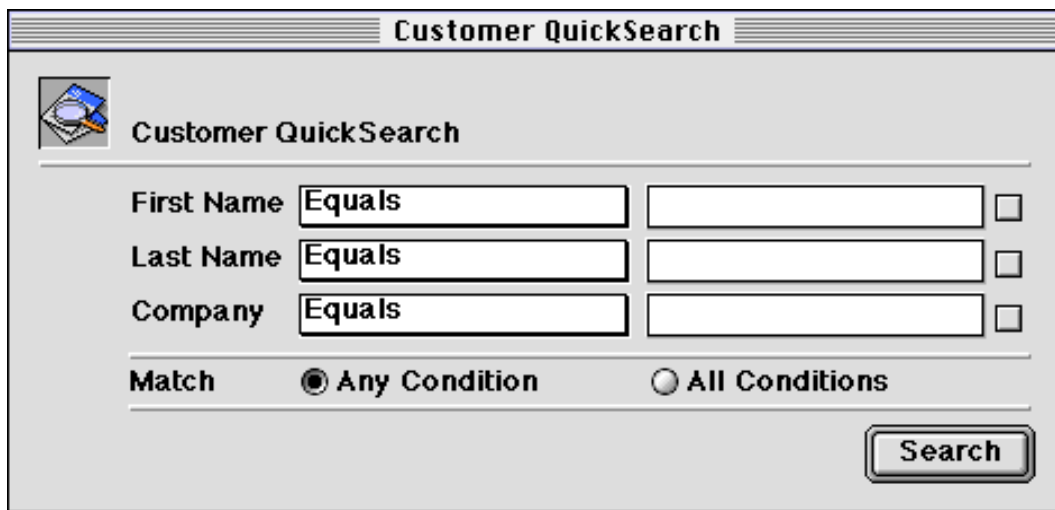
gestions and information, when they test a finished system you get bugs, feature requests, and complaints. Ironically, skipping prototyping and user testing to “save time” ends up making projects take longer and cost more.

Building Paper Prototypes

Earlier we discussed how to build screens on paper to save time and money. Paper prototyping is a logical extension of paper screen design. You can work with neatly rendered paper screens, or build mock-ups in 4D that you print out for the user. Here are polished example paper prototypes of the screens we sketched in our discussion of designing screens On paper:



The single field search screen.



The multiple field search screen.

These look considerably better than the original paper designs! A few graphic touches have been added to make the screens look finished. It is a good idea to make paper prototypes appear polished so that the user pays attention to the behavior of the screen without being distracted by cosmetic defects.

Paper prototyping is a widely used technique on large software development projects. Rather than building final screens the design team produces paper mock-ups of screens to show to users. There are special languages and environments for creating attractive paper screens. Graphic artists often produce the mock-ups instead of programmers. We like building the screens in 4D since its form tools are as easy as most drawing programs, and the form can ultimately be developed into a final system. The screens shown here have code to load the interface objects like the arrays, radio buttons, and window title. There is no code to make the interface objects work, or perform the search. At this point it is unclear which—if either—of the design approaches we'll use. Time spent writing interface control and search code is wasted if the screen is not a part of the final system. A key advantage of prototypes is that you can test and examine how the system works before building the entire system. You can get user feedback early in the design cycle when it can save you the most time and money.

Testing Paper Prototypes

After preparing paper prototypes, like those shown above, you can test them with end users. Large companies like Microsoft, Apple and Sun use special usability laboratories. Users perform a series of tasks following instructions from trained researchers. The proceedings may be videotaped for later analysis. Well, that's a bit expensive for most of us. You can get useful results inexpensively with some good questions and a paper prototype. Here are some testing guidelines:

- ✓ Prepare your screens and questions ahead of time.
- ✓ Tell the user what you are doing. Explain that you're prototyping the system and want to see how close the prototype comes to doing what they need, the way they need it.
- ✓ Give them a written description of the exercise steps. Some people hear instructions well, others understand better from reading.
- ✓ People can be intimidated by interviews or activities that look like tests, so make it clear that they are not being tested, the *system* is being tested.
- ✓ Tell people if their contributions are anonymous and confidential or not. Any approach can work if you explain it clearly and honestly.
- ✓ Remember that they are the experts and they are helping you.
- ✓ Everyone's time is important. Tell them how much time you need, start on time, and do not keep them late.
- ✓ Interview different categories of users. Whenever possible test users with different computer skill levels, and different positions in the organization. Managers and end users have radically different perspectives and objectives.
- ✓ Determine beforehand exactly what information you will reveal. Do not answer extra questions about how the system works as you are trying to find out how a user reacts to the system in the more or less "normal" situation of not having an expert at their elbow.

- ✓ Ask them to vocalize their choices and thought processes. If they comply, you'll find that what they say as they work through confusing points often provides the most valuable information you get.
- ✓ Tell the user how to contact you later if they have additional comments or questions.
- ✓ Thank them for their time and comments!

A simple way to conduct interviews is to write out a series of tasks that the user should attempt to “perform” with the paper prototypes. If you have a series of interrelated screens you can give them paper prototypes of each, without giving them any verbal queues about how the screens are related. The interface itself should carry that information effectively. If it doesn't you've learned something valuable.

Example

The two screen designs pictured earlier in this chapter were built to satisfy this requirement:

We need a customer lookup screen that lets us find customers by first name, last name, or company name quickly and easily.

It is unclear if the user needs to be able to search on more than one field at a time, so we've got two different designs to test. Here are the kinds of tasks you could ask the user to “perform” on the paper prototypes:

How do you think you would use this screen to find all customer's who live in California?

How do you think you would use this screen to find all customers with a last name starting with “A”?

How do you think you would use this screen to find customers who don't have a first name listed?

How do you think you would use this screen to find people who work for “ACME Black Dot”?

How do you think you would use this screen to find people who work for “ACME Black Dot” and have a last name starting with “A”?

Look over the paper screens and try to answer the questions yourself. You'll notice that some of the target tasks cannot be performed on either screen, and that some are only possible in one. It can be fruitful to ask questions like these as they teach you a lot about how the user goes about solving problems. You can also ask the user open-ended questions about the interface or the screens themselves:

Can you suggest a way of making this screen easier to use?

What is your reaction to the appearance of this screen?

Does anything seem to be wrong with this screen?

Does anything seem to be missing from this screen?

Is there something you often need to do that would be hard with this screen?

Not all users can answer questions like these, but when they do they often give you the single most valuable piece of information you obtain through the entire interview. Answers to questions like these have a chance of revealing information you didn't know to ask for. You'll find that open ended questions like these work well with knowledgeable users working with detail listings and report layouts.

So Many Questions, So Little Time

The test screens we've been exploring in this discussion are simple. They perform a single basic task, and have no related screens. You can see already that there are multiple design approaches to this screen, and many questions you can ask. With more complex screens—and interrelated screens of any kind—there are more alternatives to consider and questions to ask. Some people react to this by saying that the process “will take too much time” and is “too complicated.” This is not the case. You *will* perform this research, the only question is how and when. If you produce a final screen that does not satisfy the user's requirements, or that they cannot use effectively, you will have to do expensive rework, have a sub-optimal system, or both. Prototyping does not eliminate the need for further testing and corrections, but it allows you to do a great many corrections when they are inexpensive to implement. We suspect that the real reason many people avoid prototyping and user testing is that coding is more familiar. Once you start working with prototyping techniques you may discover they are incredibly fun. If you do prefer coding perhaps there is someone else on your team who loves to work with people and user interfaces. Interviewing, user interface design, usability testing and coding do not have to be performed by the same person. Non-programmers can make better interface testers and designers because they are not always thinking about how to implement the screens.

Special Advantages Of Paper Prototypes

There are several additional advantages to paper prototypes:

- ✓ You have a record of the screens. This can become a part of a specification that the user signs off on. This makes it easier to calculate how much time it will take to complete the final job since the user has already agreed what the end result will look like and do.
- ✓ You can present the prototype to a large group with an overhead or a hand-out. No computer equipment is required, and so you don't have to worry about hardware hassles or crashes.
- ✓ You can fax, mail, or e-mail the screens to remote users for comment.
- ✓ You can put pictures of the screens on a Web page for collaborative commenting.
- ✓ Everyone can write on their copy to simplify pooling comments.

Are These Techniques For You?

What Are You Doing?

Usability oriented researchers have found that organizations are most likely to adopt techniques like the ones we've discussed today if they are actively engaged in design. Organizations that are involved in other phases of implementation—like coding or installation—are unlikely to adopt these techniques. This makes sense. If you're late in the development cycle why would you adopt tools that give the greatest benefit when used early in the development cycle? Well, these techniques can easily pay for themselves even late in the development cycle, but we understand that you're probably too busy to save time at that point ;-) If you're not actively engaged in design at the moment keep these techniques in mind. The next time you're doing design work review these notes and try out one or more of these techniques.

Adapt These Techniques

Many of you have already noticed that we're talking about techniques you already use in one form or another. That's great. Some of these techniques are borrowed from other disciplines, like anthropology, some we adapted from traditional usability engineering techniques, and others we made up. There is nothing absolute or definitive about our techniques: adapt them freely. If you already have techniques you like, or you improve the ones we've presented, please let us know.

This Stuff Isn't Serious Enough

Some people won't adopt these techniques because they don't seem serious enough. Rest assured that the techniques we've described—or similar ones—are used by the world's largest and best known software companies, including Microsoft, Sun, and Apple. If you have multiple designers or programmers on your team see if these techniques appeal to one of them. Someone who finds these ideas exciting will have a good chance of applying them in a way that is effective and engaging for everyone.

Step Away From The Keyboard!

These guerilla design techniques all emphasize getting away from the computer. This is not an accident. Even if you're working alone one of the best ways to improve the quality of your code is to get away from the computer. Have you ever sat in front of the keyboard for hours in frustration trying to solve a problem? Sure you have. How many times did the answer pop into your head when you walked out the door? Almost everyone's had this experience: cultivate it. These techniques let you use different parts of your brain than normal computer work, which ends up giving you more and better ideas for when you do go back to programming. And if this turns out to be fun, you don't have to tell.