a common sense approach to web & mobile application design

second edition

robert hoekman, jr.

Designing the Obvious:
A Common Sense Approach to Web and Mobile Application Design, Second Edition
Robert Hoekman, Jr.

New Riders
1249 Eighth Street
Berkeley, CA 94710
510/524-2178
510/524-2221 (fax)

Find us on the Web at www.newriders.com
To report errors, please send a note to errata@peachpit.com

New Riders is an imprint of Peachpit, a division of Pearson Education

Editor: Wendy Sharp
Production Coordinator: Hilal Sala
Copyeditor: Jacqueline Aaron
Research Assistant: Sunny Thaper
Compositor: Danielle Foster
Indexer: Emily Glossbrenner, FireCrystal Communications
Cover design: Mimi Heft
Interior design: Joan Olson

ISBN 10: 0321749855
ISBN 13: 9780321749857

9 8 7 6 5 4 3 2 1
Printed and bound in the United States of America

Contents

This page intentionally left blank

4

Build Only What Is Absolutely Necessary

- Think Different
- Think Mobile
- Drop Nice-to-Have Features

When applications evolve based on the demands of users (or of CEOs), they tend to take a bad turn. Features used by only 10 percent of users or used only 10 percent of the time are added and get in the way of the remaining 90 percent of features. They clutter an otherwise clean interface. They interfere with the features used most often.

And when "featuritis" takes over, you quickly find yourself permanently providing tech support for things that shouldn't be in the tool to begin with, fixing more bugs, writing more Help material, and neglecting other, more important features. And while this may sound like a lot of fun to certain (slightly crazy) programmers, it's clearly the wrong approach.

The focus should not be on features, the focus should be on *focus*. An obvious application is a focused application. It's easy to explain to other people. It makes sense to those using it because the purpose of the tool is self-evident, and nothing in it strays from that purpose. Every feature supports the single situation the application is designed to support.

More Features, More Frustration

A user's frustration level doesn't map directly to the number of difficult features in an application. Frustration increases *exponentially*. For every additional feature, there is more to learn, more to tweak and configure, more to customize, more to read about in the Help documentation, and *more that can go wrong*.

For you, it's one more feature. For users, it's one more thing that adds to the already long list of frustrating things to deal with while using a computer. It's not just your application—it's everything else, too. It's the operating system, which hides files away in obscure directories and is constantly popping up little dialog boxes and error messages that you are forced to address before you can get on with your real work. It's the browser, which has no graceful way of indicating whether or not the link you just clicked is going to show you what you want to see. It's the email client, which offers no insights into how long it will take for the message you just wrote, with the rather large photo of your cat, to be sent to your grandmother.

Users contend with all these things and more during the same stretch of time they try to deal with your application. And the frustrations add up quickly.

I know, I know—none of these things bother you at all. They don't really bother me either. But that's a sad fact. It means we've become desensitized to things that are otherwise maddening. It means we've gone numb.

In short, we've become "computer-savvy."

So what's a geek to do?

You have to have killer features or your application won't be able to stand up to its competitors. Right? You have to keep adding things to new versions or no one will purchase upgrades and sales will stagnate. And you have to match the competition one-for-one so no one can ever say your application is light where the other guy's is robust. *Right?*

To paraphrase Alan Cooper, however, trying to match competing products feature-for-feature is like running through a battleground under cover fire. You can run all you want, but you have to keep shooting to get anywhere. Dishing out cover fire keeps you alive for a few minutes at a time. Long enough to hide. Companies that fight all the time to stay ahead fall into the endless cycle of trying to outdo the enemy (if the enemy has a big gun, you need a bigger gun). This goes on and on until someone falls. It's not a fun way to do things. It's a method that works only as long as the people fighting the battle continue to come up with bigger guns. They spend all their time spraying out cover fire while they run 3 feet to the next safe position.

Many companies live and die this way. To get into the fight, you have to stock up on venture capital, go into major debt, hire a bunch of rock star developers, go straight to code because there's no time to plan or design anything, and rush, rush, rush to market with a 27-page list of features. And if the enemy starts to catch up, you have to add more features, call the next version "the most robust release ever," and try to maintain your market share. Until, of course, the enemy puts out a new version with even *more* features.

It's exhausting.

It's also exhausting for users. The more features you offer, the more the user has to learn. The more options you provide, the more users have to do to get anything done. The more you allow customization, the more users have to fidget and tweak and manipulate your application. They spend more time configuring the tool than using it. As a result of fighting the fight, complicated applications often end up much less usable than one would hope.

To stay alive, you eventually have to get out of the line of fire. It's the only real option.

Think Different

A few years ago, Firewheel Design (www.firewheeldesign.com) got out of the line of fire by creating Blinksale (www.blinksale.com), a web-based invoicing system. The simple application contains only the features that are absolutely necessary for the largest percentage of its users to successfully create, submit, and track invoices.

Firewheel's decision to minimize Blinksale's feature list might look like a mistake because it seems as if it won't be able to compete in the rat race with the big boys of invoicing systems. But the small crew at Firewheel did something the big boys hadn't done: it created something that stood out.

Blinksale is aimed at contractors who don't need to do anything fancy with their invoices. Many people who need to submit and track invoices need only a few basic tools. These include a way to create the invoice, submit it, mark it as closed when payment is received, and perhaps send a receipt confirmation to the client. When the folks at Firewheel Design set out to create Blinksale, they realized they could keep it simple and satisfy the vast majority of user needs. They may have even realized that making it more complicated would decrease their chances of satisfying user needs. So they designed a web application that does one thing, and does it very well: it gives people a fast and effective way to create, submit, and track invoices.

(See how easy it is to explain? That's a good sign.)

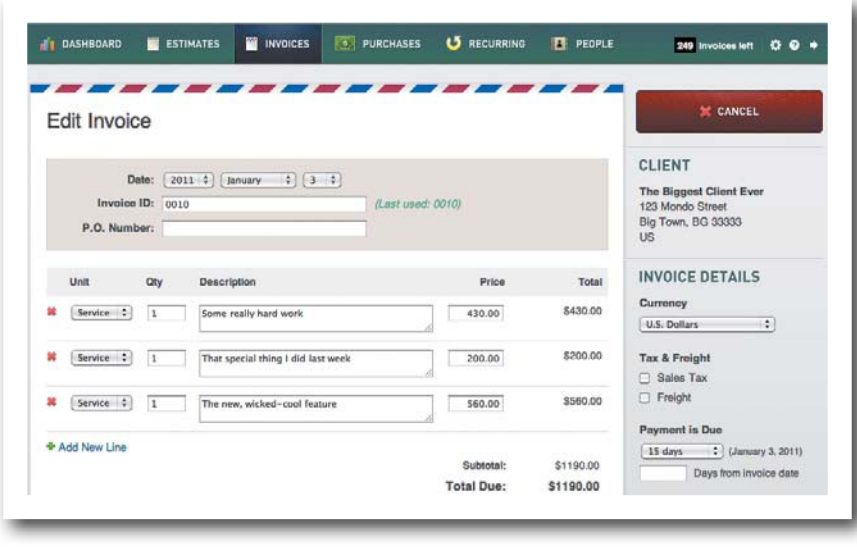
Since Firewheel created Blinksale, it's been taken over by Doublewide Labs. It's even been completely redesigned. Amazingly, it's *still* one of the best applications around.

The system can be used by plenty of people besides contractors because it's so stripped down that a trained monkey could use it (assuming the trained monkey could type). The application contains just a few key features.

The dashboard and New Invoice screen

When you sign in, Blinksale shows you a summary of your recent activity (open invoices, past-due invoices, and so on) so you get a quick, at-a-glance, dashboard-style view of the state of your invoices. It also offers an easy-to-spot New button, to start creating a new invoice.

You simply choose the client the invoice is for or create a new one—right there, on the same page—and hop over to the New Invoice screen. This page actually *looks* like a real invoice, so you maintain context the whole time you're creating it. All the fields you need to complete are displayed as form elements, so you can simply edit the invoice onscreen and click the big Save button.

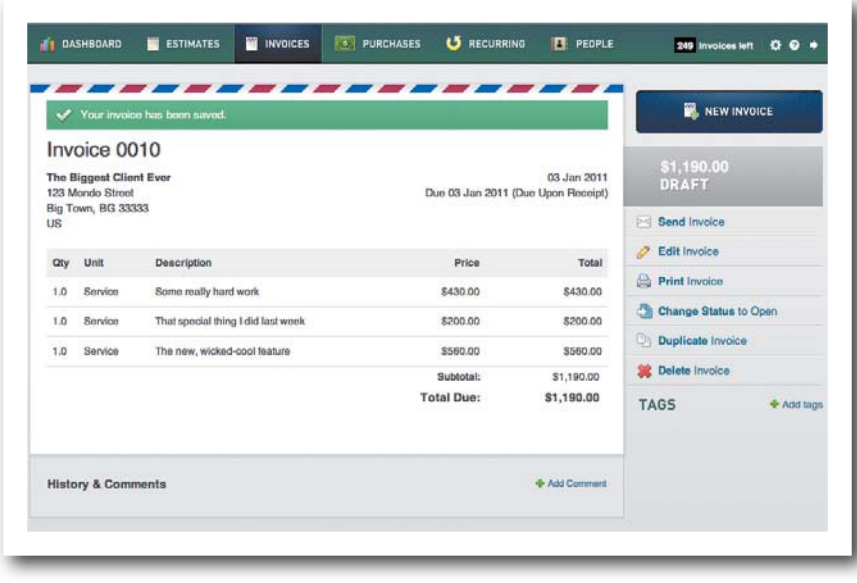


Blinksale's main invoice-editing screen is easy to use.

When you're done, you see the final version of the invoice and a few new buttons, which let you send the invoice, edit it, or delete it. One click of the Send Invoice button produces an in-line form in which you checkmark all the people in the client company to whom you want to send the invoice and write an optional message.

The finished invoice

The invoice itself is an HTML-formatted email that looks great right out of the box (well, the browser), and you don't have to configure anything at all to send off a professional invoice to a client in five minutes or less.



Blinksale generates easy-to-read invoices and lets you email them to your clients in a click.

Simple as that.

Blinksale offers a few basic templates from which to choose how you want your invoices to look. You can also send reminders to clients about late payments and create thank-you messages to send to clients who pay their bills on time.

The whole application takes less than 30 minutes to learn inside and out, and just about pays for itself every time you create an invoice (at the time of this

writing, Blinksale offers a $6 per month plan for up to 6 invoices, with plans of up to $24 per month for 250 invoices.)

Firewheel built only what was absolutely necessary for most people to successfully handle the activity of invoicing clients. And Doublewide Labs has lovingly maintained that tradition. There are no obscure configuration options, no redundant functionality (there's exactly one way to complete each task in the tool, which makes it easy to learn), and no fancy interface widgets to figure out. It just does exactly what it should, and does it within a simple, clean interface that somehow makes invoices seem friendly, like someone you'd want to take to lunch. (We'll talk more about software personality in Chapter 10.)

The result

Josh Williams, one of the creators of Blinksale, was justifiably proud of how things turned out. Back in 2004, he told me:

> *As a small design company we did our fair share of client billing. Unfortunately we've always been less than enamored with the off-the-shelf invoicing and billing software that is available at your local office supply store. After a few years of frustration we set out to build our own web-based invoicing service. Goal number one was ease of use. Goal number two was keeping our cost of design and development of the service low. Remarkably, these two goals often go hand in hand.*

Firewheel could have designed Blinksale to be chock-full of features that did everything from integrate with Intuit QuickBooks in 12 easy steps to preparing tax information and letting you export it to Intuit TurboTax at the end of the year. They could have built a product that rivaled its competition feature for feature. They didn't. They built the 20 percent people actually need. Nothing more. Nothing less. Even after a change in ownership, with Doublewide Labs at the helm, Blinksale is still hyper-focused on only the most essential features.

While there are a few extra gadgets thrown in for more computer-savvy users, Blinksale keeps things simple and focused. If all you want to do is create

an invoice and send it to a client—the single task most people will spend most of their time completing in Blinksale—you can do it in just a few minutes and be on your merry way.

Think Mobile

One of the best ways to avoid feature battles is to focus your attention on designing for mobile platforms such as Apple's iOS and Google's Android. Strategically, it's also one of the best things you can do for your business. Consider this story:

Recently, after boarding a flight to San Francisco for the Voices That Matter conference (hosted by New Riders, of course), a man in his early 30s sat next to me and pulled out his iPhone. The older gentleman who sat on the other side of him asked about it. What was it like to use it? How easy was it *really*? Wow, it sure does look fast, and neat. The younger man answered every question with growing enthusiasm. I'd seen it a hundred times before—the iPhone frequently turns otherwise perfectly jaded people into vehement, adept Apple sales representatives. But then the younger man said something that surprised me. When the older man asked the younger man what he did for work, he replied:

"I'm a cop."

He wasn't a designer. Or a marketing guru. Or a social media expert. Or an entrepreneur. He wasn't heading to a tech conference or a sales seminar. He wasn't at all the kind of person I normally see have a conversation like this one. He was a cop—a middle-class guy who puts on a blue uniform every day and relies on walkie-talkies to communicate—heading off to meet some old college friends for the weekend.

"I hardly ever use my computer anymore. I can do it all on this thing."

Touchscreens and gestural interfaces have taken rise. Long gone are the days when the Internet was considered the new frontier. Personal tech has taken over. Devices are the *new* new frontier.

And this, my friend, is a very good thing.

Over the next few years, as the obstacles to the adoption of mobile devices are eliminated, more and more people will trade in their desktop and laptop computers and start using devices exclusively. Apple and its competitors in the smartphone and tablet markets will make sure of that. And while many people in the tech industry still see some of these gadgets as luxury items—often even wondering what on Earth they would do with a tablet—these devices are designed for the other 99 percent. They're designed for that large segment of the population that uses computers for paying bills, social networking, making plans, watching videos, checking the news, listening to music, digging up recipes, learning new skills, creating spreadsheets for work, writing memos, and of course, checking email. These people use computers primarily for media consumption, web browsing, and basic document-creation. And that's exactly what the iPad and other tablets are designed to do best.

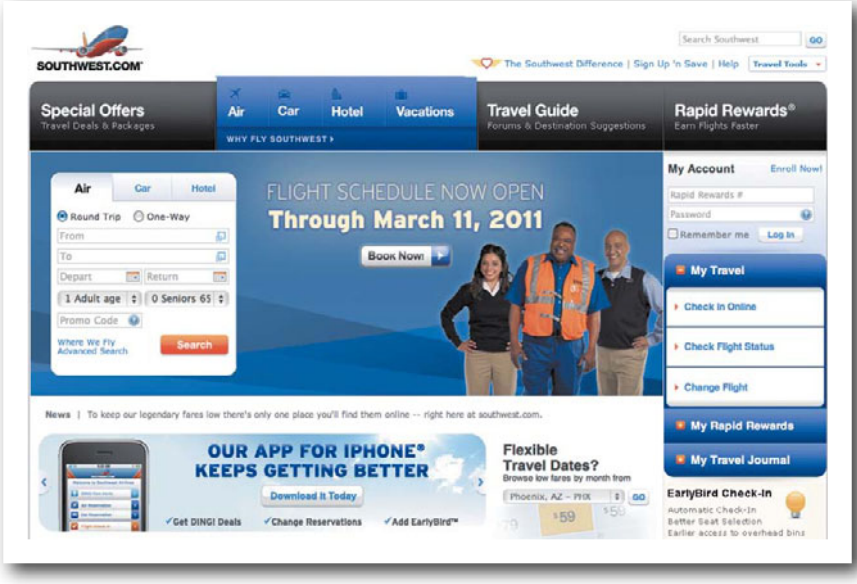
As these products evolve and get cheaper, it will simply be more affordable and more useful to buy a touchscreen tablet backed by a catalog of cheap and easy-to-use apps, with its ever-expanding array of possible use cases, than it will be to buy a desktop, laptop, or even a netbook. More people than ever before will be able to empower themselves through the Internet, and they'll be able to use it anywhere they want. The air will be completely filled with Wi-Fi signals, and all the world's information—all *your* information—will be quite literally at your fingertips, anytime, anyplace..

Personal tech is now affordable by the masses, useful for the masses, and usable by the masses. And this will only become more true in the years to come. If you're not designing for it now, you're already late.

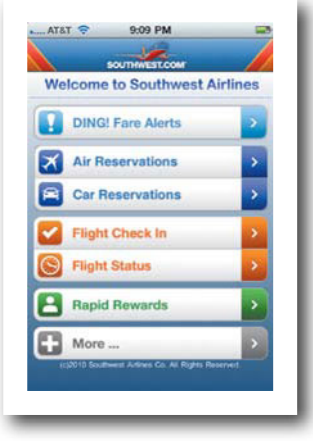
But even if you ignore this fundamental shift in personal computing, devices are good for application designers for other reasons—specifically, because they force designers to follow the principles of good application design.

The people designing for devices right now are doing a better job of embracing these principles than most people designing for the desktop *ever* have. The constraints of the medium—the limited screen space, risk of network slowdowns, difficulties of multitasking, and so on—are having the happy side-effect of encouraging designers to design more concise applications.

Southwest Airlines offers a great example. Here's the site in a desktop browser.



Naturally, the site offers a way to book a flight, car, or hotel. In fact, in this desktop version, there are two sets of tabs that offer this, one of which lets you fill out the reservation form on the homepage, the other of which takes you to another page to do it. Why? Well, because the larger tabs at the top are more than just links—they're menus. The Air tab, for example, offers a menu chock-full of links to other information, including a list of destinations Southwest flies to. And yes, plenty of people will seek out this information. But the natural effect is that the design takes up a lot of space, offers a lot of options, and requires a lot of scanning and decision-making.



The Southwest Airlines iPhone app, however, focuses only on what's absolutely necessary. On the iPhone, users can find this information by checking the To Where option on the Book a Flight screen.

Now, let's say you want to see how your Rapid Rewards stockpile is coming along. On the desktop version, there is a large Rapid Rewards tab-like link near the top of the page, an accordion tab labeled My Rapid Rewards, and a sign-in form that asks for your Rapid Rewards number. Which one is right? Which step do you take first?

In the iPhone app, you tap *Rapid Rewards.* It leads to a sign-in screen.

One of these things is so much clearer than the other.

Hey, it's your life

So if you still wonder what on Earth you'd do with a tablet, there's your answer. You'd secure your own future as a designer. As a marketing guru. A social media expert. An entrepreneur.

Don't neglect to see the significance of mobile computing because you're busy sitting at a desk with a souped-up PC, a killer video card, and 8 feet of monitor space displaying 75 open Photoshop files. Yes, you'll have to continue doing a lot of your design work there. Yes, it will continue being *easier* to do design work there. But don't delude yourself that your customers will always and forever be sitting at a desk when they use your products. They won't be.

Use your desktop computer. Just use it to design for *devices*. And make sure a tablet is sitting next to it.

Throughout the rest of this book, I'll discuss mobile-specific considerations alongside our discussion of design principles for effective applications.

Not present at time of photo

Sadly, at the time of this writing, there is no device-friendly version of Blinksale. Hey, nobody's perfect.

A competing invoicing application, Ballpark, does offer a mobile-friendly dashboard. It's not much, but it offers a cursory view of your recent activity.

Drop Nice-to-Have Features

Almost every mature application in existence contains at least a few features that were probably first described in a statement that started with "Something that would be really nice to have is <*insert description here*>." But most of these things are exactly what clutter up interfaces all over the web and on our devices, and it's our job to fend these things off with a big stick. They need to be removed from your next application before it's even built. An obvious interface is one that is focused on what's most important and leaves out the things that are simply nice to have.

In its book *Getting Real*, 37signals has this to say about focusing on only the important features:

> ***Stick to what's truly essential. Good ideas can be tabled.*** **Take whatever you think your product should be and cut it in half.** ***Pare features down until you're left with only the most essential ones. Then do it again.***

The statement is similar to something Steve Krug said in his book *Don't Make Me Think*, one of the greatest books out there on web usability. It's Krug's Third Law of Usability:

> ***Get rid of half the words on each page, then get rid of half of what's left.***

And Krug's law can be traced back to William Strunk, Jr., and E. B. White's *The Elements of Style*:

> ***Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentences, for the same reason that a drawing should have no unnecessary lines and a machine no unnecessary parts.***

Say it again, brother.

All these people are in the business of simplicity. Simplicity makes the point clear. It lets messages stand out. It offers communication that cuts through the noise.

The Unnecessary Test

To create applications that cut through the noise, you have to be willing to slice your application's feature list down to its bare bones, and you have to recognize what's most important.

With that in mind, try the following exercise, which I call the Unnecessary Test:

Open an application you've worked on recently and find a feature you thought was really important a long time ago, perhaps before you started building the application.

Ask yourself the following questions:

1. Is there more than one way to complete the task this feature supports?
2. Does this feature contribute directly to the completion of the task?
3. Is the task this feature supports vital to the activity this application supports?

If you answered no to any of these questions, the feature may be unnecessary. You've found yourself a likely candidate for the cutting room floor.

If, on the other hand, you answered yes to all of these questions, you're either looking at a rock star feature or you're not looking hard enough at the feature to be objective. Try your best to detach yourself from all the work you did and ask these questions from a more objective point of view.

Regardless of your answers, it's likely there are several features in your application that could be scrapped, so you should take the time to go through every feature and run each one through the Unnecessary Test.

When you're done with the testing, close the application and ask yourself three more questions.

1. What are the circumstances of the situation my application is meant to support?
2. If this application didn't exist and I needed to handle the same situation, and I could wave a magic wand to create an application that helped me with this situation with the greatest of ease, what would the application do? (Hint: You should limit this answer to a few very big-picture statements that relate to the principal desired outcome.)
3. How long will it take to rebuild my application to make it do that?

Sorry—that last question is a joke (sort of). After all, you're likely to have answered one of the first two questions in a way that prevents you from having to admit you were wrong. I know—I've done this myself. It's difficult to admit your application may not be living up to its promise.

If this is true, have someone else answer the same set of questions and see if the answers are different. Even better, ask one of your users.

I'm not suggesting you start ripping functionality out of an existing application. Doing this could have the rather negative side effect of making some of your users extremely upset. To the people using the more obscure features, removing them would be a huge mistake. I'm only suggesting you learn from what you've already done so you can create more focused applications in the future.

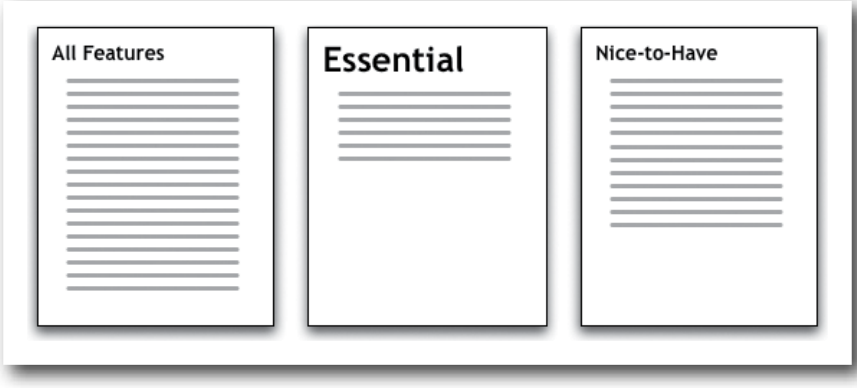
The 60-Second Deadline

Here's another quick way to learn to effectively aim low and keep your application focused on the 20 percent that matters:

Pretend I'm your boss. I walk into your office and very matter-of-factly state, "The project time line has been cut in half. We have about 60 seconds to decide what to keep and what to throw away before we meet with the client in the conference room."

How do you respond to this statement?

Whatever you do, don't impulsively offer up the theoretical answer—the one where you say how much you'd love the low-carb sandwich. *Figure out the real answer.*



Grab a notepad and a pen, write down the list of features you have planned for an upcoming application, and see what you can cut in 60 seconds. Draw a line through each feature you can cut without completely destroying the application.

The goal is to leave yourself only with what is most essential for the application to serve its purpose.

Bells? Gone.

Whistles? Gone.

Show me only the pieces you absolutely have to keep for the tool to do its job.

When you're done, cut one more feature, just for good measure. Cut the one you're holding onto only because it's really cool. C'mon, I know there's at least one on your original list. Draw a line though it.

Your 60 seconds are up. Good job.

Now, take out a second sheet of paper and write a new list that shows only what you have left, just so you can see it sitting there all nice and clean. Looks much better, doesn't it? I know, it probably hurts a bit to have lost so much stuff, but I bet your application is now easier to explain.

Finally, take out another sheet of paper and write down the list of things you drew a line through earlier. Title this page "Nice-to-Have Features," stick it in your filing cabinet, and forget about it. We'll look at it again later.

The first time you do this, it can be quite revealing. You may find you've been wasting a lot of your time and energy on things that don't really contribute to the application in any meaningful way. Of course, this may be a bit unsettling, but hey, knowing is half the battle. Next time around, you can use the Unnecessary Test and the 60-Second Deadline exercise before you start coding, to see what really needs to be built—and you can spend all your time working to make those things as good as they can be.

And since building what's most important takes much less time than building what's not important, you can get more sleep, take more vacations, get more weekends off, and live a happier, healthier life.

Or you could do what I do and use all that saved time to design more applications. I know that's what you *really* want to do.

Aim low

Regardless of how you do it, the ultimate goal is to determine what's most important to the application by whittling your list of features down to about 20 percent of what was built or what you were planning to build. Yes, some of the remaining 80 percent of your features may be useful somehow, to someone, some of the time, but they are most likely useless to 80 percent of your users, 80 percent of the time. And you probably spent 80 percent of your development time building things that aren't essential to the application.

This is because the 80-20 rule has made its way into the world of software.

Known formally as the Pareto principle (named for Vilfredo Pareto), the 80-20 rule was originally suggested to indicate that 80 percent of the effects come from 20 percent of the effort.

In terms of good, clean application design, it means that 80 percent of an application's usefulness comes from 20 percent of its features. It also works the other way around, to illustrate that 20 percent of the development work produces 80 percent of an application. The other 80 percent of the work satisfies only 20 percent of the outcome.

To create more focused applications, stick to building the 20 percent of features that are essential—the ones you'll stick on the mobile version—and you'll

take care of 80 percent of the user's needs. Let your competitors worry about the rest. While they're floundering around trying to one-up you by fleshing out the other 80 percent of the application, you could be taking 80 percent more vacations and enjoying 80 percent of the market share.

Less is more. Aim low.

Interface Surgery

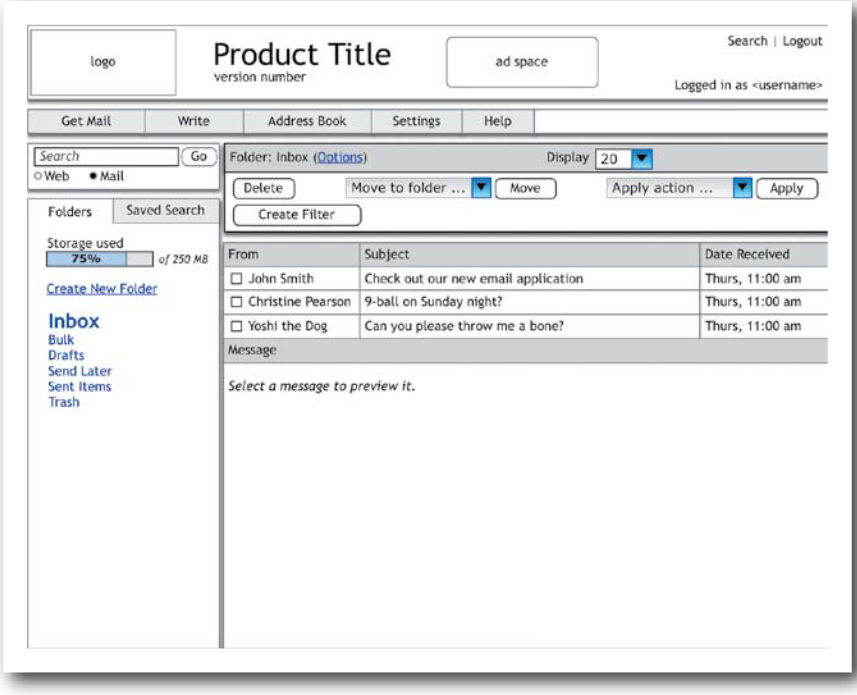
A job application form I saw once was composed of two windows. One window got the user through the first few screens of the process, and then it launched a second window to complete the bulk of the application. The first window was connected to the user's log-in session, which was timed and was designed to log out the user automatically if the system remained inactive for 20 minutes. However, the second window was not tied to the session. So, when a user tried to complete the job application in the second window—the part of the process that took the longest amount of time—the system invariably logged the user out after 20 minutes, rudely doing so without any notification whatsoever.

The company's solution was to add a bit of text in the original window warning users that they would be logged out after 20 minutes—a weak attempt to get their pesky users to stop complaining. This was a band-aid. It did not solve the problem, it just told people what to expect. Users would still have to complete the job application in 20 minutes or less. The company was essentially saying, "Sure, we've created a terrible system that will likely terminate your session before you can complete your job application, but hey, we're warning you before you start, so it's okay!"

I don't like band-aids.

Instead of putting band-aids on problems, I perform surgery on them. Interface surgery.

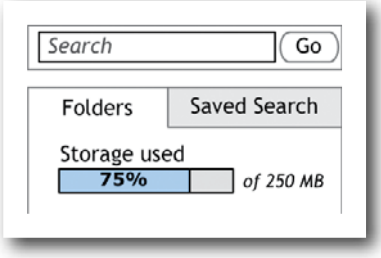
In this first installment of Interface Surgery, we'll cut out a bunch of unnecessary features from a fictitious web-mail application. Instead of finding ways to make a ton of unnecessary gadgets easier to present and use, we're going to rip them out and leave only what's absolutely essential for the application to do its job.



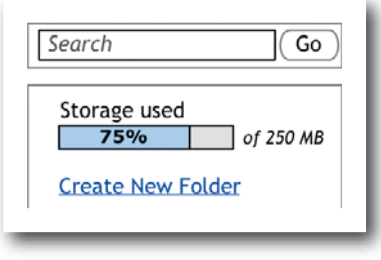
This application has a ton of features. In addition to being able to simply check your email, you can search the web, see how much storage space you've used, make sure you're logged in using a particular user name, reuse saved searches, apply actions (such as set up an automatic response email), move email to other folders you create yourself, configure options for the Inbox (such as font settings), and even change how many messages should be displayed in the list before having to switch to a new page.

Some of these things are necessary, some are not.

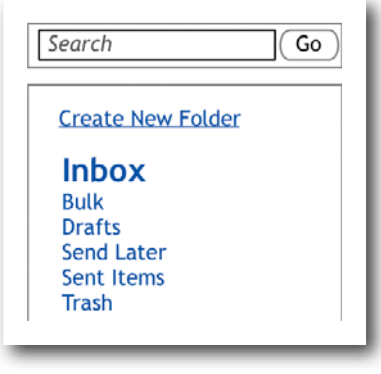
To get started, let's strip out the part of the Search feature that lets users search the web. There are already plenty of ways to search the web, and most modern browsers feature a built-in search bar, making this action accessible 100 percent of the time the user has the browser open. There's no need to replicate what's already ubiquitous. And since we're leaving only the option to search mail, we can remove the two radio buttons and shrink down the space this piece takes up.



Let's also get rid of the ability to save searches. It's more difficult to save a search, find it again later, and rerun it than it is to simply reenter a few keywords. This might be nice for some users, but it's not going to seriously benefit *most users, most of the time.* And since we're getting rid of it, we can lose the tabbed interface that displays it. Since the Folders view is now the only option, it no longer needs a label or a tab.



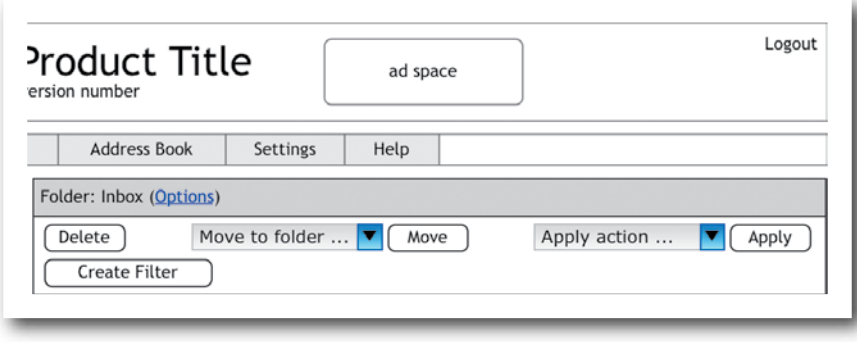
Next, let's get rid of the percentage indicator that tells users how much storage space has been used up. If we decide this is essential later, we can move it into the Settings screen. There's no reason to give it a permanent position in the main interface.



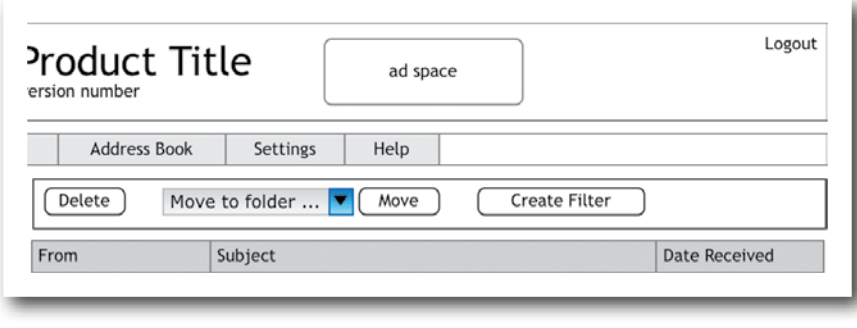
Next, let's get rid of the text that indicates which user is currently logged in. This is unnecessary most of the time, because most users will only ever have a single account, and since they have to manually log themselves in before they can see this screen, it's pointless to show them something they already know.

Also, let's kill the option to change how many messages display in the list at once. This can certainly be retained as a feature, but it's not the kind of thing users are going to use every day, so we can move it to the Settings screen.

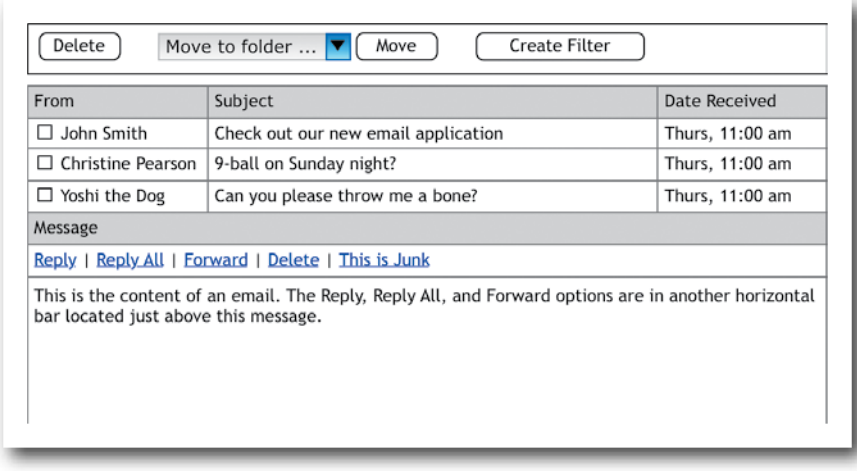
And since a Search bar is provided in the left-hand sidebar, we can remove the Search link from the top of the page.



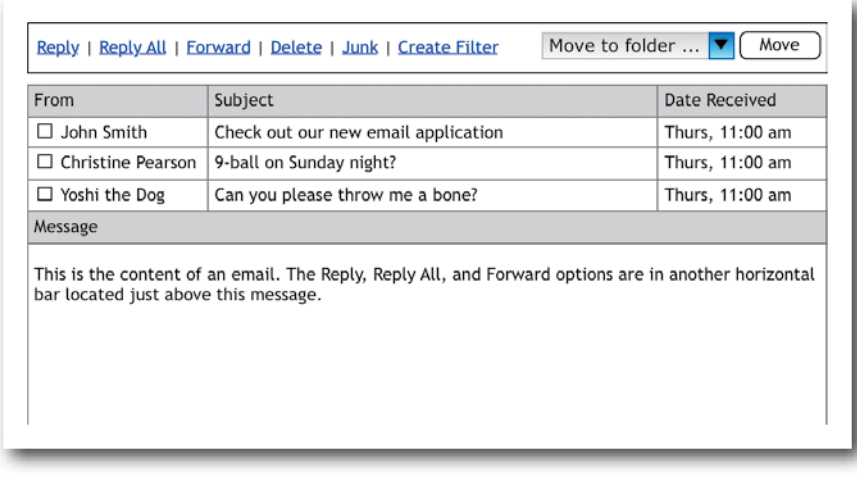
Showing a title bar for which folder is currently being displayed is redundant, because the label for the folder in the sidebar is made larger and bold when that folder is displayed. And if we remove the Folder title bar, we can free up some vertical space for more important content—like *mail.*



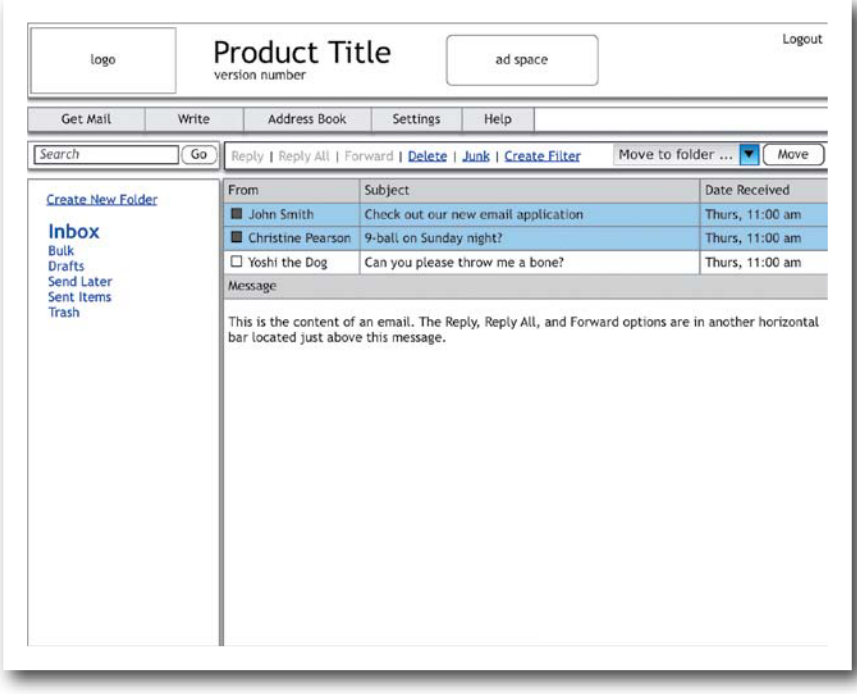
When an email is being displayed, another small bar appears above the email offering Reply, Reply All, Forward, and Delete functions, as well as a way to mark an email as junk.



But there's already a Delete button in the bar above the message list. If we remove the button and tidy things up a bit, we can consolidate the bar and unify the message options into a single interface element, which means less code, less interface, and less confusion.



Finally, let's add some logic to the application and have it disable the Reply, Reply All, and Forward links if more than one message is selected at a time. Delete, Junk, and Create Filter can all be applied to multiple messages, so we'll leave those active. In doing this, we make the message options more functional while still taking up less space.



Ahh, that's much better. We stripped out a few features, removed a few interface elements, cleaned things up, and came out with an application interface that is easier to understand at a quick glance and easier to use on a daily basis.

We'll perform interface surgery throughout this book as a way of improving applications one step at a time.

Reevaluate nice-to-have features later

So, when is it time to take the list of nice-to-have features back out of the filing cabinet? The simple answer is this: not one second before your application has been released.

Once your application is out there, being used by real users, and you've given it some time to stabilize by fixing a lot of the immediate bugs that have inevitably come up since the release, then it's time to review the list of nice-to-haves. It's also time for a good laugh.

What usually happens is that users start to speak up about what they wish your application did, things that bother them, and so on, and no one ever mentions the items on your list of nice-to-haves. Users very quickly form different perspectives on your application than you may have ever had, and since none of them use the application exactly the way you thought they would, the complaints and wish lists that emerge are usually different than what you thought was important.

If this is the case for you, feel free to put that list of nice-to-haves into the *other* filing cabinet—the one shaped like a trash can—and call it a day. The things we often think are so important at the beginning of a project usually prove to be about as useful as adding another color to a logo. And more often than not, adding them way back when would have meant putting the rock star features at risk by making them harder to find, harder to configure, *harder to use*.

Let them speak

Once your application is being used out in the wild and you want to hear all the little screaming voices of your users, you need to give them a way to talk to you. This means providing a way for users to offer feedback about your product or talk to others about it, getting out of the way so they can speak freely while you take notes and carefully interpret.

Something as simple as setting up a forum on your site and directing people there from your Support page can dramatically lower your customer-support costs (a forum costs extremely little to maintain), while greatly increasing the amount of information you get from customers.

Note, however, that you will probably not like everything that gets posted. Invariably, there will be some dissatisfied and possibly rude users who scream about your "horrible" application and say nothing constructive, but you have to let this happen. If you moderate user comments to filter out the negative, you'll defeat the purpose of the forum, which is to hear the complaints. The goal is to feel the pain.

When you allow your users to speak up, you'll quickly come up with a whole new list of nice-to-haves. Put those in the filing cabinet as well.

Avoid bending to users' whims if the high-demand features don't fit into your grand vision for the application. You might try pooling a few beta users together and have them try out a prototype of the proposed functionality to see how it really works before unleashing it to all your customers. There's no shame in pulling the feature back out if it just doesn't work. Better now than later.

Focus only on the features that are the most essential. *Build only what is absolutely necessary.*

Index

B

C

D

E

F

G

H

I

J

K

L

Q

R

S

T

U

V

W

X

Y

Z