



DESIGNING FOR THE DIGITAL AGE

HOW TO CREATE HUMAN-CENTERED
PRODUCTS AND SERVICES

KIM GOODWIN

Sample
Chapter

Designing for the Digital Age: How to Create Human-Centered Products and Services

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2009 by Kim Goodwin

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-22910-1

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data is available from the publisher.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993, or fax (317) 572-4002.

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc., and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

DESIGNING FOR THE DIGITAL AGE
**How to Create Human-Centered
Products and Services**

Kim Goodwin



Wiley Publishing, Inc.

A Note From the Author

Dear Friend,

What you're about to read is an excerpt from my new book, *Designing for the Digital Age*. It's a book not just about interaction design, but about how interaction design, visual design, and (sometimes) industrial design combine to make great products and services. It's a book about design research and design strategy: How do we design the right thing, and for whom are we designing it? It's also about design execution: How do we create designs that balance practical usability, intangible desirability, and economic feasibility? Finally, it's about communication and collaboration: How do we persuade business people that the design is right, and how do we work with engineers to make sure it gets built? Oh...and how do we do all that on a schedule?

This book attempts to answer these questions based not just on my own experience, but also on what I've seen and heard from dozens of Cooper designers, hundreds of clients, and many hundreds of people in my classes. This book is not simply a recitation of how we practice design at Cooper, though it draws heavily from that; replicating our practice in every detail probably won't meet your needs. Instead, I've tried to describe how you can expand or contract, supplement or subtract from a set of core techniques that have helped Cooper designers create successful consumer kiosks, assisted surgery systems, touchscreen phones, Web sites, complex analytical tools...even services and organizations.

I hope this excerpt gives you a taste not just of the book's content—all 768 pages and 300 or so illustrations—but also of the book as a beautifully designed product in itself. If you like what you see, you can pre-order the real thing at Amazon and other retailers. It should be widely available by March 3rd. In the meantime, look for updates on www.cooper.com.

Happy reading!

A handwritten signature in black ink, appearing to read "Tim Cook", with a long horizontal flourish extending to the right.

Contents at a Glance

1	Goal-Directed Product and Service Design	3
2	Assembling the Team	15
3	Project Planning	35
4	Research Fundamentals	51
5	Understanding the Business	65
6	Planning User Research	85
7	Understanding Potential Users and Customers	113
8	Example Interview	155
9	Other Sources of Information and Inspiration	183
10	Making Sense of Your Data: Modeling	201
11	Personas	229
12	Defining Requirements	299
13	Putting It All Together: The User and Domain Analysis	351
14	Framework Definition: Visualizing Solutions	377
15	Principles and Patterns for Framework Design	405
16	Designing the Form Factor and Interaction Framework	425
17	Principles and Patterns in Design Language	479
18	Developing the Design Language	497
19	Communicating the Framework and Design Language	515
20	Detailed Design: Making Your Ideas Real	551
21	Detailed Design Principles and Patterns	571
22	Detailed Design Process and Practices	605
23	Evaluating Your Design	649
24	Communicating Detailed Design	659
25	Supporting Implementation and Launch	685
26	Improving Design Capabilities in Individuals and Organizations	693
Index		710

 **GettyGuide**
discover more



Introduction

You've probably picked up this book because you are a designer, whether by profession or by inclination. Design is, arguably, something that every person in the world does—laying out the text in a school report, decorating a living room, and arranging plants in a garden are all acts of creation that can have both utilitarian and aesthetic value. However, most such acts consider a small set of idiosyncratic needs: the habits and preferences of an individual, or perhaps of the handful of individuals who make up a household.

Design as a *profession*—by which I mean everything from product design to architecture—exists to provide both utilitarian and aesthetic value on a large scale. Professional designers must define financially viable products, services, and environments that meet the practical, physical, cognitive, and emotional needs of a wide range of people. Like someone deciding what color to paint the living room, a professional designer can—and, to some extent, does—try something, decide that it doesn't work, and try something else. Yet designers must try, fail, and eventually succeed on a deadline, within a budget, and over and over again. Eventually, all experienced designers develop a set of implicit or explicit techniques to help them do just that, and to do it better and faster over time. This book aims to share a set of explicit process and practices that have worked for many designers over the course of hundreds of diverse projects; in other words, a *method*. An effective method, along with appropriate training and aptitude, is what distinguishes professional designers from anyone else who may perform individual, instinctive acts of design.

Why an Explicit Method?

This book offers an explicit, start-to-finish method for defining and designing the form and behavior of processes, services, and artifacts in our increasingly complex digital age. Some designers are hungry for an explicit method, while others may bristle at the thought, expecting that it will limit their creativity. However, there's nothing inherently good about chaotic or ad hoc approaches. The method described in these pages is not intended as a set of constraints or as a recipe to be unthinkingly followed in every situation; no method should be followed by rote.

Instead, think of the method as something akin to the harness and wire used in martial arts movies: simultaneously providing support, safety, and a powerful boost, but useless without the skill, creativity, and judgment of the practitioner. Or if that analogy doesn't work for

Professional designers must define financially viable products, services, and environments that meet the practical, physical, cognitive, and emotional needs of a wide range of people.

**Certainly,
good design
can happen
without an
explicit method.
However, in
the words of
Louis Pasteur,
“Fortune favors
the prepared
mind.”**

you, how about this one: the designer’s creative spark is the electricity, and the method is the power grid that channels it where it can do the most good.

Why does a designer’s creative spark need to be channeled? Certainly, good design can happen without an explicit method. However, in the words of Louis Pasteur, “Fortune favors the prepared mind.” Without the scientific method to structure his thinking, an accident with a spoiled culture would not have led him to the germ theory of disease (and yet the method alone didn’t do the trick).

Design and science have something else in common: in each field, ideas are be subject to examination and judgment by others. If you have a method that explains how you got from point A to point B, people are more likely to judge in your favor than if you say, “Trust me—I’m a professional.” I expect you’ll find the methods in these pages useful if you’ve ever:

- Had to argue with a powerful CEO about why his personal preferences shouldn’t drive the design
- Been uncertain whether design option A or B is better
- Had a group of hard-core engineers smell blood in the water when you used “because it looks cool” as a defense
- Had stakeholders repeatedly change their minds about what the product is
- Needed to convince stakeholders that no, really, people don’t use your product that way
- Had a design meeting that resembled a rugby match
- Come up with a cool design concept that turned out to be unworkable a few weeks later
- Wondered how you could possibly learn enough about neurosurgery, stock portfolio management, or chemistry to design a product around it
- Had your design bomb a usability test
- Stared at a blank whiteboard, uncertain where to begin

Both as a consultant and as an in-house creative director, I’ve been in most of these situations, and I’ve observed other designers struggle with these and other challenges. An effective method removes much of the worry in these situations so you can instead focus on doing what designers do best: generating usable, desirable solutions.

Of course, no method is perfect, and no method should be engraved in stone. The methods in this book have evolved over the years and will continue to do so as designers try new things and share the successful ones as best practices—one reason I’ll be sharing my latest experiences and resources (including materials to use for some of the exercises) at www.designingforthedigitalage.com; I hope you’ll share your own experiences, too. However, I’ll offer you the same suggestion I share with new hires at Cooper: try the techniques as described over the course of several projects so you can master them before you carve a new trail through the underbrush. You’ll probably find that the core methods address a wider variety of situations than you expect and afford all the flexibility you could need.

Why This Book

Every designer has the power to improve or even preserve life for some segment of humanity. Unfortunately, even the best designers can’t design everything, and good designers are in limited supply. I also know plenty of potentially great designers who simply don’t have the tools they need to make sure their designs see the light of day. This is especially true in our current digital age, when many design problems require the application of multiple disciplines, including interaction design, visual and information design, information architecture, industrial design, and more. Users have only one experience of a product or service, though, so this book attempts to include the perspectives and activities of all of these disciplines. (However, given that industrial design and graphic design make use of long-standing, well-understood methods, I have not attempted to address those disciplines in the broad sense, but only as they relate to interactive products and services.)

Although I love the ability to influence lives through doing meaningful design, I learned long ago that I can influence even more lives by helping other designers be more effective. My aim with this book is to help as many designers as possible make a difference in the world. Because designers cover a wide range of experience and skills, experienced designers may find that some parts of the content (particularly Chapters 15, 17, and 21) are merely useful refreshers. However, each chapter of the book includes content that I hope will:

- Help experienced designers be both rigorous and persuasive in their practice, to ensure not only that they’re doing great design, but that their design gets built
- Give designers from different disciplines a shared framework for collaborating on today’s increasingly complex products, which often combine software, hardware, services, and environments
- Help design students understand not only a coherent design process, but also the essential practices—from collaboration and project management to leading stakeholder discussions—that make real projects successful
- Show consulting designers how to engage with clients for the long term
- Help in-house designers see how consulting practices can make them more effective

Design is not—and never will be—a science. It will also never be a cookie-cutter process that anyone can do with an appropriate checklist in hand—the method doesn’t make the design, the designer does. This book cannot give you the imagination and aptitude for visualization, nor can it give you the judgment and mastery of craft that only come with experience. However, I hope what you’ll take from this book will help you more reliably design the right product or service, design it well, and get the design out into the world where it can improve the quality of human lives.



CHAPTER 16

Designing the Form Factor and Interaction Framework

Any effective approach to design starts with defining the big ideas, iterating until you get them right, then adding more detail and iterating until the product or service is ready for people to use. The fundamentals of this process look very similar whether you're designing hardware, software, services, or environments: Start with the best idea you can think of based on your understanding of users, examine it in various ways to find problems, then throw it out if it's hopeless or refine it if it's close to working.

However, unlike other design problems, interactive products and services change state over time and engage in a type of conversation with their users. These factors are what make personas and scenarios so useful as design tools—personas help you predict user behavior and desires in any circumstance, while scenarios help you see how the product should change state over time. Of course, this early part of the design work also takes considerable visualization skill, an ability to think in systematic terms, and a deep knowledge of design principles and patterns to come up with a solution that's not just workable, but *desirable*.

The approach outlined in this chapter focuses on how to visualize form and behavior: what the product or service is, what it does, and how it looks and works from a user's point of view. Design processes are never as neat and linear as they look in diagrams—there's always some variation due to the design problem and the working styles of individual team members. However, there are certain thought processes that need to happen (whether explicitly or in a designer's head) for good design to emerge. Some of them are usually best done earlier than others, as shown in Figure 16.1; the process is recursive but has an overall flow. The process for designing behavior is much the same regardless of whether there is hardware design involved; simply ignore any boxes in the diagram that don't apply. Sections in this chapter describe how the process applies to specialized design problems.

If you're designing a service, the first step is to figure out what “products,” environments, or modes of interaction comprise the service, then design each as you would an individual product. The same is true for designing product lines.

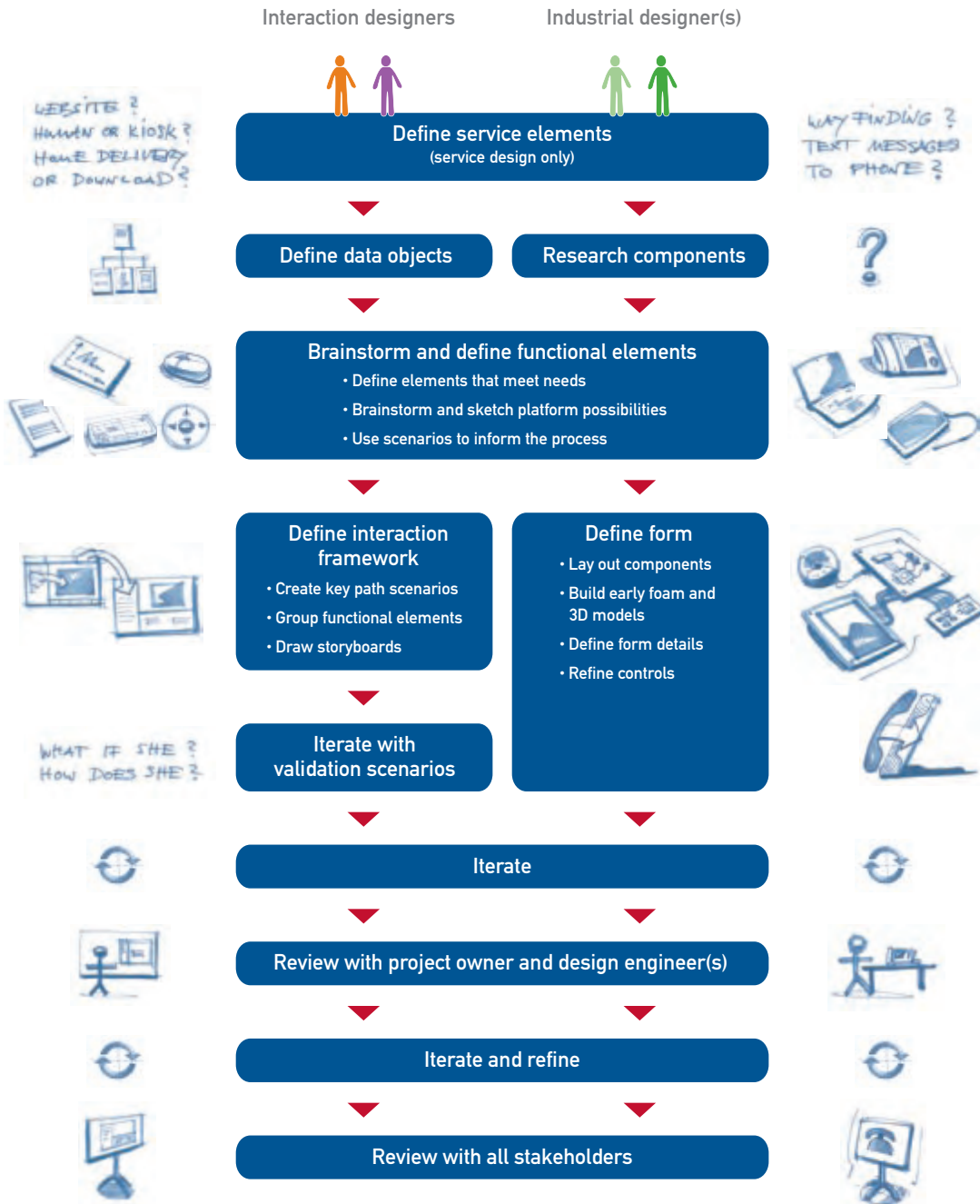


Figure 16.1. Overview of form and behavior definition process.

For each product (or service component), first focus on the puzzle pieces appropriate to each discipline. Interaction designers (IxDG and IxDS) must clearly define the data objects, their attributes, and their relationships. At the same time, industrial designers must understand the various components (such as batteries and circuit boards) that are likely to be required, as well as any physical or engineering constraints on their layout; these factors will influence the volume and shape of the device to some extent. Industrial designers generally collaborate early on with mechanical engineers to determine what flexibility there is. Once the members of each subteam are satisfied that they understand the issues, they should come together (along with the visual designer) to begin developing solutions.

Early solution development is both analytical and generative. Some teams find it useful to begin with the analytical by translating functional needs into specific solution components, while others prefer to begin in a more generative fashion, brainstorming a wide array of solutions and then using an analytical approach to determine which possibilities will be most fruitful. The analytical starting point is helpful when time or constraints are tight, since it helps eliminate possibilities very quickly, or when the system is very complex. It can also be useful if the team is having difficulty being generative, though this is not usually a problem with experienced designers in the room. In reality, although I've explained them as separate tasks, most teams bounce back and forth between the generative and the analytical, exploring possibilities and ruling them out as they go.

Whichever starting point you choose, both activities need to happen. The whole design team needs to begin making some decisions about how to address functional needs; many are purely design decisions with little business impact, while a few require input from other product team members or stakeholders. The whole team also needs to spend some time (whether it's an hour in a

tightly constrained project or a week for a more ambitious one) doing broad exploration.

When designing a device or a service that involves a physical environment, the whole design team sketches possible solutions together, then narrows these to the best candidates. Members of each discipline iterate these concepts for a day or two, with interaction designers using scenarios to evolve the behavior while the industrial designer(s) evolve the form based on ergonomics, engineering considerations, and other factors. The visual designer(s), in the meantime, begin working on visual language studies (as discussed in Chapter 18). How early the industrial designer(s) incorporate the design language is largely a matter of personal preference, though it's usually not worth focusing on this until the basic form is settled.

What one subteam learns during this independent iteration affects the work of the others, so the whole design team checks in anywhere from daily to twice a week, depending on how interdependent the software and hardware are and how much each team member tends to stray during exploration. Members of each discipline may have somewhat differing points of view about the design approach. This tension is usually productive as long as each sees the value in the other's concerns. Personas can help resolve most differences; the team lead may need to do so if there is an impasse, but this should be a rare event in an effective team.

It's important to review work in progress with the project owner and design engineer(s) as soon as one or more good directions emerge, usually about four or five days into the design. This helps avoid wasting time on solutions that aren't feasible or viable from a business perspective. Good design engineers can often help improve the design, as well.

The remainder of this chapter outlines each aspect of this process in more detail. Because the

The kinds of
“objects” users
think about help
shape the design.

fundamental ways that industrial designers work are well understood in that profession, I won't go into great detail on the industrial designer's tasks, but will focus primarily on how the interaction design develops, how form and behavior drive one another to achieve a coherent solution, and how personas and scenarios affect form.

IxDG and IxDS: Define Data Object Types and Relationships

Just as it's important to know what sorts of things will be stored in a kitchen or carried in a vehicle, you need to determine what kinds of objects the personas would use in your product. Although what constitutes an “object” in a user's mind isn't always an “object” in a programmer's mind, what you define at this point is the foundation of the **data model**; engineers may make a few adjustments to your version so it fits their needs, but they'll find it tremendously helpful.

A **data object type**¹ is a species of thing a user can create, manipulate, or look for, such as a file on a hard drive, a person in a contact list, or a store in a shopping mall directory. A data object is usually something on a screen, but could also be something physical (such as an airplane boarding pass) or aural (such as a telephone call). In almost all cases, an object type is something there can be multiple instances of.

Not all of the information needs you identified as requirements (see Chapter 12) will become data objects. For example, consider the data needs of Katie, our photographer persona, which are outlined in Table 12.1. She needs to understand how many shots will fit on her camera's memory card, whether she has enough battery power remaining to take those shots, and what the exposure settings are. These are things she has to see in some kind of information display, but they're not object types because she can't create them or move them around. The only data object type in her camera interface is a photo. In the photo management application on her computer, though, there are multiple object types, including photos, groups of photos, and perhaps master photos with adjusted copies.

In an ideal world, the data object types would come directly from your personas' mental models. This is possible in some simple applications, though in more complex cases it's often necessary to introduce new object types that don't exist in the mental model. Profiles are a

1. It would be more precise to call it an “interface data object type” to distinguish it from a programming object, but the term is cumbersome.

good example of an introduced object type. They don't exist in most users' mental models, but clustering a complex group of network or data analysis settings into a single profile (such as "work" or "home") can help users make time-consuming changes with a single click. You may also find that there's a mental model analogue that needs to be translated; before the advent of digital music management, for instance, you wouldn't have found a mental model object called a playlist, but anyone who was a teenager in the era of cassette tapes probably had some awareness of individual songs collected into "mix tapes." It's perfectly acceptable to introduce new types of data objects if you have a good reason for doing so, provided you're building on the mental model rather than breaking it.

Physical-world metaphor is both useful and limiting in this respect. Apple's iPhoto, for example, relies on albums as the data object representing groups of photos. This is consistent with most users' mental models. However, Apple then introduced another type of data object called a folder, which exists solely to organize albums (see Figure 16.2). In the real world, though, albums don't fit in folders. Also, unlike folders in the operating system, iPhoto folders cannot contain individual photos; they can only contain albums. Both of these points are contrary to expectation, which makes the data model less easy to grasp than it could be. It's relatively easy to overcome in a simple application like iPhoto even if it's a bit like a door hinge that squeaks every time you use it. In more complex applications, though, this kind of dissonance can create significant usability problems.

Avoid getting overly abstract in your definition of object types, though. It might seem elegant to say that both patients and doctors should be tracked in a hospital system as "people," but users need different information for each and definitely don't think about them in the same terms.

Object types alone don't paint a complete picture, however. Your initial draft of the data model should include the following for each type of object:

- What it's called
- What it is
- How it can be related to other object types
- What users can do with it
- What states it can be in
- What attributes it has that your personas care about



Figure 16.2. Apple's iPhoto arranges photos in albums, which makes sense, but who stores albums in folders?

Don't be concerned if you can't articulate all the attributes, states, and actions yet. The objects, their definitions, and their relationships are most critical at this point, but you might still uncover some as you design; document templates or e-mail filters,

for example, might not occur to you as data objects until you work through less-common scenarios. Include any future objects you anticipate, even if they won't be part of the first release. Table 16.1 shows an example of a preliminary data model.

Table 16.1. Preliminary data model for a veterinary practice management application.

Object	Definition	Relationships	States	Actions	Attributes
Client	A human or organization whose pet has been seen (or has an appointment to be seen) by a veterinarian or technician	<ul style="list-style-type: none"> – Always contains one or more pets – Usually contains one or more bills 	<ul style="list-style-type: none"> – Incomplete (record started but missing information) – Active (current client) – Archived (manually archived or has not visited in some number of years) 	<ul style="list-style-type: none"> – Create – Edit – Read – Archive – Find 	<ul style="list-style-type: none"> – Name – Address – Phone numbers – E-mail – Account standing – Billing history
Pet or patient	An animal who has been seen (or has an appointment to be seen) by a veterinarian or technician	<ul style="list-style-type: none"> – Always part of a client file – May contain one or more visits 	<ul style="list-style-type: none"> – Incomplete (record started but missing information) – Active (current client) – In need of follow-up – Archived (manually archived or has not visited in some number of years) – Deceased 	<ul style="list-style-type: none"> – Create – Edit – Read – Find 	<ul style="list-style-type: none"> – Name – Species – Breed – Age – Sex – Color – Temperament – Health history – Medications – Allergies

Object	Definition	Relationships	States	Actions	Attributes
Visit	An instance of a pet being seen	<ul style="list-style-type: none"> – Always part of a client file – Usually associated with a bill 	<ul style="list-style-type: none"> – Incomplete (record started but missing information) – Complete 	<ul style="list-style-type: none"> – Create – Edit – Read 	<ul style="list-style-type: none"> – Notes – Vital signs – Procedures performed – Tests ordered – Diagnoses – Prescriptions – Supplies used – Amount billed – Follow-up required
Appointment	A time scheduled for a visit with one or more pets and a veterinarian or technician	Always associated with a client, one or more pets, and a veterinarian or technician	<ul style="list-style-type: none"> – Incomplete (record started but missing information) – Complete 	<ul style="list-style-type: none"> – Create – Edit – Read – Delete 	<ul style="list-style-type: none"> – Client – Pet(s) – Veterinarian or technician – Date – Time – Room – Equipment – Procedure or appointment type
Bill	A statement of the amount due for one or more visits	Always part of a client file	<ul style="list-style-type: none"> – Incomplete (record started but missing information) – Unsent – Awaiting payment – Overdue – Paid 	<ul style="list-style-type: none"> – Create – Edit – Read – Mark paid 	<ul style="list-style-type: none"> – Client – Pet(s) – Veterinarian or technician – Date – Time – Procedure or appointment type – Supplies used – Amount owed

Continued

Object	Definition	Relationships	States	Actions	Attributes
Veterinarian	A doctor of veterinary medicine who treats pets at this facility	Associated with multiple clients, pets, visits, bills, and appointments	<ul style="list-style-type: none"> – Active – Inactive (no longer seeing patients at this hospital) – Available – Unavailable 	<ul style="list-style-type: none"> – Create – Edit – Make inactive 	<ul style="list-style-type: none"> – Name – Availability – Procedures and appointment types they do or don't do
Technician	A non-veterinarian who assists veterinarians, gathers information, and performs some routine treatments	Associated with multiple clients, pets, visits, bills, and appointments	<ul style="list-style-type: none"> – Active – Inactive (no longer seeing patients at this hospital) 	<ul style="list-style-type: none"> – Create – Edit – Make inactive 	<ul style="list-style-type: none"> – Name – Availability – Procedures and appointment types they do or don't do
Room	A location that can be reserved for a visit	Associated with certain procedures and appointments	<ul style="list-style-type: none"> – Available – Unavailable – Inactive (permanently unavailable) 	<ul style="list-style-type: none"> – Create – Edit – Make inactive 	<ul style="list-style-type: none"> – Name – Availability – Equipment – Associated procedures
Cage	A pet holding area that can be reserved for boarding or ongoing care	Associated with certain appointments	<ul style="list-style-type: none"> – Available – Unavailable – Inactive (permanently unavailable) 	<ul style="list-style-type: none"> – Create – Edit – Make inactive 	<ul style="list-style-type: none"> – Name – Availability – Size or appropriateness for certain animals
Piece of equipment	An in-demand medical device that can be reserved for a visit	<ul style="list-style-type: none"> – Associated with certain appointments – May be part of a room 	<ul style="list-style-type: none"> – Available – Unavailable – Assigned to a room – Inactive (permanently unavailable) 	<ul style="list-style-type: none"> – Create – Edit – Assign – Revoke assignment – Make inactive 	<ul style="list-style-type: none"> – Name – Availability – Room – Associated procedures

Explicit **nomenclature** and object type **definitions** are important even for seemingly obvious terms; if one person thinks of a “photo album” as a simple collection of images with no explicit organization

while another thinks it has a user-specified page layout, miscommunication will bog down your design meetings until you straighten it out.

Relationships are also important to articulate. Like mental model objects, data objects may be related in several ways, such as:

- **Many to many:** many people may send many messages
- **Many to one:** many documents may reside in one folder
- **One to many:** one folder may contain multiple documents
- **One to one:** one company has one tax identification number
- **Hierarchical:** albums contain photos
- **Temporal:** a law is generated from, and replaces, a bill

It may not be critical at first, but sooner or later you will also need to define what **actions** (such as creating, moving, editing, and deleting) users can and can't perform upon each type of object, as well as what **states** each object can be in. Your scenarios are good guides for this. Don't worry if you can't write down a lot of detail yet, since some necessary actions and states will only become clear as you work through the design.

Finally, you need to develop some understanding of what object **attributes** your personas care about, such as the sender and subject of an e-mail message or the vendor and terms in a purchase order. These attributes usually map to database fields later on. As with actions and states, don't be too concerned about filling out every possible detail at this point; what you most need to understand is whether the personas need a lot of information about each object or only a little; this determines how much screen space you'll need to display that information. You'll continue refining the attributes as you do detailed design.

To develop the contents and structure of your data model, start with the mental models you gleaned from your interviews. As you add any new object types, check them against your personas and

scenarios. First, ask yourself if the new object type is really necessary or if you're making a fine distinction that never matters to the personas. Scenarios are also critical to generating data objects. Look for the nouns in your scenarios: lists, documents, images, or whatever else is being used, viewed, or manipulated. Not all will be data objects, so compare them to the criteria above. Finally, consider whether the object type as you've defined it will make sense to your personas.

Most data objects are easy enough to figure out, but complex software such as analytics or IT administration tools can require more careful thought. Some additional data objects might emerge as you design, but for now, ask yourself whether there are dependencies between any of the object types on your list and any others, whether any of the objects need to be grouped into higher-level objects, or whether business constraints will result in any specific objects (such as a log that provides an audit trail). This may help you define some of the trickier object types.

Exercise

Define the data object types, relationships, attributes, and valid actions and states for the LocalGuide or RoomFinder.

Full Design Team: Define Possible Functional Elements

Once you have a good idea of what the data model looks like (and a good grasp of the product's likely internal components, if hardware is involved), you're ready to start translating the functional needs identified during requirements definition into a set of possible functional elements. If you are designing a multi-environment service or a dedicated device rather than software for an existing platform, it's essential that industrial designers be involved in these initial discussions.

Functional elements are solution components that are visible to users.

Functional elements are solution components that are visible to users; they take up space on the screen or have some physical representation (such as a button or knob) on a device. In a way, making decisions about functional elements is like listing the rooms, storage areas, and staircases you'll need in a building before determining how they'll be laid out. As with every part of the design process, this is iterative: Start with a list of major elements, begin using those to draw screens or hardware configurations, and then gradually add more detail to your list as you draw. Don't get too attached to solutions until you begin sketching and trying them out, especially when you're doing broad exploration for a novel platform.

Each functional need in the requirements list (see Chapter 12) leads to a functional element that addresses the need. In many cases, these are pretty obvious; a need to play video implies a video display and either physical or virtual playback controls. Tedious as this kind of translation may seem, it's still worthwhile for a couple of reasons. One is that any designer can benefit from having an explicit list of all the puzzle pieces that have to fit on the screen or device. This is especially useful for less-experienced designers (who might otherwise miss an important element and have to start over) or those who aren't natural visual thinkers (who might otherwise be uncertain where to start on a blank whiteboard).

The other is that on nearly every project, there are one or two items in the list of needs that have major business implications, such as whether to use an inexpensive segment-based screen, which allows for very limited flexibility in what can be displayed, or a more expensive, pixel-based LCD.

Many experienced designers almost unconsciously develop a mental list of functional elements rather than an explicit one. This can work, particularly if you're stuck designing alone, but has a couple of drawbacks. The most important is that an unconscious list of elements doesn't get examined explicitly. Mental lists are also difficult for others to engage with. It's not just that your teammates won't have access to what's in your head—you're also likely to find collaboration difficult if everyone on the team is working from a slightly different mental list of functional elements.

It's not necessary at this point to identify every possible functional element in the product; all you need to do is figure out the major ones. What these are varies depending on the nature of the design problem.

Functional elements in product design

Whether you're designing a Web site, a desktop application, or a device with some sort of information display, there are generally three basic types of functional elements critical to accomplishing the scenarios:

- Display areas for data or content, such as a video or a list of email messages
- Tools or controls, such as on-screen widgets or hardware buttons that interact with screen contents
- Places to put tools and controls, such as toolbars or palettes on a screen, or control surfaces on a device

Display areas, physical navigation or input controls, and areas to put software tools are important at this point, but individual software controls (such as buttons and list boxes) are not. Trying to identify every widget would be a waste of time

because you'd unmake many of those decisions later on. Just focus on things that will require significant amounts of physical or screen space; you'll figure out the rest when you get to detailed design.

Although interaction designers are inclined to think about the parts of the hardware that are directly involved in the visual and tactile interaction, such as knobs, buttons, and displays, industrial designers must also think of functional elements that address sound, power, storage, ergonomic needs, physical connectivity with other devices, and so forth.

Table 16.2 shows a partial list of needs translated into functional elements for an office telephone, including both interactive elements and other important hardware components. Table 16.3 shows a similar partial list for an e-commerce Web site. Some people find it helpful to portray the scenario in the same table to show how they relate (see Table 16.7 later in this chapter).

Table 16.2. Functional elements for an office telephone.

Functional need	Functional element
Place calls to known contacts and colleagues	Contacts list/directory displayed on screen
Place calls to numbers not in contacts list	Keypad (physical or virtual)
Review messages	On-screen display; audio playback for remote access
Manage calls (hold, transfer)	Hold button, transfer button (physical or virtual)
Adjust tilt of keypad and display	Adjustable stand to tilt entire device
Use headset	Standard wired headset port in addition to wireless
Capture and play sound as a speakerphone	Microphone along front of device; speaker
Store handset	Cradle

Table 16.3. Functional elements for a shopping Web site.

Functional need	Functional element
Find a specific product	Search field
See what's available	Category listing
Learn about a product	Product information display area
Store items under consideration	Shopping cart

Service design usually entails interaction with multiple channels and environments—in other words, multiple “products.”

Framework

Functional elements in service design

Service design is unique in that it often entails interaction with multiple channels and environments, so it may involve the design of multiple “products.” On the simple end of the service design scale, a movie rental service includes selecting the movies you want, getting them, watching them, and then returning them. This might all occur on your computer: Pick a movie from a Web site, download it, watch it on your computer, then have your rental privilege expire. It might also involve multiple devices and channels: a Web site to select movies, a delivery service that brings you the discs, your home DVD player to watch them, and some sort of packaging and service to return them. At a minimum, you can design the Web site and the packaging; ideally, you could design the whole service from end to end.

Service design is mostly about making sure you understand every point at which you can affect a user’s experience, then finding some way to make it better. Your context scenarios should have covered each of these points; the tricky bit is in identifying where you need to design new products, where you can take advantage of existing third-party solutions, and how to work around aspects of the service you can’t change. In a sense, this is defining what “products” you need to design, then following the design process described here for each.

For example, if you were to redesign airline travel, you might start by translating your list of needs into service elements. Table 16.4 shows a partial list of example elements. Once you had identified the solution components of the service, you could then use more detailed scenarios to define the functional elements of the Web site, the cell-phone-boarding-pass interaction, and so on.

Table 16.4. Service elements for airline travel.

Functional need	Service element
Find a flight that fits time and budget needs	Web site that acts like a really good travel agent (“Did you know you could save \$200 by flying on Sunday?”)
Find the best place to stay on a business trip and the best way to get there	Web site that offers hotel and transportation options based on preferences and meeting location
Know when to leave for airport	Web site that estimates time based on route to airport, typical traffic, and typical security wait times; airline sends notice to mobile phone if anything changes
Avoid boarding pass hassle	Send electronic boarding pass to passenger’s mobile phone; allow those without phones to print from Web site
Quickly deal with unexpected situations such as a cancelled flight	Easy access to a live human in the airport or on the phone without a long wait time
Avoid checked luggage hassle	Offer unique RFID luggage tags that automatically associate the bag with passenger and flight that day; just put the bag on a conveyor belt

Making decisions

As Tables 16.2 through 16.4 imply, defining functional elements involves making both business and design decisions, which are largely about trade-offs. If there are multiple solutions that might address a need, which is the best fit for the personas, scenarios, and goals? Which requires the least effort (both mental and physical) on the part of users? Which is easier to build? Which might generate additional revenue? Which solution is more economically, socially, and environmentally sustainable?

Consider voicemail for an office telephone as an example. When someone checks messages, you could provide an audible list of new messages or you could display a list of calls on a screen. First,

consider which is better from a design perspective by thinking through scenarios using each possible solution. If your persona has several messages when she returns to her desk, a visual display has the benefit of letting her quickly choose whichever message is likely to be most important to her at the moment. When she has to check voicemail while out of the office, audio playback may be the only option if you don’t also control her mobile-phone interface. This probably means you have to have audio playback, but there’s a good design argument for also providing a visual solution. The visual solution will require more than the two-line display that’s common to office telephones, though, as well as additional coding time, so you’d need to involve the appropriate stakeholders to decide whether the gain in revenue potential is worth the cost.

Ideally, there is time to explore multiple platforms and architectures for physical products.

If a design choice mostly affects usability and desirability without major business implications, then it's generally best to make that decision within the design team. Of course, you won't always be entirely correct about which issues do or don't have a business impact; this is one good reason for frequent check-ins. Some of your decisions may be subject to change as you iterate the design, but it's usually most effective to make the best decision you can and move on, rather than researching every possible implication of every choice.

Exercise

1. For the LocalGuide: Define functional elements for the end-to-end service, then for the device itself.
2. For the RoomFinder: Determine whether it's a single interface on a single platform or a multiplatform system and then define functional elements accordingly.

Full Design Team: Define Possible Platforms

For some design projects, such as constrained redesigns, the range of possible solutions is narrow—there is no doubt about whether you're designing a Web app, a handheld device with a touch screen, or an inexpensive kiosk with membrane buttons and a segment-based display. In other cases, the possibilities are much broader, so the design team's job is to help the product team think creatively and explore multiple **platforms**. A platform is defined by its:

- **Form factor:** If it's a device, is it a semiportable tablet, a tiny handheld that fits in a pocket, or a massive console with a dedicated room in a hospital? What size and orientation is the screen, if any, and what technology does it use (such as segments, pixels, or electronic paper)?
- **Input and output methods:** By what mechanisms does a user navigate, select, consume, and enter data? Is it a touch screen, a physical keyboard, voice input, or some technology no one has ever seen before?

Ideally, the design team has time to explore several platforms, since each will have different advantages and disadvantages regarding usability, cost, revenue potential, and future flexibility. However, the differences among platforms can have a tremendous impact on the behavior of a device, so the interaction designers usually need to walk through a set of scenarios for each one. This makes multiplatform exploration more

time-consuming than, say, developing multiple physical appearance concepts for the same basic phone or looking at several different structures for a Web site. For this reason, it's usually necessary to narrow the platform options to two or three in fairly short order, develop some sketches and quick foam models, and focus on one platform as soon as stakeholders are confident. It's a rare organization that will fund more than one platform possibility through detailed interaction design.

However, the industrial and interaction designers may still explore multiple **architectures**—arrangements of essentially the same components—for each platform. Figure 16.3 illustrates three different platforms: a touch screen, a version with a four-way controller, and an option with soft buttons (physical controls around the edge of the screen that have varying functions identified by on-screen labels). It also shows a rendering of two different architectures for the platform with the four-way controller; the control's physical placement (and hence the overall shape of the device) differs, but the input method and the orientation of the screen are the same, so the interaction with either device will be essentially the same. As a result of this approach, industrial designers may show stakeholders a handful of different architecture sketches for each platform, such as those in Figure 16.4.

Interaction designers, on the other hand, generate plenty of ideas at the whiteboard, but they exercise each possibility using scenarios, resulting in fewer options (often only one option per platform) being visible to stakeholders. It's not that interaction designers are less generative or open to exploration, but that interaction design simply cannot be evaluated with a single sketch. A recent client is a useful example; the company was working with us on interaction design for a

Interaction design cannot be evaluated with a single sketch, so exploring multiple concepts takes a fair amount of time.

Three different platforms



Two architectures for one platform



Figure 16.3. Platforms versus architectures.

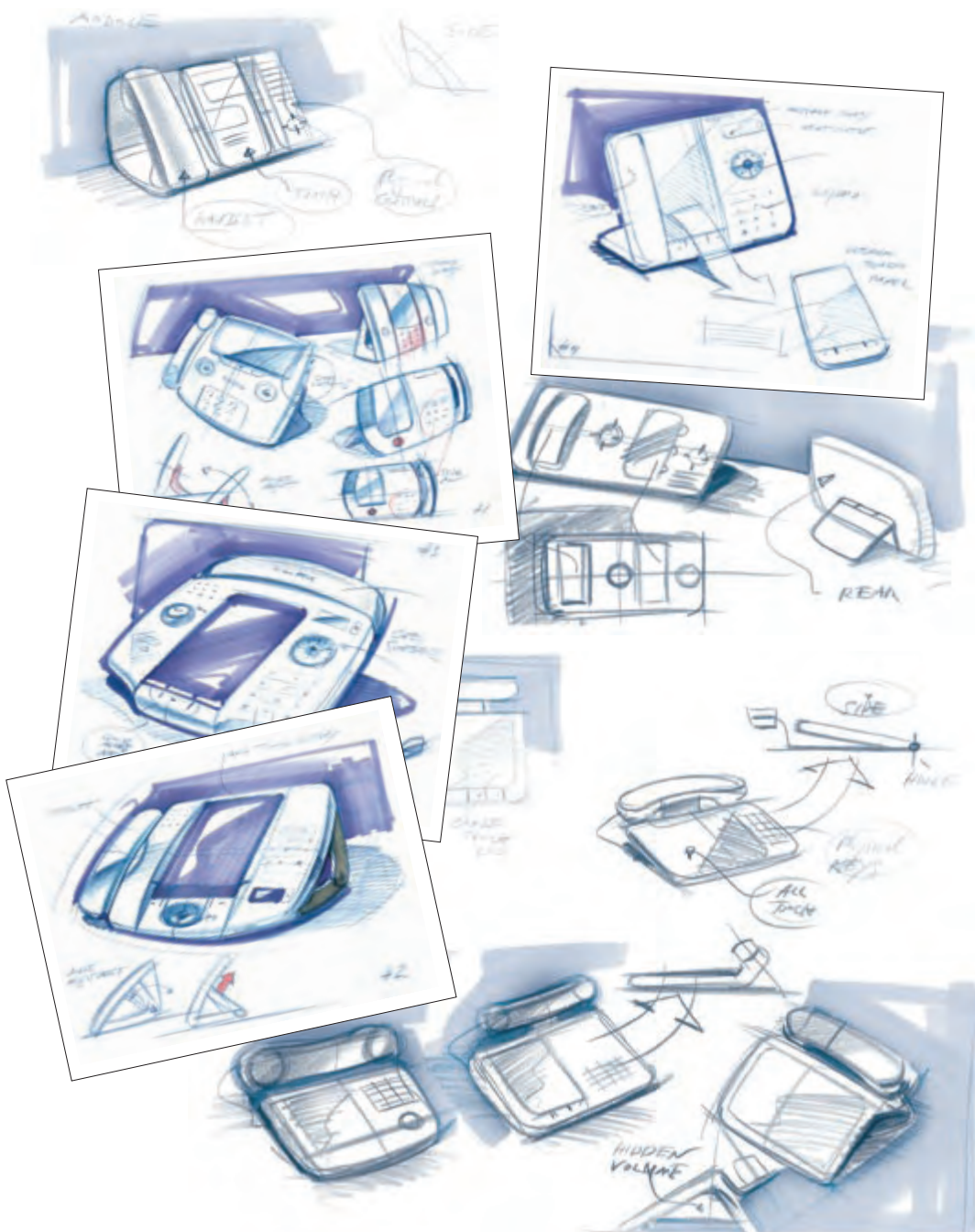


Figure 16.4. A range of architecture and platform sketches.

new platform, but had hired a separate firm to do visual design, hoping to generate some goodness through using firms with two different approaches. Naturally, that firm's visual designers were interested in interaction just as our interaction designers were interested in the visual expression of the behavior, so their team threw out a few interaction design concepts. Stakeholders thought one of the ideas looked pretty slick, fell in love with it, and asked the interaction designers to incorporate it. Unfortunately, the concept had never been vetted with a scenario, so it really didn't work very well, leaving the stakeholders frustrated.

Input and output methods

For devices, input and output methods play a large part in determining form factor. The most important considerations usually involve how much text your personas will enter, how direct the manipulation must be, and what your COGS (cost

of goods sold) budget is. The usage environment and scenarios should help you decide whether visual or audible output is best and whether input should be optimized for voice, one hand, two hands, or even feet.

Most interactive products use visual output, which is appropriate in most circumstances (at least for sighted users) because it's a richer communication medium and because it's quiet and private. Audible output is common in customer service systems, which makes sense because it shouldn't require a computer to tell your utility company that your power is out. Audible output is also useful when users really need to look somewhere other than at a screen, such as when they're driving.

The input options are more complex. Table 16.5 describes advantages and disadvantages of some common input methods.

Table 16.5. Advantages and disadvantages of various input methods.

Method	Advantages	Disadvantages
Touch screens	<ul style="list-style-type: none"> – Allow direct manipulation – Flexible display of most appropriate options in any context – Gestures offer useful shortcuts for common tasks 	<ul style="list-style-type: none"> – Expensive – Can get smeared – Not so good for text input – Add weight and thickness – Require large controls unless you use a stylus
Soft buttons (on-screen labels used to indicate what a nearby physical control will do)	<ul style="list-style-type: none"> – Usually cheaper and lighter than touch screen – Allow fairly direct manipulation; little cognitive effort if button and label are clearly associated – Somewhat flexible display of most appropriate options in any context 	<ul style="list-style-type: none"> – Less flexible than touch screen – Gets very clumsy if you have more options than buttons

Continued

Method	Advantages	Disadvantages
Cursor-based selection with up-down and/or left-right buttons plus a selection button (as in a television remote control or on many phones)	<ul style="list-style-type: none"> – Can be relatively fast, especially if you accelerate scrolling speed when the button is held down – Less expensive than touch screens – Familiar idiom for most users 	<ul style="list-style-type: none"> – Slower than touch screen or soft buttons if you're trying to choose among just a few options – Less direct than touch screen or soft buttons; requires more cognitive effort – Repetitive-motion strain if you require individual button presses instead of accelerating
Cursor-based selection with a mouse, scroll wheel, or trackball	<ul style="list-style-type: none"> – Faster than 2-way or 4-way controller for selecting among a few options – Generally less expensive than touch screens – Familiar idiom for computer users 	<ul style="list-style-type: none"> – Requires fine motor control – Slower than touch screen or soft buttons if you're trying to choose among just a few options – Less direct than touch screen or soft buttons; requires more cognitive effort
Gyroscopes and accelerometers	<ul style="list-style-type: none"> – Allow gesture-based input to control orientation, speed, etc. 	<ul style="list-style-type: none"> – Imprecise – Unfamiliar idiom for many users
Knobs and dials	<ul style="list-style-type: none"> – Can allow very precise control depending on engineering – Allow quick access if dedicated to a single function – Jog dials (which return to a fixed position) can allow for fast scrolling 	<ul style="list-style-type: none"> – Protrude more than other controls – Cost varies depending on characteristics – Mapping of clockwise and counterclockwise rotation to vertical or horizontal movement may have to be learned
Stylus on touch screen or separate input pad	<ul style="list-style-type: none"> – Great for drawing and handwriting input – Minimize fingerprints and smearing on touch screens 	<ul style="list-style-type: none"> – Easily lost; people may use pens and pencils instead – Writing can be slower than typing – Handwriting recognition is usually imperfect
Voice	<ul style="list-style-type: none"> – Hands-free input is safer in some circumstances – No repetitive-motion injuries 	<ul style="list-style-type: none"> – Imprecise except for a limited set of commands – Inappropriate in noisy or open work areas or for confidential information

Method	Advantages	Disadvantages
Keyboards, number pads, and other physical buttons	<ul style="list-style-type: none"> — Fast text entry — Familiar idiom 	<ul style="list-style-type: none"> — Large — Repetitive-motion injuries are common with keyboards — Buttons covered in membrane can be too stiff and membranes can puncture
Foot pedals	<ul style="list-style-type: none"> — Appropriate for some applications when both hands are occupied, such as performing surgery or playing a guitar 	<ul style="list-style-type: none"> — Awkward in most situations

Input methods for desktop applications or any other software running on an existing platform are largely predetermined, though you don't always have to stick with a keyboard and mouse; voice, pen, or other input options may be feasible for specialized activities.

Other form factor considerations

Context plays a major role in determining form factor. The more portable the device needs to be, for example, the smaller users will generally want it. Internal components also have a great deal of influence—if there's no cost-effective way to make them smaller, then the device must be at least a certain size or shape.

The other big driver of form factor is the size and orientation of the screen, which should be guided by the type of content you need to display. Any device that largely involves lists of selectable items, such as songs on a music player or medications on a hospital infusion pump, usually benefits from a portrait orientation that allows for display of more list items; extra horizontal space is usually wasted. Landscape screens are usually better for columns of data, graphs of events over time, or video. With multifunction mobile devices,

you can allow for viewing either way depending on the type of content or even the device's physical orientation.

Full Team: Brainstorm with Sketches

It's best to begin discussing functional elements and platform considerations even before you begin sketching. However, don't feel like you have to make firm decisions about everything before you draw, because what you learn through sketching will inform your thinking about all of these issues. An efficient design team is unlikely to spend much more than an hour on discussion before they start putting sketches on the whiteboard.

As discussed in Chapter 12, brainstorming encourages creative thinking and can result in some great ideas; it can also clear flawed ideas out of your head to make room for better ones. Unlike requirements brainstorming, early solution brainstorming is usually most effective with just the design team, though I've encountered a few design engineers who have a lot to offer. The key is to keep the group small so it doesn't get bogged down by a lot of people wanting air time, and to

Dip into detail
to see if the
solution works;
then quickly
return to a
higher level.

include only people who are good at thinking in unconstrained ways, comfortable with sometimes-messy exploration, and able to interpret high-level sketches.

Begin with broad exploration at first—even the obviously flawed ideas might spark better ones. Many designers are facile at proposing one idea after another and don't need much help with this stage, but even the most prolific designers have slow days. It's helpful to revisit your context scenarios even during brainstorming; storytelling stimulates the imagination and can help you get started. Changing the rules, such as by pretending the product is magic or is a helpful human, can also help spark new ideas.

Give yourself permission to propose partial solutions based on other designs. For example, if you're thinking about a medical device, perhaps there's something about the way a game controller or mobile phone is designed that's partly applicable. Some designers deliberately use a list of specific product types to stimulate thinking, such as, "What's good about [cars, mobile phones, toys, video games, etc.] that would apply in this situation?" This is where the variety inherent in consulting comes in handy; something about the golf course irrigation system you designed last month might stimulate an interesting idea for a video game design or assisted-surgery system. This is one reason to treat an internal design group as a consulting organization in which designers aren't dedicated to a single product for years at a time.

Regardless of the range of design problems you get to work on, having a brain full of design patterns (and knowing how to apply them) makes a tremendous difference when you're developing initial solutions. Experience is one effective way to build your vocabulary, but it's possible to do so through examining multiple products and reading books on the topic (see Chapter 15 for several important patterns).

The design meeting techniques discussed in Chapter 14 are critical in this early ideation, though because this is brainstorming, it's best to elicit and clarify ideas without going on to critique them just yet; instead, elicit and clarify, then build on the idea or propose another idea of your own. Unlike most design meetings, it's common in this one for multiple people to stand at the board with markers in hand. However, it's still advisable to have someone (typically the IxD synthesizer) ensuring effective work process. Use the biggest whiteboard available so your ideas aren't constrained by space.

As you propose and evolve your ideas, you'll likely find yourself revisiting the list of functional elements and other characteristics, either explicitly or unconsciously. It may or may not be useful for you to keep track of a list in writing; since these functional elements are represented in the sketches, many designers don't feel a need to update a written list. This is fine, since the list of functional elements is not meant to be an administrative task, but merely a way to make an important thought process explicit.

Once you have a number of possible solutions on the whiteboard, you can start evaluating them. Personas, goals, and scenarios are typically the most useful tools for this (see the discussion of validation scenarios in the "Evaluate, iterate, and refine the framework" section). Dip into detail just long enough to figure out if you can make something work. Quickly return to the high level. Although you shouldn't be overly obsessed with cost or other constraints at this point, you should be able to throw out any solutions that are simply too far out there to be viable; before you throw them out, though, consider whether there's anything about them that might be worth incorporating into another solution. Capture rejected ideas before you erase them, however; such notes can serve as a reminder (for the design team and stakeholders) of why you didn't go down a particular path.

Brainstorming for software on a fixed platform

For software on a known platform, this initial brainstorming is generally brief since there are only so many patterns and idioms to use. An hour is usually more than enough to get some crazy ideas up on the board, evaluate them, and clear your head for a more scenario-oriented approach. If this brainstorming goes on too long, you may get attached to ideas that are unsuitable.

Brainstorming for services and new platforms

The initial brainstorming session is more critical for designing services and new hardware/software platforms, since the range of possibilities (and their effect on associated costs) is much greater.

Your sketches should identify major product or service components and approximate interaction among them. Focus on questions like how users will enter data and make selections, rather than on exact screen contents or control types. It's fine to note great ideas about these details if they come to you, but move on once you've done so to avoid getting bogged down.

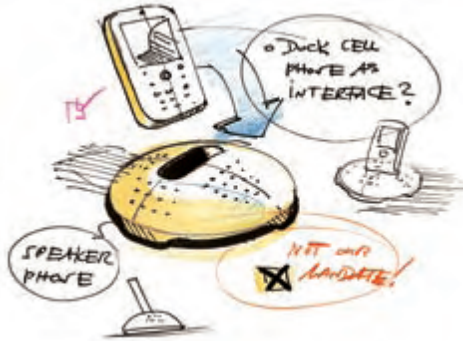
How many ideas you select to pursue in more detail depends on how much time you have. Ideally, you'll be able to choose at least two to iterate for a few days: one that you think represents the best interaction (which often involves pricier components) and one that will allow for a decent experience at lower cost. If you have the luxury of pursuing more than two directions, you might use other criteria to select them, such as what will look most different from the competition, what will be the smallest or sturdiest, or what will best emphasize some other desirable quality.

Table 16.6 shows a condensed version of how a design team narrows and refines a set of office telephone ideas. Note how the IxDS keeps the discussion moving and continually brings the persona into the evaluation. You can also see that the designers are relying on their knowledge of design principles and patterns as they go. Each team member acknowledges where there are problems to be solved, ensures that someone is responsible for exploring each issue further, and moves on.

Table 16.6. Brainstorming and narrowing office telephone concepts.

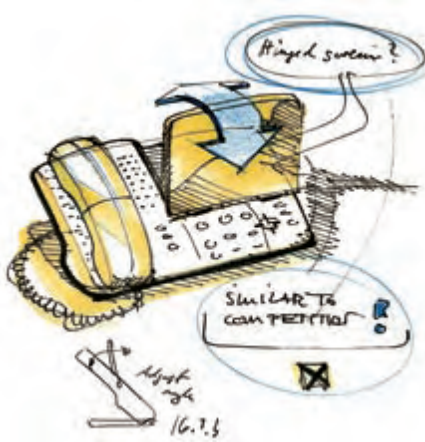
Sketches

Discussion among team members



IxDs: We've brainstormed a bunch of ideas now; let's try to narrow things down and refine them a bit. This idea of docking a cell phone on a speaker-phone base is great, since the average cell phone is much smarter than desk phones already. It seems pretty far out from our mandate, though.

IxDG: Yes, you're right. It doesn't fit the business model at all. We should offer it up as an idea and see if they want to pursue it, but not spend a lot of time on this.



VisD: I think we should also rule out this one with the separately hinged screen; there's nothing inherently wrong with it, but it looks like every other desk phone out there.

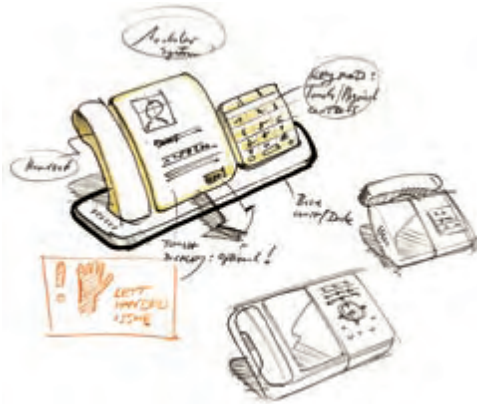


IxDG: Agreed. What about the big touch screen idea? It would allow for very direct interaction—no figuring out which line you're putting on hold—and good integration with the directory.

ID: Yes, there's a lot to be said for that direction. It would have to be big to get all that on the screen at once, though. Also, I'm concerned about having no physical controls at all, though I'm not sure why that bugs me.

IxDs: What happens when Scott needs to dial a number that's not in his directory?

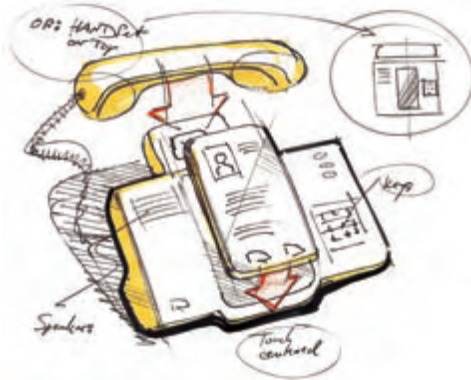
Sketches



Discussion among team members

ID: There you go ... I knew there was a reason it seemed bad. That argues for a physical keypad. Maybe we can make it modular, like this, to allow for selling the phone with or without a touch screen.

IxDG: I think that could work. Isn't that going to pose a challenge to anyone left-handed, though?



ID: Not necessarily, but we could put the handset on top. It would certainly be nice for visual balance. I can try it out both ways.



VisD: We should aim for a portrait screen in either case. It will be better for scrolling through a directory of contacts.

ID: That makes sense. Portrait screens are a little harder to source, but not a big deal.

Continued

Sketches

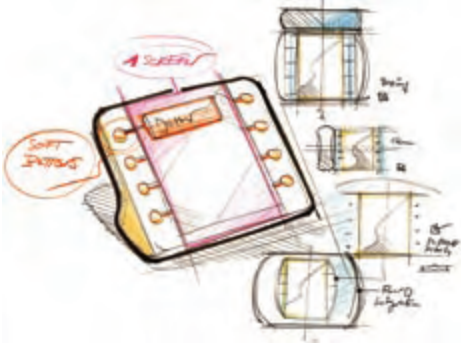
Discussion among team members



IxDS: So, that seems like one good direction to pursue. It's on the expensive side, though. Which of the cheaper possibilities seems like a good alternative? Soft buttons, maybe?

IxDG: Yes, we could use plain old LCDs with soft buttons, like this ... maybe one screen for the directory and voicemail and one for calls.

ID: By the time you deal with two separate screens, I'm not sure it's cheaper. I understand that the interaction is pretty clear that way, but it also leaves an impression of being complicated.



IxDS: [The IxDG] and I will see if we can find a way to do it in one screen.

IxDG: Yeah, we can figure that out. If we could put soft buttons all around it, the interaction could be almost as direct as a touch screen.

ID: Hmm ... that could look pretty crazy with that many buttons. Let me think about how to keep it from getting too busy.



IxDS: So, it sounds like we should spend some more time on this direction. How would Scott scroll through a long list, though?

ID: If it's just in two directions, the best option is probably a jog dial. The scrolling can accelerate depending on how far you turn it.

IxDG: Nice. I wonder about the mapping of scrolling up or down to turning a dial clockwise or counterclockwise, though.

ID: I see what you mean. We can do pretty much the same thing by basically putting the dial edge-on, so Scott just pushes it up or down. There are other controls that do similar things. I can work with the mechanical engineer to see what our options are.

Spend a little more time refining each of your chosen ideas as a group before splitting up to refine the various aspects of the design. The interaction designers should focus on defining an interaction framework for each platform, including interactions with any physical controls, while the industrial designers explore ways that each platform would allow for comfortable use, necessary hardware components, and so on. Visual and industrial designers also begin work on the design language at this time; see Chapters 17 and 18 for more on this.

ID: Refine the Form Factor

While the interaction designers are working through scenarios, the industrial designer continues exploring the form, ideally in collaboration with a second industrial designer to encourage broader exploration. This may involve some sketching, but typically moves quickly to crude physical prototypes built out of malleable materials, such as foamcore, cardboard, string, paperclips, and whatever else is at hand. Some industrial designers like to “sketch” in three dimensions with tools and blocks of foam. If the design includes components or mechanical issues the designer isn’t familiar with, he may consult a design engineer. Figure 16.5, for example, shows how the ID begins thinking about the volume required for internal components using both hand sketches and quick 3D renderings.

The industrial designer’s first objective during this day or two is to see if anything about each form factor (such as its size, cost, or fragility) is unlikely to work, so the entire design team can change direction as quickly as possible. At the same time, the industrial designer works toward



Figure 16.5. Industrial designers must consider internal components early on.

refining the candidate form(s) to feel good in the hand, accommodate the necessary components, and so forth, consulting with mechanical and electrical engineers as necessary.²

- Note that the sort of process described here is focused on interactive products, which have a minimum of moving parts and a relatively predictable set of internal components (boards, fans, screens, batteries, buttons, storage, dials, etc.). This means form can be driven by design; in more mechanically complex products, the mechanical and electrical engineers need closer involvement from day one and may even be driving the nature of the solution. In any case, the industrial designer usually needs to consult closely with the engineers on any product that needs to be small, since compactness usually introduces plenty of engineering challenges.

IxDG and IxDS: Define the Interaction Framework

Key path scenarios will help you group functional elements and lay out your sketches.

As soon as you have either a known or potential platform and input method, such as a desktop computer with keyboard and mouse or a mobile phone with a three-inch touch screen, you can begin working out the interaction framework. For most applications, this is a definition of the major screens or other functional divisions (such as pages for a Web site or menus for an IVR system), the approximate structure and contents of those divisions, and the means for navigating among them, including any relevant hardware controls and how they interact with on-screen content.

Develop a first draft of the framework

Developing the framework is, for many people, the “magic” part of interaction design. It can be a daunting task for any team without strong visualization skills and is the hardest part of the design for even skilled and experienced designers to get right. The techniques described here help with both generating concrete ideas and with ensuring that they’re reasonable.

There are three thought processes that tend to lead to reliably good design in a short period of time. They may occur in sequence, but are more likely to occur somewhat simultaneously. One is the iteration of your context scenarios into **key path scenarios**, which begin to describe the functional elements that help accomplish the activities. Another is the **grouping of functional elements** into sets of tools and display spaces that are commonly used together. The third is the development of **sketches** that represent how the different screens and major components will look.

Many generative interaction designers begin with a few sketches, then use the scenarios to adjust them. If visualization is not your strong suit—or if you’re good at visualization but feeling at a loss about where to start—try starting with the scenarios, then using them to dictate a first cut at the screen layout. Most people get better results this way. The grouping of functional elements usually happens implicitly in either case, though some people are most comfortable with *explicitly* grouping the functional elements based on the scenario before they sketch. This also has the advantage of making it easy for others to understand your reasoning. I’ll describe the process using this explicit approach, then discuss what it looks like with a different starting point.

DEVELOP KEY PATH SCENARIOS

The **key path scenarios** are revised context scenarios that describe flow using the major functional elements. Key path scenarios are still at a fairly high level for all but the simplest of interfaces; they're focused on the most critical (or "key") paths through the interface rather than on exact controls, on-screen text, or the variations on the key paths.

Example—a veterinary hospital management application

Imagine that you're designing a practice management application for veterinary hospitals (this example is considerably simplified for the sake of illustration). Say you've determined that you need two distinct interfaces, a business interface and a clinical one. Laura, the receptionist, is the primary persona for the business interface. Her goals are to:

- Stay calm amidst the chaos
- Keep clients and clinical staff from getting cranky
- Keep all the details straight

Her key activities include the following, which can be expressed in individual context scenarios or, more likely, in a few longer scenarios that reflect her constant interruptions:

- Review the day's appointments
- Create records for a new client
- Check someone in
- Check someone out
- Send bills
- Follow up on a billing question
- Follow up on overdue bills
- Make an appointment
- Change an appointment
- Take a message for a vet

For the sake of illustration, though, let's consider these as separate scenarios; otherwise, the example will get too complex. Let's focus on making an appointment; the progression from context scenario to requirements to functional elements is shown in Table 16.7.

Table 16.7. Example context scenario translated into needs and elements.

Context scenario	Functional needs	Functional elements
Laura takes a call from Mr. Cowell, who needs to make an appointment for his cat to have a tumor removed. Laura looks him up and sees that he has two cats.	<ul style="list-style-type: none"> — Look up callers among existing clients — See overview information about each client and pet 	<ul style="list-style-type: none"> — Area to view client list — Display of overview information for client — Display of overview information for multiple pets
Xena is flagged for follow-up, so she confirms with Mr. Cowell that the procedure is for Xena. She selects that pet's name and chooses the procedure type.	<ul style="list-style-type: none"> — See what pets need follow-up of some kind without delving into detail 	<ul style="list-style-type: none"> — Visual feedback on pet name for follow-up — Area to set appointment parameters

Continued

Context scenario	Functional needs	Functional elements
The system shows the next several non-urgent appointments for Dr. Harvey, Xena's usual vet, when the surgery and hospital space required for the procedure are also available.	<ul style="list-style-type: none"> – System should know how much staff time and what facilities or equipment are required for typical procedures – System should suggest suitable appointment times when all required resources are available, excluding some appointments that may be reserved for more urgent procedures 	<ul style="list-style-type: none"> – Editable default settings that allocate staff time and resources to procedures (admin interface) – Display of best appointment times
She suggests the first couple of dates to Mr. Cowell, who says he was hoping to take care of it sooner. Dr. Bailey also has an opening sooner, but Mr. Cowell prefers Dr. Harvey. She can also see that Dr. Harvey has two slots that are close to the required parameters but don't quite fit them. She looks at the calendar to see what's on either side. One is just Dr. Harvey's administrative time. She moves that to later in the day and offers the new alternative to Mr. Cowell, who accepts.	<ul style="list-style-type: none"> – See other times that almost work to allow for human judgment 	<ul style="list-style-type: none"> – Display of appointment times that might work if calendar is modified – Calendar display for all appointments by veterinarian
<p>Laura tells him he'll get a confirmation in the mail. The postcard immediately gets sent to the printer on Laura's desk.</p> <p>Laura hangs up the phone and greets the client at the desk, who's ready to check out.</p>	<ul style="list-style-type: none"> – For clients without e-mail, automatically print reminders 	<ul style="list-style-type: none"> – Practice-wide preferences for default printing and e-mail reminders (admin interface)

To turn this context scenario into a key path scenario, add the functional elements, like this:

Laura takes a call from Mr. Cowell, who needs to make an appointment for his cat to have a tumor removed. Laura finds him in the **client list** and opens his record to see detail in the **client overview display area**, which shows that Mr. Cowell

has two cats, one of whom is flagged for follow up. She looks at the **pet overview display area**, sees that Xena needs surgery, and confirms with Mr. Cowell that the procedure is for Xena. She clicks to create a new appointment and chooses the procedure type in the **appointment parameters area**.

In the **best appointment display**, the system shows the next several non-urgent appointments for Dr. Harvey, Xena’s usual vet, when the surgery and hospital space required for the procedure are also available. She suggests the first couple of dates to Mr. Cowell, who says he was hoping to take care of it sooner. Dr. Bailey also has an opening sooner, but Mr. Cowell prefers Dr. Harvey. She can also see two time slots in the **alternate appointment display area** that are close to the required parameters but don’t quite fit them. She looks at the **calendar** to see what’s on either side of these. One is just Dr. Harvey’s administrative time. She moves that to later in the day and offers the new alternative to Mr. Cowell, who accepts. Laura tells him he’ll get a confirmation in the mail. The postcard immediately gets sent to the printer on Laura’s desk. Laura hangs up the phone and greets the client at the desk, who’s ready to check out.

Notice that the level of detail doesn’t change much—the focus is on chunks of screen real estate, not on widgets and detailed data. You’ll want to translate each of your context scenarios into a key path scenario. You can tackle one scenario, sketch the framework for it, and then do a second scenario, or you can iterate all of your scenarios and do some explicit groupings first. Visually oriented people may be most comfortable with one scenario at a time, since this gets to sketching faster.

Whether you write down each scenario in detail depends on how much time you have and how much your stakeholders want traceability in the process. An experienced design team in a hurry can often do this kind of thing live in a meeting, without writing everything down. I recommend taking the more thorough approach until you’ve mastered it, though.

GROUP FUNCTIONAL ELEMENTS

The next step is to look for evidence of which functional elements should be used together and which don’t belong in the same screen. This grouping of elements into screens is based on the fundamental design principle that things people use for a particular task should be within easy reach, while things they don’t use for that task should be out of the way. A **screen** (or a page on the Web, or a menu in a voice interface) is a distinct collection of tools and content that your personas will think of as a “place” to go. The contents of a “place,” and sometimes their arrangement, can change to some extent without users feeling like they’ve left one room and gone to another; changing a calendar from week view to day view, for example, doesn’t feel like going to an entirely different place, but switching from a calendar to a list of e-mail messages does. At this point, you should mostly be concerned with screens or places rather than the individual states they can be in.

Because good grouping is based on natural flow, scenarios are indispensable tools for determining how elements relate: If several elements are used together, they may belong on the same screen, while an element that doesn’t get used alongside them probably belongs on another screen. Microsoft PowerPoint and Apple Keynote are clear examples: You need text and drawing tools at your fingertips when you’re creating slides, but they’d only get in the way when you’re delivering a presentation.

Some people find it easy enough to look at a scenario and see which elements are used together, but others find it helpful to list major functional elements on the whiteboard, then diagram the flow among them for each scenario.

Figure 16.6 illustrates this for the veterinary appointment scenario above. In this case, a single diagram is not terribly informative, but you can

see that Laura will return to the client list after this fairly brief transaction. Diagramming the other scenarios would show a similar tendency to return to the client list every couple of minutes. This implies that an organizer/workspace pattern (see Chapter 15), with the client list persistently available, might be a good place to start sketching if you want to avoid pogo-stick navigation.

Figure 16.7 illustrates a more subtle case using the following scenario, in which a persona named Ray organizes his images:

After a day of hiking on his New Zealand vacation, Ray has two memory cards full of landscape and wildlife images. He hooks the card reader up to his laptop and turns it on. PhotoMaster opens as soon as the laptop detects the card. It automatically begins importing the images into a default “new imports” category, which appears in the category organizer, quickly loading image previews into the multi-image preview area so he can start organizing images while the large files from his 12-megapixel camera slowly load. An indicator tells him

this card will take about 10 minutes to load. He has a little time before dinner, so he begins to do so.

Ray starts by applying several keywords to the entire batch of photos using the keyword pane. He then looks at each image in the single-image viewer and decides whether to keep it, sometimes comparing a few side by side in the comparison area to determine which ones are best. Nearly half the photos are rejected; they disappear from the screen but stay in temporary storage for a while in case he changes his mind. He then adds keywords to each image using the keyword pane.

When the card has finished loading, PhotoMaster notifies Ray that it’s finished. He inserts a new card, and PhotoMaster automatically starts loading these images into the same group as Ray continues working. As soon as all the images are loaded and the rejects deleted, Ray tells the application to back the images up to his PhotoMaster online archive and heads to dinner.

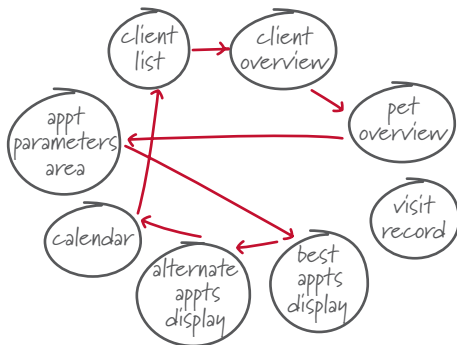


Figure 16.6. Flow among functional elements—a diagram of how major functional elements are used in the veterinary appointment scenario.

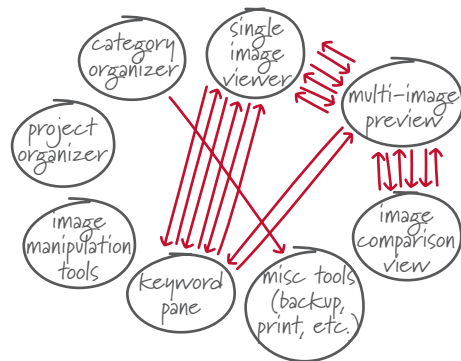


Figure 16.7. A flow diagram example for organizing photos.

Ray is constantly moving between the single-image view and multi-image view. He's also moving back and forth between the multi-image preview and the image-comparison view, but he does not move directly between single images and comparison. This implies that the multi-image preview needs to appear persistently during this activity, but that the single-image view and comparison view don't need to be visible at the same time. You can also see that the image manipulation tools, which are required for another scenario, are not used in this one, so they may not need to be accessible at the same time as the keyword pane. Other scenarios may disprove this hypothesis, but it provides a reasonable starting point for sketching some tentative screen layouts.

START TO STORYBOARD SCREENS AND NAVIGATION

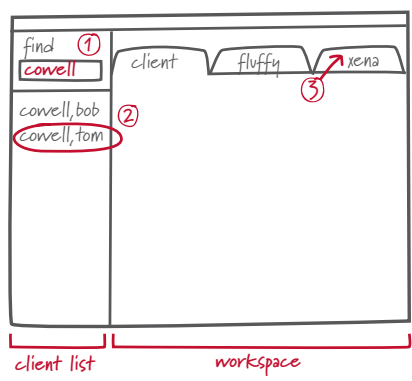
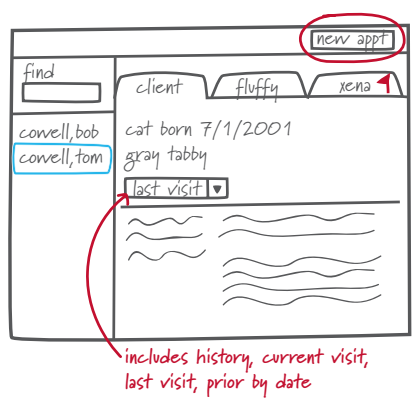
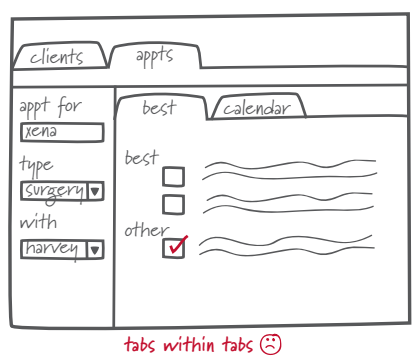
The point of all this thinking is to visualize good, concrete solutions on a deadline—to help you jump across the chasm between understanding users and solving their problems, and to bring others across that gap with you. This requires drawing an approximation of each major screen or other “place,” being able to describe how it works in combination with others (and perhaps hardware controls), and having confidence that the details you haven't addressed yet will be straightforward to work out later on.

Start by storyboarding one scenario for the primary persona using simple thumbnails. These usually consist of empty rectangles describing what major elements go on each screen and how they relate to one another. These rectangles might contain the name of the functional element or some kind of rough conceptual representation of the element's contents, but should not contain specific widgets or detailed text. As discussed in Chapter 14, staying at a high level at first helps you move faster and develop a cleaner, more consistent design approach.

The scenario's flow usually suggests how to position the screen elements: For readers of Western languages, whatever the persona uses first probably belongs at the top or left side of the screen. Whatever she uses next belongs to the right of or below that first element. The nature of each element's contents often suggests its size and shape: tall, narrow rectangles for long lists, most of the screen for photos or other rich content, and so forth. You'll have to take your best guess at how much space is needed based on your understanding of the objects and attributes. Of course, these elements may change size, shape, or location as you evolve the sketches, but everything gets easier once you have something on the whiteboard to work with. Table 16.8 shows an initial set of storyboards for the veterinary appointment scenario.

The point is to visualize good, concrete solutions on a deadline—and to help others get there with you.

Table 16.8. Sketching from a scenario.

Scenario step	Sketch	Comments
<p>1. Laura takes a call from Mr. Cowell, who needs to make an appointment for his cat to have a tumor removed. Laura finds him in the client list and opens his record to see detail in the client overview display area, which shows that Mr. Cowell has three cats, one of whom is flagged for follow-up.</p>		<p>This first sketch draws upon the organizer/workspace pattern and the data model, which indicates that pets are parts of the client record. The “find” field is understood as a placeholder for some way or ways to locate clients.</p>
<p>2. She looks at the pet overview display area, sees that Xena needs surgery, and confirms with Mr. Cowell that the procedure is for Xena. She clicks to create a new appointment.</p>		<p>Whoops, better add a toolbar for that “new appointment” button. Notice there’s a little bit of detail about what may be on the screen. The interaction designer captures an idea for navigating visits, but quickly moves on.</p>
<p>3. She chooses the procedure type in the appointment parameters area. In the best appointment display, the system shows the next several non-urgent appointments for Dr. Harvey, Xena’s usual vet, when the surgery and hospital space required for the procedure are also available. She suggests the first couple of dates to Mr. Cowell,</p>		<p>Uh-oh ... what should happen when that button is clicked? Does it make sense for a calendar to live as a tab inside the client record? It seems more like a global tool, so the team decides to have a client screen and calendar screen, with top-level tabs to switch between them. Tabs within tabs are unfortunate, but the team recognizes this as an issue they can solve later and keeps going. Perhaps the client</p>

Scenario step	Sketch	Comments
<p>who says he was hoping to take care of it sooner. Dr. Bailey also has an opening sooner, but Mr. Cowell prefers Dr. Harvey. Laura can also see two time slots in the alternate appointment display area that are close to the required parameters but don't quite fit them.</p>		<p>list doesn't need to be visible in appointment view, so they use that space for appointment parameters. The content shown is understood as a placeholder for more complex controls. Appointment possibilities appear on the right. The screen shows the best and alternate appointments on the same screen for comparison; the best options should be at the top.</p>

4. She looks at the **calendar** to see what's on either side of these. One is just Dr. Harvey's administrative time.

moves admin time

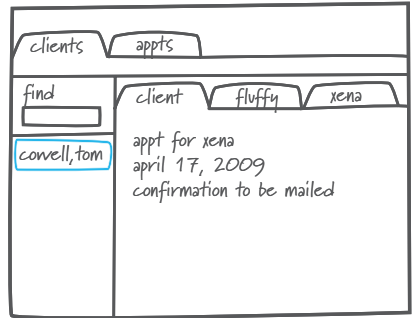
Clicking on one of the alternatives brings Laura to a calendar view, again shown with tabs. The team acknowledges that this jumping from tab to tab might seem awkward, but it's probably not worth worrying about until they've storyboarded the other scenarios. The calendar view shows other veterinarians (who might be alternatives if the first choice vet doesn't work out) and highlights the open appointment time and potentially adjustable administrative time.

5. She moves that to later in the day and offers the new alternative to Mr. Cowell, who accepts. Laura tells him he'll get a confirmation in the mail. The postcard immediately gets sent to the printer on Laura's desk.

books it somehow...

Laura grabs the administrative time and moves it, and then somehow books the appointment. The "book" button is understood as a placeholder for some more refined mechanism.

Continued

Scenario step	Sketch	Comments
<p>6. Laura hangs up the phone and greets the client at the desk, who's ready to check out.</p>		<p>Laura goes back to client view. Hmm ... perhaps making the client list available only in the client view is a bad idea after all, since most of the scenarios indicate she'll go back to it often. The team decides to back up and see what would happen if the client list were persistent, as the diagram in Figure 16.6 indicated.</p>

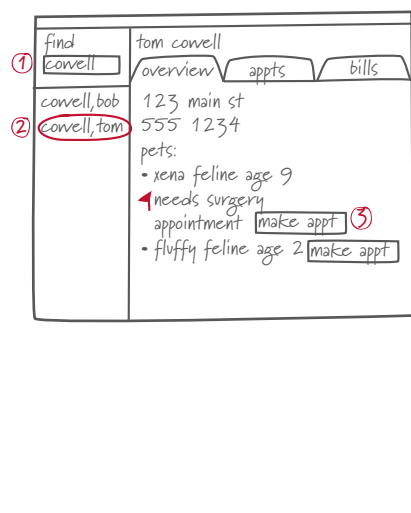
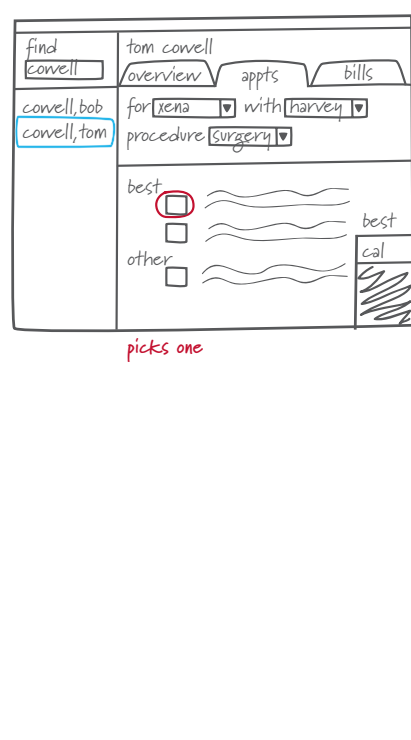
Relying on your scenario will yield better design decisions than using abstract reasoning.

If you pause and consider the persona, the scenario, and the data model from Table 16.1, you might recognize a few problems with the storyboards in Table 16.8. One is that the sketch in step two is showing a lot of information about each pet, which a receptionist like Laura has no need to see. Another is that there's most of a screen dedicated to general client information, but the data model tells you that other than bills—which may be complex enough to warrant their own tab—there's not much to track about each client beyond a name and contact information. You may also see that where the design goes a little astray is where the team forgets the scenario and decides, based on some abstract reasoning, that appointments should be a global tool. What the team *should* be asking is whether they can think of any scenarios in which a view of appointments is not linked to a client.

What the previous paragraph describes is just the sort of conversation that should happen in a design meeting once there's a design on the board. Getting this kind of storyboard developed doesn't take long, so a little patience doesn't cost much even if the solution seems off base at first.

When you get to a point like this where the design isn't quite working, sometimes the right thing to do is to backtrack and try again. So, imagine that the design team has realized they're dedicating a lot of space to things Laura doesn't need in her role as a receptionist, so they take another crack at the storyboards using a persistent client list, as the flow in Figure 16.6 suggested. Table 16.9 shows how the design might evolve.

Table 16.9. Revising the design.

Scenario step	Sketch	Comments
<p>1. Laura takes a call from Mr. Cowell, who needs to make an appointment for his cat to have a tumor removed. Laura finds him in the client list and opens his record to see detail in the client overview display area, which shows that one of his cats, Xena, is flagged for follow-up because she needs a surgery appointment. She confirms with Mr. Cowell that the procedure is for Xena. She clicks to create a new appointment.</p>		<p>This first sketch again supposes that appointments are part of the client record, rather than a top-level tool. It also eliminates the tabs for each pet, since Laura doesn't need all the clinical information, and just puts a summary on an overview tab.</p>
<p>2. She chooses the procedure type in the appointment parameters area. In the best appointment display, the system shows the next several non-urgent appointments for Dr. Harvey, Xena's usual vet, when the surgery and hospital space required for the procedure are also available. She suggests the first couple of dates to Mr. Cowell, who says he was hoping to take care of it sooner. Dr. Bailey also has an opening sooner, but Mr. Cowell prefers Dr. Harvey. She can also see two time slots in the alternate appointment display area that are close to the required parameters but don't quite fit them.</p>		<p>Laura can go straight to the appointment tab without an intermediate screen, so this is better. The IxD generator sketches two sub-tabs for a calendar view and the two sets of appointment options, though this seems overly complex.</p>

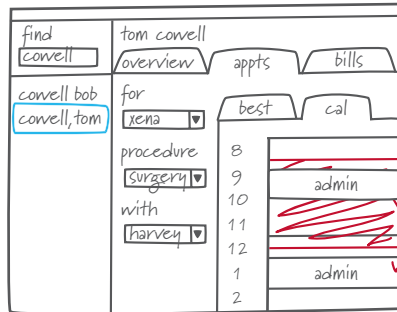
Continued

Scenario step

Sketch

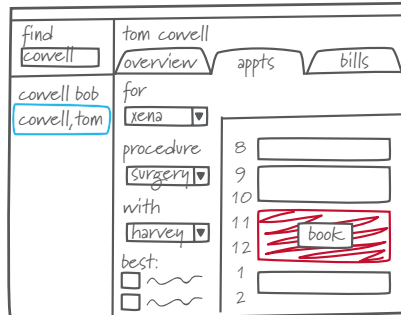
Comments

3. She looks at the **calendar** to see what's on either side of these. One is just Dr. Harvey's administrative time.



In this view, the IxDG tries a slightly different layout, since using a pane that's wider than it is tall, as in the previous step, is less effective for displaying a list of appointments. This version is a better use of space, but still has the unfortunate sub-tabs.

4. She moves that to later in the day and offers the new alternative to Mr. Cowell, who accepts. Laura tells him he'll get a confirmation in the mail. The postcard immediately gets sent to the printer on Laura's desk. Laura hangs up the phone and greets the client at the desk, who's ready to check out.



In this step, the IxDG tries a version without tabs, reasoning that a list of potential appointments doesn't take that much screen space and can be used to drive the calendar. The calendar view may not have room to show the other veterinarians (who might be alternatives if the first choice vet doesn't work out), but this seems a reasonable compromise. The scenario is accomplished with less extraneous navigation, so this looks like a better direction.

Framework

ADD TO THE DESIGN AND ADJUST IT FOR ADDITIONAL SCENARIOS

Although you could start with fresh "rectangles" for each scenario and try to combine them later, it generally works better to start with the structure you roughed out in the first scenario—even if it seems like it's a little bit wrong—and use what you learn from subsequent scenarios to add onto it and adjust it. In other words, rough out scenario number one, change that structure based on scenario two, make sure it still works for scenario number one before moving on to do the same

for scenario three, and so on. A structure should start to coalesce after two or three scenarios.

For our veterinary application, let's say that scenario number two involves Mr. Cowell calling up to change his appointment. Figure 16.8 shows how a relatively small adjustment to the design (from Table 16.9) accommodates the additional scenario; existing appointments are listed above the tools for new appointments and can be selected to appear in the calendar. There are some niceties of the interaction that will need to be ironed out, and it may not be quite the right answer yet,

but the fact that this design concept mostly works after a second scenario is a good sign that the team is headed in a productive direction.

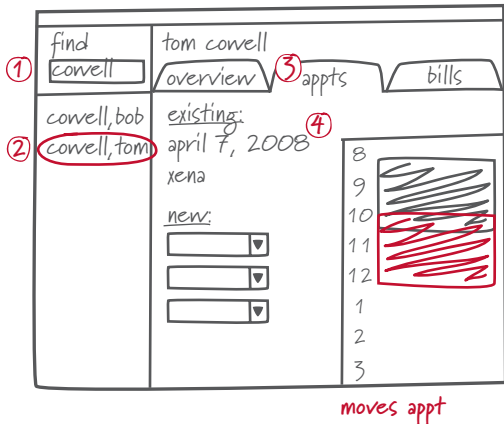


Figure 16.8. A minor adjustment to the design accommodates an additional scenario.

As you can see from the examples, what actually happens at the whiteboard is a combination of the scenarios, the data model, design judgment, and a good vocabulary of design patterns. Other ideas and considerations will affect what you draw, sometimes with good results and sometimes not. If you always return to the scenarios and goals as your touchstone, though, more abstract considerations are less likely to get you into trouble. When teammates make assertions not based in the current scenario, ask them to describe a scenario in which their assertion makes sense.

This kind of iteration is very fast; with an effective design team, the entire progression from key path scenario to this point would likely take less than half an hour. Even though the design team is not developing multiple directions to show stakeholders, they're quickly trying and eliminating many directions at the whiteboard.

As you proceed through each of your context scenarios, consider whether you can consolidate any of the screens you've drawn. Are two screens showing slightly different views of the same content? If so, perhaps they can be combined. Do you have two screens that use the same functional elements in different places? Maybe you can find a way to use just one layout for both, as long as it doesn't introduce major problems; the gain in simplicity is usually worth the minor trade-off of moving a pane to a different spot. Are there any other ways you can apply design patterns to simplify the overall structure?

Your context scenarios probably include either a scenario for a task unique to a secondary persona or a scenario describing how the primary and secondary approach the same tasks differently. You can either do all of the scenarios for the primary and then adjust for the secondary, or do the secondary persona's version of the similar scenario before moving on to a different set of tasks. Try both approaches to see which works better for you. If what you've drawn for the primary persona works, that's great. If not, adjust the drawing to accommodate the secondary persona's needs, then revisit the primary persona's scenario to make sure your change doesn't introduce problems. If you need to introduce additional elements to address a secondary's needs, make sure you do so without getting in the primary persona's way; otherwise, it's possible you really have two primaries who need separate interfaces.

Develop rough thumbnail storyboards for *all* of your key path scenarios before you delve into more detail on any one screen; otherwise, you're likely to waste a lot of time refining a screen that you'll wind up throwing away. (Worse yet, you might get attached to it and not *want* to throw it away.) It's easier to stay at a high level if you plow through thumbnail storyboards for the entire set of scenarios in a short time, such as an afternoon if you have a handful of scenarios, or perhaps a day if you have a dozen. If you were working on a device, the level

of detail shown here would be sufficient for the interaction designers to share with the rest of the design team and confirm whether the selected platform is likely to work out. However, it usually takes another couple of days before things are solid enough and articulated enough to be worth sharing with the project owner and design engineer(s).

HOW THE SKETCH-FIRST APPROACH DIFFERS

Most generative interaction designers are inclined to sketch screens before working through functional groupings and turning context scenarios into key path scenarios, often because they've done some of this thinking in their heads, but sometimes because they think much better in sketches than they do in words. If you want to use this approach, go ahead and get your ideas on the board, then immediately turn to your context scenarios and walk through what you've just drawn. If the flow makes sense and your sketches adhere to good design principles, then the context scenarios with your functional elements plugged in become the key path scenarios. Chances are, however, that your first round of sketches doesn't quite work. If there's not an easy fix you can see from walking through the scenarios, then it's usually best to start over using the scenario-first approach.

EVALUATE, ITERATE, AND REFINE THE FRAMEWORK

Next, you'll want to sketch each screen with slightly greater resolution, though without worrying about exact widgets, text, and similar details. Plan your time so you can spend about half a day for a straightforward scenario or about a day for a more complex one. You can also carve up your time by screen rather than by scenario; this is more manageable, but can sometimes be less coherent since a scenario may require more than one screen to complete. In either case, limiting the amount of time you spend on each design topic helps ensure that you're evolving the entire design to the same degree.

Try iterating your key path scenarios and your list of functional elements (whether it's in your head or written down) to a slightly greater level of detail. Draw a rough draft of each screen on the whiteboard, indicating approximately what sort of tools and content appear in each pane or region. Adjust each screen as needed for any secondary personas, just as you'd adjust for a second scenario (as shown in Figure 16.8). Indicate specifics only where necessary to accomplish the key path scenarios; don't obsess over whether a widget is the best choice or what its label should be. These are only placeholders, since you might realize later that your first guess doesn't work very well.

It feels good to get the design articulated this far, but don't get attached to it yet. You need to start throwing more scenarios at it to determine whether it will hold up to a realistic range of circumstances. There's usually no need to write up detailed scenarios or formal use cases to evaluate and refine the design. **Validation scenarios** are informal "what ifs" posed by a member of the team (usually the IxD synthesizer) once there's a design on the board. The person who proposed the solution (usually the IxD generator) then shows how the situation would be handled by the existing proposal or, if the design doesn't address the need, modifies the design to address the issue. For example, using Figure 16.8 as a starting point, the conversation might go something like this:

IxD S: What would happen if Mr. Cowell wanted to make an appointment for both cats at the same time?

IxD G: That does seem like a pretty common case. I suppose Laura could use a multi-pet appointment control on the overview tab, but I think those controls are really shortcuts and Laura would understand that you do more complex appointment setup by going to the appointments tab. Once she's there, we could either add "multiple pets" as an option in a pet selector list box

or maybe use some kind of multi-select control. We'd have to offer an appointment type for each pet, and then the system would suggest open times that are long enough to account for all of them. The screen might get a little dense, but I think we can work it out.

IxDs: OK, that doesn't seem like it will break the framework. We can save the details for later. What about ...

This example is typical of how loose and informal validation scenarios usually are. Notice how the IxDG offers just enough detail to ensure that something can be worked out. Once both designers are satisfied, they move on. (If you have any non-designers in the room, they may be uncomfortable with this because they don't have the pattern and principle vocabulary to see how the problem could be solved. Plan to spend more time on this activity than you otherwise would.)

Early validation scenarios should be variations on the key paths that are either relatively common or, if they occur infrequently, are somehow essential to the product's success. It's still much too early to throw obscure edge cases at the design. If your key path scenario involves the persona ordering a routine prescription refill from a pharmacy, it's reasonable to explore how the proposed design would handle a prescription that's not covered by insurance or is too old to be refilled. What happens if someone has to have a prescription delivered while she's traveling out of the country is probably something to save for later. You'll need to solve this problem eventually, but it's obscure enough that it shouldn't be driving the design.

If a teammate proposes a validation scenario that seems too far out there for this stage of the design, make sure you understand where the concern is coming from; it's possible he just isn't being very articulate. If you feel he's hung up on an inappropriate detail, use the 15-minute rule or, if you can't get someone else in the room right away, put the issue in the "parking lot" until you can. (See Chapter 14 for a description of both techniques.)

You'll know you have a workable direction when the design you've sketched seems to handle multiple validation scenarios without breaking. Once you reach this point with any part of your framework, make sure you've got a clean set of whiteboard drawings with all the necessary parts in the right states. Assign explicit, straightforward names to each screen, pane, and important widget. Pause for a few minutes to capture your sketches, label elements, and take notes about behaviors you discussed. The design may not be ready to show to the

Early validation scenarios should be common or important variants of the key paths.

entire set of stakeholders yet, but it's probably ready to share with your project owner and design engineer(s). If you're developing multiple design directions, this is a good time to move on to your next one.

How to approach specific design situations

The process described above works very well for single-interface desktop applications. It's equally effective for almost any design problem, but some circumstances require slight modifications to the approach.

TIGHT CONSTRAINTS

If you're doing a relatively minor update of a legacy product or facing some other tight constraints, inventing new functional elements and navigation structures would be a waste of time because nothing close to your solution would get built. However, a less ambitious version of the same process can yield great results. (It also helps if you view constraints as just another part of the design problem, rather than as frustrating limitations.)

Start by making a list of the existing functional elements. Develop your key path scenarios to describe the ideal flow among them. It may not be that difficult to put components on different screens if the scenarios suggest a different grouping for the elements; it all depends on how they're built. If there are needs in the scenario that can't be met with existing functional elements, you have a starting point for a discussion about feasibility. If it turns out that you can't add or make significant changes to elements and can't change what appears on which screen, then you're really working at the level of detailed design; see Chapters 20 and 22.

MULTIPLE INTERFACES

When you have multiple roles and therefore multiple primary personas, as in many enterprise applications, each primary persona will need a unique interface with tools and information focused on his specific needs. Unique logins ensure that each user sees the appropriate interface. This adds another layer of complexity to the framework definition puzzle because each primary's framework usually has at least a partial overlap with the others. Maximizing that overlap without compromising the needs of any primary is tricky, but it's desirable for a couple of reasons. The one most development teams focus on is cost: The greater the overlap in the design, the more efficient coding, testing, and ongoing maintenance can be. However, some overlap in interface components also helps users in different roles teach one another, troubleshoot problems, or discuss issues related to a shared file.

When you have multiple interfaces to address, start by doing rough storyboards for each primary persona's interface (to about the level shown in Table 16.9) before you get into detail on any one of them. Once you have basic structures in mind for each, step back and look for opportunities to share components. If it won't cause problems to modify two similar functional elements to make them identical, collapse them into a single element shared across the two interfaces. Once you've made the components as modular as you can without hampering any of the personas, you can work through the scenarios for each primary and associated secondaries in more detail. Just leave yourself a little time to revisit each primary's framework after you've done this to see if any elements you were hoping to share aren't working out and whether any more opportunities for consolidation have emerged.

WEB SITES

Information-focused Web sites involve two unique considerations not addressed by the generalized version of this scenario-to-framework process. One is that informational sites seldom have complex interaction; the primary design challenge is to get users to the right piece of information. The other is that users with entirely different needs are all coming to the same place; it's not practical to separate people into unique interfaces the first time they visit, and sometimes not worth bringing people to different home pages based on cookies.

Getting users to the right information

The interaction framework of an informational site (and of many e-commerce sites) is seldom about managing distinct sets of tools and data for distinct tasks. Other than the occasional bit of account management, there's usually just one task: find the right piece of information or the right product. The field of information architecture, which I would argue is a specialized subset of interaction design, is focused on addressing this need in various contexts.

Solving this problem starts with understanding whether your personas are looking for a uniquely identifiable product or piece of information, such as a copy of a particular book or the date of the first lunar landing, or for something they can't specifically identify, such as a new outfit to wear to a friend's wedding. Search works well in the first case, but categories usually work better in the second; the hard part is figuring out what structure of categories will lead your personas to the right sort of information or product.

Scenarios help you envision the sequence in which your personas will look for information; when combined with the taxonomies in mental models (see Chapter 7), they can help you get users to their desired items with very little effort. When your persona is looking for a new outfit, what information does she know when she starts,

and what criteria does she use first to filter out the information or products she doesn't want? What criteria does she use after that? Shoe-shopping Web sites such as Zappos.com or Endless.com are good examples of this: Most women looking for shoes think first about what style and color of shoe they need, so the categories start there, allowing users to identify potentially interesting shoes before filtering by size, width, and other criteria.

Focusing on differing persona needs

Although there is generally one primary persona for a Web site, you can use other personas as primary for certain sections or types of content. An emotional buyer, for example, makes a good primary for a luxury car Web site because the site will lose him if *any* part of it doesn't appeal to his self-image. However, it makes more sense to direct detailed specifications at other sorts of prospective buyers who are more likely to care about them.

Use the **site primary** to develop your first take on the framework, with the others serving the same purpose as any other secondary persona. Once you get into areas of the site for which the site primary isn't the main focus, use the most relevant persona as the **section primary** for that part of the design.

DEVICES

Handhelds, telematics, medical instruments, and other devices clearly present unique interaction design challenges due to their physical forms. One of those challenges is squeezing the necessary software controls and information into a small display (as on a handheld) or relatively low resolution (as on most televisions, though this is improving with HD). A parallel workspace or hub-and-spoke pattern (see Chapter 15) usually works to manage information on a small screen. Before you spend a lot of time at the whiteboard, though, mock up a screen of the appropriate physical size and resolution on your computer to see how much text or other information you can realistically display.

The other challenge involves assessing and refining the form factor for a new platform or figuring out how to work within the hardware specifications of an existing device.

New devices

When designing a new device, you're probably either working with a single candidate platform or comparing a couple of possibilities. In either case, the first priority is to carry the interaction framework far enough to assess the proposed form factor and input methods, so you can work with the industrial designer to change directions if necessary. This begins much like desktop software design, with a quick run-through of each scenario (for each possible direction, if applicable).

The trickiest problem with hardware/software interaction design is determining what sort of hardware controls you need and how they'll interact with the on-screen content. With desktop software and Web sites, interaction designers are accustomed to having dedicated screen space and buttons for just about everything, so no widget has to serve multiple functions. This makes the interaction more clear; no one has to wonder why that button sometimes prints and sometimes closes the application. For this reason, if the device does not involve a touch screen or movable cursor, an interaction designer's first instinct may be to have dedicated physical controls (and sometimes even dedicated displays) for just about everything; the multiscreen sketch in Table 16.6 illustrates this tendency. It's fine to start here, but a device with a button for everything starts to look like the inside of a 747 cockpit and may add too much size or cost.

If a device has a touch screen, on the other hand, interaction designers may be inclined to do just about everything on-screen, since they can make specific controls available only in the context where they're needed. This may be great from a cognitive perspective, but it may be the wrong answer for

some interactions due to ergonomic or safety considerations, or the need for immediate access to a function at all times. There are also some interactions that just feel better with physical controls.

Go ahead and start your initial storyboards with whatever combination of software and hardware controls you think you'll need. As with software-only design, step back when you're done and consider the possibilities for consolidation, as well as which controls need to be physical. Collaboration with an industrial designer adds valuable perspective. In general, the following are good candidates for dedicated physical controls:

- Anything people need to use without looking, such as the volume control on a car stereo
- Controls that have to be instantly available at all times, such as a mute button on an office telephone
- Functions that are divorced from the on-screen interaction, such as power buttons or “lock” controls that prevent accidental input on pocket devices
- Anything that needs to be very responsive and able to take a beating, such as direction and firing buttons for a game controller
- Things that must be accessible to the visually impaired with no adjustments required, such as elevator buttons
- Functions that need to be found in the dark, such as the pause button on a remote control
- Interactions that require manipulating multiple controls at a time, such as mixing audio (though multitouch screens can also accommodate this)

However, transient choices or functions that are only sometimes available should rarely employ physical controls, partly because they're not worth the real estate, and partly because it's difficult to convey that a hardware control is unavailable under certain circumstances (though it can

be done with LED backlighting, for instance). Touch screen controls, a movable cursor, or soft keys are usually better options.

When deciding whether a single button can serve more than one purpose without a soft label, consider how closely related the actions are. Play and pause, for example, make sense as a single control because they're opposite states; this idiom is familiar from light switches and power buttons. (Mind you, it's easy for users to get confused about button state if you change the label and there's no clear audio or video playback, as in a Web conferencing application; does the right arrow mean that "play" is the current state, or that clicking the button will cause the system to play?) Holding down a menu button to turn on a backlight, though, is obscure because there's little relationship between the two functions. Also explore how different types of information can take over the screen temporarily as needed, rather than trying to show everything at once. Consider the following conversation and the resulting sketch, shown in Figure 16.9.

IxDs: Scott arrives at his desk, sees that he's got messages, and looks at a visual list to pick the top priority. He gets a call while he's listening, so we have to have space for an active call to show up without totally losing his context. The same is true for the directory; he could be looking something up when a call comes in. For that matter, he could be on a call and need the directory to add someone to a conference call.

IxDG: Right. So, it seems like each of these tools takes over the screen, but has some flexible behavior to allow screen sharing when a call comes in. There's no reason he'd need the directory and voicemail at the same time, though, so those don't need to coexist, right?

IxDs: Probably, yes.

IxDG: OK. That implies that the directory and voicemail should essentially be tabs, but that the tabbed area shrinks as needed when a call comes in, something like this (draws on board). Each tab slides open like a drawer and slides partially closed as needed. He'd reasonably need the directory information while he was on a call, though I don't think he'd ever open up the voicemail while on a call. Still, I don't see a reason for the two to behave totally differently. This seems pretty clean.

IxDs: Sounds like it will work. Let's try some scenarios to see if it breaks.

When designing for several possible hardware platforms, carry each interaction framework design far enough to evaluate how well each form factor and input method works.

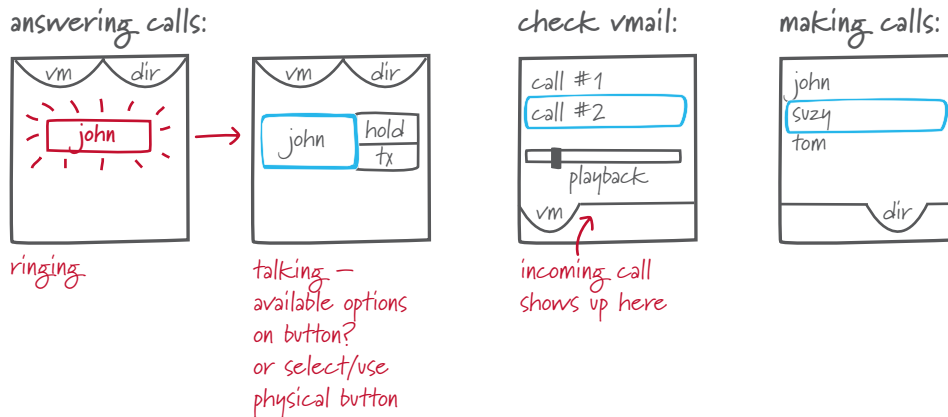


Figure 16.9. Making the office phone work with one screen.

As with any other design problem, consider what will or won't make sense to your personas and adjust your design accordingly. Run through your key path scenarios, then use validation scenarios to continue assessing and iterating the design. You'll know you're headed in a good direction when the design is handling every scenario you throw at it.

Existing platforms

Designing for an existing device can be frustrating if the hardware design wasn't done with the software in mind. I had a client once who wanted interaction design for a multifunction device with exactly one hardware button; they wanted the device to be dead simple, so they apparently thought nothing could be simpler than a single button. Unfortunately, the visual simplicity added a lot of cognitive complexity; two more buttons for separate navigation and selection would have been much more clear. However, not all devices can be redesigned, either because of time and money constraints, or because you're designing an application to run on third-party hardware.

If the hardware is reasonably flexible, such as a smart phone with keyboard and touch screen or cursor-based navigation, you can proceed much

as you would for a desktop application. When the input is limited to a few soft buttons, you'll probably need to use a hub-and-spoke pattern, with the number of spokes determined by the number of soft buttons; structuring the information in this case is a lot like structuring a Web site's information architecture. Even more constrained devices may require a tunnel (see Chapter 15 for explanations of both patterns). If the hardware has an unusual configuration, take stock of what you have and brainstorm about how you can use each of the existing controls.

VEHICLE INTERFACES

Automobile interfaces (such as dashboard controls, navigation systems, and entertainment systems) are much like other device interfaces, but they involve one unique and critical factor: The person operating them is probably driving a two-ton missile down the highway at high speed, so designers must consider safety of paramount importance—first, do no harm.

Use physical controls that can be distinguished by position and feel to help keep the driver's eyes on the road. (You should see the stares I got shopping for a car stereo once, as I closed my eyes

and felt my way around the different faceplates to find one I could use without looking!) Minimize reliance on text and images; make them large and high contrast if they're unavoidable. Make typical interactions, such as switching between a radio station and CD, easy to accomplish with a single physical control. Consider putting a couple of commonly used controls, such as the stereo skip/seek and mute buttons, within thumb reach on the steering wheel to limit one-handed driving. Use audible feedback and simple voice commands when possible.

In addition to keeping eyes and hands free for driving, you also need to help keep the driver's *mind* on the road. Numerous studies indicate that cognitive distraction, not just visual distraction or fumbling with controls, is a major factor in safety. Even hands-free cell phone conversations decrease response time and substantially increase the risk of accidents. Redelmeier and Tibshirani³ found that accidents quadrupled when drivers were talking on the phone. McKnight and McKnight⁴ found that simple, quick calls (such as "I'm stuck in traffic and running late") made by younger drivers were slightly less distracting than tuning the radio, but that more involved conversations such as business or intense personal calls had far greater effects on reaction time and other driving safety factors. Older drivers were distracted by even brief calls. Research into other in-vehicle distractions (such as listening to e-mail using a voice interface)⁵ indicates similar problems due to the cognitive load.

First, consider whether a particular interaction really needs to happen in a moving vehicle. If it's unavoidable, be sure you not only follow good design principles, but allow time for extensive usability testing and iteration. Whatever you do, though, don't force a driver to scroll through and accept two pages of legal disclaimers before she can use the navigation system; it might keep you from getting sued, but is even more likely to cause an accident.

AUDIBLE INTERFACES

Most audible interfaces have something in common with informational Web sites: The challenge is to get users to the right information or transaction as quickly as possible, and there are a limited number of ways to do so. The safe route is a hub-and-spoke, menu-driven approach: "To access your existing accounts, press one. To create a new account, press two." Even if you're planning to use random-access voice input, it's still worth drawing a menu structure of all the options (and listing every way you can think of to phrase them) so the system can recognize a wide range of requests and offer them as a list when users can't think of a command.

As in information architecture, use the scenarios and mental model to determine what questions or identifiable tasks the persona is starting with, what words she uses to describe it, and what her decision-making process looks like from that point on. Rough out a menu structure based on your first scenario for the primary persona, add to it using other scenarios, then assess and refine it using secondary personas and validation scenarios.

3. Redelmeier, D.A. and Tibshirani, R.J. "Association between cellular-telephone calls and motor vehicle collisions." *New England Journal of Medicine*, February 13, 1997.
4. McKnight, J. and McKnight, A.S. "The effect of cellular phone use on driver attention." National Public Services Research Institute/ AAA Foundation for Traffic Safety. <http://www.aaafoundation.org/resources/index.cfm?button=cellphone#a23>.
5. Joanne Harbluk in "Interview with Joanne Harbluk on safety and usability in vehicles." <http://www.carleton.ca/hotlab/hottopics/Articles/May2003-InterviewwithJoan.html>.

Game design is partly about the world you've envisioned and partly about what the users bring to that world.

You may find that your set of scenarios essentially defines the top-level menu. For example, imagine that your primary persona for a bank's automated phone system typically does the following things:

- Checks her balance before paying bills or planning a big expenditure (often)
- Transfers money between accounts (often)
- Orders checks when she's running low (sometimes)
- Investigates discrepancies on her statements (infrequently)

If you also have a secondary persona who is a new customer, he likely has a scenario involving opening a new account; this plus your primary persona's four activities are a logical place to start for your top-level menu, and their frequency implies where they belong in the sequence.

Next, focus on one scenario, such as checking the account balance. Once your primary persona says she wants to check her balance, she needs to select one account or listen to the balance for all accounts; which approach is more likely to work for her? If her balance is low, what next step will she take, and what additional information will she want? After you've run through each scenario, you'll have a partial menu structure. You can then throw validation scenarios at your tentative framework and adjust it as necessary.

GAMES

I'll confess that I've never personally designed a game, though I've had multiple game designers ask me how personas and scenarios apply. So, here are a few thoughts on the subject, for what they're worth.

Game design is partly about the story and the world you've envisioned and partly about what the users (who play the main characters) bring to that world: Just imagine how *The Matrix* movie would have differed if Woody Allen had played Neo. For this reason, I believe personas and scenarios offer useful ways to engage a broad range of users. Suppose you were designing an action/adventure game for these personas:

- Jason, a methodical person who reads magazines from front to back and bought the animated Star Trek on DVD just so he'd have every series.
- Tony, who thinks of blowing up monsters as stress relief and is easily bored by action that's too slow.
- Lisa, who competes with her brother in everything from school grades to how fast she can brush her teeth.

You'd want to offer plenty of chases, explosions, and high-speed action for Tony, but you could hide weapons, supplies, or clues to a mystery in various levels for Jason. If Jason weren't among your personas, it might not be worthwhile to develop these details. You could make Lisa happy by showing when she's approaching or passing her brother's top score.

Scenarios seem applicable in predicting possible user behavior so you can decide what's possible within the confines of the game and how the game responds to various actions. If Tony encounters a monster, he'll attack it with guns blazing. What happens if Jason decides to dodge the monster in favor of going after a clue? Does it give chase or continue to guard the basement door? Answers to these and similar questions seem like they would be helpful in creating a rich experience.

Full Design Team: Iterate Form and Behavior Together

Once you've spent a day or two exploring a device design from both the hardware and interaction design perspectives, get the whole design team together to share what you've learned and decide where to go from here. Ideally the next steps



Figure 16.10. Reviewing work in progress.

involve detailing and improving on the existing directions(s), but if something you've learned makes a platform not viable, you'll need to pick up another direction from your earlier brainstorming or come up with something else. In Figures 16.10 and 16.11, for example, designers use a range of sketches as well as rough foam models to review and refine the hardware architecture.

Interaction designers and industrial designers may have different biases when it comes to the number, location, and types of input controls. Interaction designers are often most concerned with cognitive issues, such as how self-evident the device's behavior will be, while industrial designers may lean more toward visual simplicity and minimal physical effort. These differences should decrease the longer the two disciplines collaborate, but can be noticeable when each works with the other for the first time. Talking through the issues using the personas and goals for perspective resolves most disagreements. Generally, interaction design considerations should drive the number, type, behavior, and locations of controls while ergonomic and engineering considerations should guide their exact form. However, if skilled, reasonable people from all disciplines can't agree on a solution, chances are you need to keep looking for alternatives.



Figure 16.11. Using a foam model to refine control placement.

A great design that seems elegant and obvious is not necessarily obvious before it exists.

Once the essential form factor and input mechanisms are settled, each discipline can work slightly more independently if necessary; daily check-ins are still a great idea, but every few days can be workable.

Exercise

Design the interaction framework (and any hardware platform) for the LocalGuide or RoomFinder.

Typical Challenges in Designing the Framework

A great design that seems elegant and obvious is not necessarily obvious before it exists; simplicity is difficult to achieve. Although they strive for simplicity, most interaction designers are fascinated by complex systems, subtle distinctions in human behavior, and obscure details that seem irrelevant to other people. These traits are helpful to a point, but can get in the way when you're creating solutions.

Many first attempts at interaction frameworks—by designers of all skill levels—are more complex than they need to be. This is not surprising, given that designers who have just filled their brains with many research details must set them aside and focus on the largest issues. It's also a natural part of any design process; simple, elegant designs of any kind, from posters and packaging to complex machinery, are almost always the result of many iterations and the work of multiple people. If someone looks at your framework and says it's too complex, listen closely, because they're probably right.

Most often, this complexity is due to insufficient distillation, such as failing to recognize that two similar elements could be combined into one. Attempting to over-distill the design to reduce navigation can create excessive visual and cognitive work, though, so this is always a matter of balance. Complexity can also result from a designer's insistence on making some distinction most users don't care about, such as separating songs released on albums from songs released as singles in a list of music. Designing for edge cases too early in the process is another common culprit.

Broken or incomplete data models may not seem to cause trouble right away, but can lead to gaps in the design that are hard to fill later. Over-abstraction of the data object types can also make it hard for users to find their way around, and can make it hard to structure the interface.

Although a designers' mind-set during early concept development should be optimistic, it's also possible to overshoot reality by too much. A slightly ambitious design entices stakeholders to consider where they can stretch, but an overly ambitious one either causes despair (since they'll never be able to ship it) or makes them wonder why they hired you. Over time, you'll learn to read situations and determine how much to push, but checking in with your project owner just a few days into design should help you avoid going over a cliff.

Of course, any designer can obsess over a specific idea he thinks is cool, to the detriment of the rest of the design. Personas and scenarios help minimize this problem, but just about every designer has an occasional case of stubbornness. Sometimes it takes a whole design team weighing in to help deflect a designer from a problematic course. If that doesn't work, a room full of stakeholders can do the trick. Failing that, you have to fall back on a usability test (or worse, market feedback). The earlier the problem is caught, the cheaper it is to fix, both in terms of time and money and in terms of designer credibility; this is another reason frequent check-ins are worthwhile.

Project Management for Defining Platforms and Frameworks

Other than time management (which is discussed in Chapter 14) the essential project management challenge in framework definition is getting the right sort of feedback at the right time. On one hand, it's important to uncover major problems as early as possible. On the other, a design concept at this stage has a fragile existence; it's far too easy for stakeholders who are not accustomed to ambiguity to lose faith in the proposed design (or worse, in the design team) for the wrong reasons.

It's best to start by getting feedback from the most understanding audience (the rest of the design team), followed by the most knowledgeable external audience (such as the project owner, design engineer, and perhaps a subject matter expert), and finally the rest of the stakeholders (see Chapter 19 for more on that meeting). You might also be considering direct user feedback or usability testing.

Internal design team check-ins

Even though each design discipline can and should work separately on different aspects of the problem, frequent team check-ins are important because each team member may be able to improve on the work

The essential project management challenge in framework definition is getting the right sort of feedback at the right time.

of others, and because each needs to understand where the others are headed. The team lead needs to ensure that everyone is going in the same direction, that the direction is going to meet the expectations of the stakeholders, and that the work is at the right level of detail and quality.

One form of team check-in is centered on an internal milestone, such as reviewing a first draft of the framework or design language studies (covered in Chapter 18), developing an outline for a presentation, or some other activity that needs to be done by a certain date. For example, I know from leading a lot of projects that if a team doesn't have some coherent platform sketches and interaction design rectangles emerging by day two or three of the phase, they may need a little help.

Another is a more generic, "What's everyone up to?" affair that ideally happens on a daily basis, generally at the beginning or end of the day. These are often gatherings at someone's desk to see what she accomplished the previous day, or in a conference room to look at what's on the whiteboard. If anyone is falling behind or needs help with anything, this is a good opportunity to discuss it. Unscheduled check-ins may occur as needed if someone on the team needs help with a sticky problem and calls in one or more of the others.

Although someone is responsible for leading each aspect of the design and the accompanying narrative, the entire team should share responsibility for the entire design. If one person sees a design or communication solution that seems broken, he should ask the rest of the team to explain it, ask why the proposed solution is good, and offer a critique if the explanation doesn't address the concern. Visual designers should look in particular for opportunities to simplify the on-screen grid (the structure for laying out screens; see Chapter 21). Industrial designers should look for ergonomic and other issues with the type and placement of controls, as well as where there might be opportunities to improve interaction using physical

controls. Interaction designers should use personas and scenarios to assess visual and industrial design. Everyone should look for framework and narrative coherence, simplicity, and adherence to persona goals and behaviors.

Once one or more concepts start to emerge, start thinking about how stakeholders will react and what questions they'll have when they see sketches. If you know someone will ask a particular question, object that a pet idea isn't represented in the design, or raise some other concern, discuss how you'll respond. It's also likely that you'll have a few questions for stakeholders as you work through the design; make sure someone on the team is following up on these. E-mail is a great tool for tracking this sort of conversation; consider using a team e-mail alias so each designer sees the discussion with the client team and can judge what parts of it affect her work.

Project owner, SME, and design engineer review

Before you spend too much time on any direction, it's important to get business and technical perspectives on your work. This means scheduling an hour or two with your project owner, design engineer(s), and perhaps a subject matter expert or two if necessary; some design engineers (especially mechanical engineers) might have been involved already. This informal meeting usually happens after three to five days of design, depending on the complexity of the problem. A short framework phase may have just one such meeting before you share the work with other stakeholders; a longer phase might involve two. Whether you have one meeting or two, there are several things that are important to accomplish: You need to get a sanity check on the design, make sure the project owner and design engineer(s) are prepared to support the design direction in the larger stakeholder meeting, and prepare for any bombshells you expect other stakeholders to drop in that meeting.

DETERMINING WHOM TO INVITE

With just a few days to do design, what you can show at this point is bound to be incomplete and probably a bit wrong, so unless you're working within a product team that understands how to look at early design and has a lot of faith both in you and the design process, it's important to have a small and reasonably friendly audience for this first review. In most cases, it will only be a week or so more before you're ready to show something to a larger (and more skeptical) group.

The one person who absolutely has to see the work in progress before other stakeholders is the project owner; it would be disastrous for him to be surprised by your work in a public meeting. If he knows what the likely sticking points are going to be ahead of time, he can work to prepare individual stakeholders as necessary.

It's also a good idea to include the design engineer(s), provided you have someone who can play that role. A good DE is adept at assessing sketchy ideas and can not only give you helpful feedback about how difficult various parts of the design will be to engineer, but may also offer up some possibilities you weren't aware of that can improve the design. However, I have worked with some clients whose engineers were obstructionists, usually due to insufficient skills or to managers who did not allow them reasonable amounts of time for their work. It would be a bad idea to exclude engineers from your process for very long, but it can be helpful to get executive buy-in (and perhaps bring in additional engineers) to keep the design from "dying in committee." No matter what the culture is, though, the project will fail unless you find productive ways to collaborate with the engineering team.

Some subject matter experts can provide invaluable insights early in the process. The majority I have worked with, however, are uncomfortable with very ambiguous early sketches and are con-

vinced the design team can't possibly understand what they're doing; SMEs of this sort can do more harm than good. Consider the aptitudes and inclinations of any subject matter experts before showing them anything particularly rough. Some SMEs are best involved only when you share the design direction with the other stakeholders.

SETTING EXPECTATIONS

Regardless of who is attending the meeting, you need to make sure they know what to expect. Emphasize that the meeting is an informal walk-through of work in progress at the whiteboard. If you have any handouts, they'll just be copies of sketches. Tell your meeting participants that sketches are ambiguous, you've only considered a couple of high-level scenarios so far, and your answers to many questions might be, "We don't know yet." Make sure they understand what you need from them: expertise, a gut reaction about how it fits user and business needs, and a general sense of what about the design may pose challenges (political, technical, or otherwise).

PREPARING YOUR AGENDA AND MATERIALS

An informal check-in of this sort shouldn't take a lot of special effort to prepare for, but you need to spend a few minutes considering what you're going to show and in what sequence, as well as what questions you need answers to. If possible, put some drawings on the board before everyone arrives; this saves time and reduces the pressure you're under during the meeting (though some designers like to wow stakeholders with their on-the-spot whiteboard skills).

Make sure each team member has a set of meeting notes handy and everyone knows what he or she is responsible for in the meeting. Most often, the IxDG or IxDS talks through the scenarios as the IxDG draws on the whiteboard. The industrial designer typically talks through the hardware design progress, either with a sketch or a crude

physical prototype. The IxDS is responsible for capturing action items and responses to questions, though the need for this is often minimal. The visual designer, who may be mostly observing at this meeting, supports the design arguments from a visual design and brand perspective.

There are generally two narrative approaches to this sort of meeting. One is to start by describing the anatomy of the design in conceptual terms, then walk through scenarios. The other is to start with scenarios, then discuss anatomy in more detail. Which approach works better depends on the nature of the design problem, how different your solution is for the status quo, the tendencies of the audience, and your comfort with either approach. See Chapter 19 for more on this topic.

CONDUCTING THE MEETING

Before you get started, recap the expectations about what participants will and won't see and what you need from them. Ask them to tell you whether the design makes sense and seems likely to solve the business problems.

If there's a design engineer in the room, ask for an assessment of implementation difficulties. I usually start by saying something like, "First, tell us if you see anything that makes you want to scream." This acknowledges that you know you're pushing on constraints. Some engineers (who aren't temperamentally inclined to be design engineers, or whose skills aren't up to the job) might say, "We can't do that." What this really means is that they can't do that within the timeframe or other constraints they've been given (or it can mean they don't know how). Don't ask, "Is this technically feasible?" Almost anything is, given sufficient time and money. Instead, ask, "What would it take to build this?" Say that you're not looking for specific commitments about what's feasible in the allotted time, since you understand it will take some work to figure out just how hard certain things are. The business project owner,

who is also in the room, can then say whether he wants to rule anything out right away.

If this is your last (or only) meeting before the larger stakeholder review, talk about what you're planning for that meeting. Ask the project owner what questions and concerns she expects people to raise and discuss how you should handle them. If the project owner hasn't hosted a similar meeting before, describe what she can do to prepare certain stakeholders.

You might also want to review the section on conducting the more formal design vision meeting in Chapter 19; many of the questions asked at that meeting will also crop up during informal check-ins.

User feedback

I've had a number of clients over the years who wanted to conduct usability testing as soon as there was an approximate concept. The underlying motivation is usually a good one: "We all think it's good, but we should see if users agree." Although testing is a good idea for most products—and a must for some—the framework is almost always too early to conduct a test because there just isn't enough design detail for users to perform tasks using a sketch of the interface. Although you don't need to wait until design is completely finished to test some kinds of products, you do need more than squiggles or *lorem ipsum* fake text.

What you can sometimes do to get user feedback is present the design much as you would to stakeholders (see Chapter 19) and ask for reactions. This may or may not be worth your time, depending on how engaged and thoughtful your users tend to be. Consumers are unlikely to be helpful at this point, but demanding users of very specialized professional tools (who are essentially subject matter experts) may provide useful insights.

However, regardless of how sophisticated the users are, be prepared for some amount of negative response based on the low fidelity of the sketches. This is no problem if the response doesn't get back to stakeholders (don't count on it!) or if they know how to take such feedback. Perfectly normal comments that are probably due to ambiguity rather than design flaws can cause uneducated stakeholders to panic and want to abandon a good direction prematurely. Always prepare stakeholders for this issue before seeking any user feedback.

It's easier and more effective to get user feedback on a visual direction or approximate hardware form factor at this point, though either has limited value since users and customers will eventually respond to the entire product rather than isolated parts of the design; an iPod without its software is a pointless block of metal and plastic. Rather than spending time and money on focus groups, most designers just figure out who in the office is most like the target users and ask them for a 30-second impression, such as how comfortably a foam prototype fits their hands.

Summary

The image of the designer who magically brings forth brilliant ideas like Athena springing from the head of Zeus is as iconic—and as unrealistic—as the image of the programmer/inventor tinkering in his garage and coming up with an overnight success. Does it happen? Maybe, but not to most designers, and never on a deadline. Design absolutely takes creativity and a dash of inspiration, but it also takes teamwork, iteration, and a lot of thought. Good process makes that easier. Any process that reliably and quickly gets you good results (i.e., good design that stakeholders can understand, believe in, and build) is a fine process; the one presented here is probably not the only one, but has been proven to work for numerous designers in a wide range of situations.

Whether or not the nuances described here work for you, I've yet to work with any designer who didn't benefit from implementing some core concepts:

- Treat the user experience as one design problem, whether it includes software, hardware, or services; you can break the problem into parts, but only if you keep bringing those parts back together to make sure they still fit.
 - Define your data model early, since a user's data helps define her tools and the structure of her environment.
 - Take some time to translate your functional needs into functional elements before you start to draw; don't worry about the tiny details, but use the exercise to help identify good design opportunities and important business trade-offs early on.
 - Regardless of what you're designing, use scenarios to guide ideation, iteration, and assessment. Don't be afraid to judge a design by other criteria, but always go back to the scenarios.
 - Get early feedback from the sources who are most likely to have the right information and know how to respond to work in progress.
-

DESIGNING FOR THE DIGITAL AGE

Whether you're designing consumer electronics, medical devices, enterprise Web apps, or new ways to check out at the supermarket, today's digitally-enabled products and services provide both great opportunities to deliver compelling user experiences and great risks of driving your customers crazy with complicated, confusing technology.

Designing successful products and services in the digital age requires a team with expertise in interaction design, visual design, industrial design, and other disciplines. It also takes the ability to come up with the big ideas that make a desirable product or service, as well as the skill and perseverance to execute on the thousand small ideas that get your design into the hands of users. It requires expertise in project management, user research, and consensus-building. This comprehensive volume addresses all of these and more.

"Kim's book is nothing less than a complete handbook for an entire profession. Kim's unique background in the practice, pedagogy, and epistemology of the design business has given her the experience needed to write the ultimate 'how-to' book. Every step in this fascinating and multi-faceted discipline is described in detail in simple, readable prose, richly illustrated with examples taken from real products, real clients, and real design problems. This book is comprehensive in its scope, exhaustive in its depth, authoritative in its practice, and priceless in its wisdom. I've no doubt that this will become the most dog-eared, annotated and worn-from-many-readings volume in your library."

ALAN COOPER

Bestselling author of *The Inmates Are Running the Asylum* and *About Face 3: The Essentials of Interaction Design*

"Kim is one of the brightest minds in the world of user experience design. Her work on Goal-Directed Design and persona development has set a standard."

JARED SPOOL

Founding Principal, User Interface Engineering

Kim Goodwin is VP Design and General Manager at Cooper, where she leads both an integrated practice of interaction, visual, and industrial designers and the development of the acclaimed Cooper U design curriculum. Kim knows the design world from multiple angles; she started as an in-house and freelance designer and spent several years as an in-house creative director before joining Cooper 11 years ago. Kim has led projects involving a tremendous range of design problems, including Web sites, complex analytical and enterprise applications, phones, medical devices, services, and even organizations. Her clients and employers have included everything from one-man startups to the world's largest companies, as well as universities and government agencies. This range of experience and a passion for teaching have led to Kim's popularity as an author and as a speaker at conferences and companies around the world.

RESOURCES & DISCUSSION AT:

www.designingforthedigitalage.com

Shelving Categories:

COMPUTERS/User Interfaces

DESIGN

\$69.99 US

\$83.99 CAN

978-0-470-22910-1

9780470229101

Sample
Chapter

cooper
www.cooper.com

 **WILEY**
Publishers Since 1807