cooper
www.cooper.com

4TH EDITION

ABOUT FACE

THE ESSENTIALS OF INTERACTION DESIGN

THE COMPLETELY UPDATED CLASSIC ON CREATING DELIGHTFUL USER EXPERIENCES

Alan Cooper, Robert Reimann, David Cronin, Chris Noessel

WILEY

About Face

The Essentials of Interaction Design

Fourth Edition

Alan Cooper
Robert Reimann
David Cronin
Christopher Noessel
with Jason Csizmadi
and Doug LeMoine

WILEY

About Face: The Essentials of Interaction Design, Fourth Edition

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published by John Wiley & Sons, Inc., Indianapolis, Indiana
Published simultaneously in Canada

ISBN: 978-1-118-76657-6
ISBN: 978-1-118-76640-8 (ebk)
ISBN: 978-1-118-76658-3 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2014930411

For Sue, my best friend through all the adventures of life. —Alan

For Alex and Max, and for Julie. —Robert

For Jasper, Astrid, and Gretchen. —David

For Ben and Miles, for your patience and inspiration. —Christopher

And for all the designers and engineers in our industry who are helping to imagine and build a better future.

ABOUT THE AUTHORS

Alan Cooper has been a pioneer in the software world for more than 40 years, and he continues to influence a new generation of developers, entrepreneurs, and user experience professionals.

Alan started his first company in 1976 and created what has been called "the first serious business software for microcomputers." In 1988, he invented a dynamically extensible visual programming tool and sold it to Bill Gates, who released it to the world as Visual Basic. This accomplishment earned Alan the sobriquet "The Father of Visual Basic."

In 1992, Alan and his wife, Sue, cofounded the first interaction design consulting firm, Cooper. By 1997, Cooper had developed a set of core design methods now used across the industry. Personas, which Alan invented and then popularized in his two best-selling books, *About Face* and *The Inmates Are Running the Asylum*, are employed almost universally by user experience practitioners.

Today, Alan continues to advocate for more humane technology from his farm in the rolling hills north of San Francisco.

Robert Reimann has spent over 20 years pushing the boundaries of digital products as a designer, writer, strategist, and consultant. He has led dozens of desktop, mobile, web, and embedded design projects in consumer, business, scientific, and professional domains for both startups and Fortune 500 companies alike.

One of the first designers at Cooper, Robert led the development and refinement of many of the Goal-Directed Design methods described in this book. In 2005, he became the founding president of IxDA, the Interaction Design Association (`www.ixda.org`). He has led user experience teams at Cooper, Bose, frog, and Sonos, and is currently Principal Interaction Designer at PatientsLikeMe.

David Cronin is a Design Director at GE and a member of GE's Design and Experience studio leadership team. Prior to that, he was Director of Interaction Design at Smart Design's San Francisco studio and a former Managing Director of Interaction Design at Cooper.

David has helped design products to serve the needs of surgeons, museum visitors, investment portfolio managers, nurses, drivers, dentists, financial analysts, radiologists, field engineers, manufacturing planners, marketers, videographers, and people with chronic diseases. During his time at Cooper, he contributed substantially to the principles, patterns, and practices of Goal-Directed Design.

Christopher Noessel designs products, services, and strategy for health, financial, and consumer domains as Cooper's first Design Fellow. He has helped visualize the future of counterterrorism, built prototypes of new technologies for Microsoft, and designed telehealth devices to accommodate the crazy facts of modern health care.

Prior to working at Cooper, Chris cofounded a small interaction design agency, where he developed exhibitions and environments for museums. He was also Director of Information Design at marchFIRST, where he helped establish the Interaction Design Center of Excellence. In 2012, he coauthored *Make It So: Interaction Design Lessons from Science Fiction*. He publishes regularly to the Cooper Journal and continues to speak and teach around the world.

CREDITS

Acquisitions Editor
Mary James

Project Editor
Adaobi Obi Tulton

Technical Editor
Christopher Noessel

Senior Production Editor
Kathleen Wisor

Copy Editor
Gayle Johnson

Manager of Content Development and Assembly
Mary Beth Wakefield

Director of Community Marketing
David Mayhew

Marketing Manager
Carrie Sherrill

Business Manager
Amy Knies

Vice President and Executive Group Publisher
Richard Swadley

Associate Publisher
Jim Minatel

Project Coordinator, Cover
Todd Klemme

Compositor
Maureen Forys, Happenstance Type-O-Rama

Proofreader
Nancy Carrasco

Indexer
Johnna VanHoose Dinse

Cover Designer
Jason Csizmadi

Art Director
Jason Csizmadi

ACKNOWLEDGMENTS

The other authors and I are extremely proud of this fourth edition. We and many others worked hard to update, illustrate, and improve it, but without the efforts of Mary James at Wiley, it would not exist. Mary's quiet, insistent support made the mountainous project seem imminently doable. Once we began, Mary continued to marshal necessary resources and encourage everyone to make this book a reality. Mary recruited some imposing talent at Wiley to wrangle the many moving pieces of a project such as this. Project editor Adaobi Obi Tulton has done a fabulous job making the gears mesh and the wheels turn smoothly. Marketing manager Ashley Zurcher's early support for the project helped us get the paper, color, graphics, and promotion we aspired to. Her enthusiasm gave us confidence to do our best.

Ever since Apple changed the landscape of consumer computing with its smartphones and tablet computers, Robert Reimann has been gently nudging me to update *About Face*. When I turned the tables on him, asking him to do the lion's share of the writing, he unhesitatingly agreed. Most of the changes in this book are his, along with most of the credit. Chris Noessel generously agreed to act as technical editor, and his contributions can be felt throughout the manuscript. The capable writing of Dave Cronin and Doug LeMoine added much to the depth and completeness of this new edition.

Visually, this edition is far advanced over its predecessors. Many individual members of the Cooper design staff contributed their talents. Supremely talented visual designer Jason Csizmadi led the effort, organizing and coordinating, not to mention drawing and Photoshopping late into the night. You can see the beautiful fruits of his labor on these pages, from (and including) cover to cover. Other designers whose work is featured herein include Cale Leroy, Christina Beard, Brendan Kneram, and Gritchelle Fallesgon, along with Martina Maleike, James Laslavic, Nick Myers, and Glen Davis.

Other Cooperistas contributed their care and talent at many points during the yearlong writing effort. In particular, big props to Jayson McCauliff, Kendra Shimmell, and Susan Dybbs, who helped immensely in keeping the project on track as work and life brought their distractions. Similarly, Steve Calde and Karen Lemen were always ready to help as the project evolved.

We would also like to thank the following colleagues and Cooper designers for their contributions to this and previous editions, to whom we are greatly indebted: Kim Goodwin, who contributed significantly to the development and expression of the concepts and methods described in Part I; Hugh Dubberly for his help in developing the principles described at the end of Chapter 7 and for his assistance in clarifying the Goal-Directed process with early versions of the diagrams found in Chapter 1; Gretchen Anderson and Elaine Montgomery for their contributions on user and market research in Chapter 2; Rick Bond for his many insights into usability testing featured in Chapter 5; Chris Weeldreyer for his insights into the design of embedded systems in Chapter 19; Wayne Greenwood for his contributions on control mapping in Chapter 12; and Nate Fortin and Nick Myers for their contributions on visual interface design and branding in Chapter 17. We would also like to thank Elizabeth Bacon, Steve Calde, John Dunning, David Fore, Nate Fortin, Kim Goodwin, Wayne Greenwood, Noah Guyot, Lane Halley, Ernest Kinsolving, Daniel Kuo, Berm Lee, Tim McCoy, Elaine Montgomery, Nick Myers, Ryan Olshavsky, Angela Quail, Suzy Thompson, and Chris Weeldreyer for their contributions to the Cooper designs and illustrations featured in this book. It should also be noted that the parts of Chapter 3 concerned with cognitive processing originally appeared in an article by Robert Reimann on `UXMatters.com` and are used with permission.

We are grateful to clients David West at Shared Healthcare Systems, Mike Kay and Bill Chang at Fujitsu Softek, John Chaffins at CrossCountry, Chris Twogood at Teradata, and Chris Dollar at McKesson for granting us permission to use examples from the Cooper design projects featured in this book. We also want to thank the many other clients who had the vision and foresight to work with us and support us in their organizations.

We would like to acknowledge the following authors and industry colleagues who have influenced or clarified our thinking over the years: Christopher Alexander, Edward Tufte, Kevin Mullet, Victor Papanek, Donald Norman, Larry Constantine, Challis Hodge, Shelley Evenson, Clifford Nass, Byron Reeves, Stephen Pinker, and Terry Swack.

Thanks to my agent, Bill Gladstone, who again created a successful business framework for all this to happen.

As always, the most long-suffering contributors to a work like this one are the authors' families. We appreciate how much our partners and children have sacrificed so that we could create this book.

CONTENTS

FOREWORD

I began working on the first edition of this book 20 years ago. Fittingly, I wrote a manifesto—a challenge to frustrated practitioners to step forward and begin creating software that users would love. In those days, few designers and precious little software didn't make your head hurt to use. Strong measures were called for.

Today, the technology landscape is much different; consequently, this fourth edition is also much different. In 1994 the state of the art of personal software was an address book or spreadsheet. Today, the digitization of all forms of media has put consumers neck-deep in technology. Powerful handheld apps are now in the hands of amateurs and non-technical users—apps for music listening and production; for photography, video, news, and communications; for home security and environmental control; for health, fitness, and personal tracking; for games and education; and for shopping.

Over a billion people have a full-fledged computer in their pocket and access to millions of applications and websites. The value of making these user-facing products easier to understand and use is clear. We interaction designers have earned our seat at the table and are well established as an integral part of teams that produce successful, widely used digital products.

The primary challenge of the first two decades of interaction design practice was to invent the process, tools, roles, and methods needed to succeed. Now that we have demonstrated our success, our relationship to others in our organization is changing. Each of these best practices is now evolving as we integrate our skills more deeply into our teams. Specifically, we need to work more effectively with business people and developers.

Twenty years ago, developers too had to fight for inclusion and relevance. Although firmly embedded in the corporate hierarchy, they lacked credibility and authority. As consumer digitization increased, developers grew dissatisfied as the agents of users' misery. They knew they could do better.

The agile movement and, more recently, the growth of lean practices are each efforts by software developers to have more influence on their own destiny. Developers were just as frustrated as designers at the sorry state of digital interaction, and they wanted improvements. They realized that the software construction process had been modeled after industrial archetypes that didn't suit the new digital medium.

A few brave developers began experimenting with unorthodox methods of creating software in smaller increments while maintaining closer contact with their clients. They wanted to avoid lengthy development efforts—"death marches"—that resulted in unhappy users. They also were motivated by a natural desire to find a process that more reliably resulted in better products that they could be proud of.

Although each variant has its adherents and detractors, the course of software development has been forever altered by these new approaches. The notion that the old ways weren't working well is now widely accepted, and the quest for new methods continues.

This new self-awareness in the development community is a huge opportunity for interaction designers. Before, developers saw designers as competing for scarce resources. Now, developers see interaction designers as useful helpers, able to contribute skills, experience, and viewpoints that developers cannot. As developers and designers have begun to cooperate instead of competing, they have discovered that their powers are multiplied by working side by side.

Every practitioner—developer as well as designer—wants to create a product they can be proud of. To improve outcomes, both groups have been rethinking the entire development process, demanding better tools, better guidance, and better access. Historically, though, developers and interaction designers have pursued their common goal separately, developing tools and processes that work in separate silos. The two practices are quite different in many ways, and neither will work in subservience to the other. The challenge, then, is to learn how they can work together, effectively, successfully, in mutual support.

At the most forward-looking companies, you can already see this happening: Developers and designers sit next to each other, working cooperatively and collaboratively. When designers and developers—and the many other practitioners working with them—collaborate fully, the result is far better than any other method we've tried. The speed with which the work gets done is much greater and the quality of the end product much higher. And users are more delighted.

On the business side, executives often misunderstand the role of interaction design. It sometimes seems that the only place where it is truly understood is in tiny start-ups. Although larger companies may have many interaction designers on staff, managers persistently fail to incorporate their design expertise into the process until it's too late.

All the design skill and process in the world won't succeed unless the corporate culture supports interaction design and its objectives. Apple isn't a paragon of user experience because of its employees' design skills, but because Steve Jobs, its former (and legendarily autocratic) leader, was a tireless advocate of the power of design.

Few companies have leaders as bold as Jobs. Those that do tend to be small start-ups. You will find it difficult to convince business people of the value of collaborative design work. But each year will see more success stories—more proof of the value of this new work paradigm. I remember when Apple and Microsoft, not to mention Google and Facebook, were tiny start-ups with many doubters.

The two opportunities that face interaction designers today are finding, or creating, advocates on the business side, and learning how to collaborate with the newly sympathetic development community.

What is indisputable is the awesome power of interaction design: giving technology users a memorable, effective, easy, and rewarding experience as they work, play, and communicate.

—Alan Cooper

INTRODUCTION TO THE FOURTH EDITION

This book is about **interaction design**—the practice of designing interactive digital products, environments, systems, and services. Like most design disciplines, interaction design is concerned with form. However, first and foremost, interaction design focuses on something that traditional design disciplines do not often explore: the design of *behavior.*

Most design *affects* human behavior: Architecture is concerned with how people use physical space, and graphic design often attempts to motivate or facilitate a response. But now, with the ubiquity of silicon-enabled products—from computers to cars and phones to appliances—we routinely create products that *exhibit* complex behavior.

Take a product as basic as an oven. Before the digital age, it was quite simple to operate an oven: You simply turned a single knob to the correct position. There was one position for off, and each point along which the knob could turn resulted in a unique temperature. Every time the knob was turned to a given position, *the exact same thing happened.* You could call this a "behavior," but it is certainly a simple one.

Compare this to modern ovens with microprocessors, LCD screens, and embedded operating systems. They are endowed with buttons labeled with non-cooking-related terms such as Start, Cancel, and Program, as well as the perhaps more expected Bake and Broil. What happens when you press any one of these buttons is much less predictable than what happened when you turned the knob on your old gas range. In fact, the outcome of pressing one of these buttons often depends on the oven's operational state, as well as the sequence of buttons you press before pressing the last one. This is what we mean by *complex behavior.*

This emergence of products with complex behavior has given rise to a new discipline. Interaction design borrows theory and technique from traditional design, usability, and engineering disciplines. But it is greater than the sum of its parts, with its own unique methods and practices. And to be clear, interaction design is very much a *design* discipline, quite different from science and engineering. Although it employs an analytical approach when required, interaction design is also very much about synthesis and imagining things as they might be, not necessarily as they currently are.

Interaction design is an inherently humanistic enterprise. It is concerned most significantly with satisfying the needs and desires of the people who will interact with a product or service. These goals and needs can best be understood as *narratives*—logical and emotional progressions over time. In response to these user narratives, digital products must express behavioral narratives of their own, appropriately responding not only at the levels of logic and data entry and presentation, but also at a more human level.

In this book we describe a particular approach to interaction design that we call the Goal-Directed Design method. We've found that when designers focus on people's goals—the reasons why they use a product in the first place—as well as their expectations, attitudes, and aptitudes, they can devise solutions that people find both powerful and pleasurable to use.

As even the most casual observer of developments in technology must have noticed, interactive products quickly can become complex. Although a mechanical device may be capable of a dozen visible states, a digital product may be capable of *thousands* of different states (if not more!). This complexity can be a nightmare for users and designers alike. To tame this complexity, we rely on a systematic and rational approach. This doesn't mean that we don't also value and encourage inventiveness and creativity. On the contrary, we find that a methodical approach helps us clearly identify opportunities for revolutionary thinking and provides a way to assess the effectiveness of our ideas.

According to Gestalt Theory, people perceive a thing not as a set of individual features and attributes, but as a unified whole in a relationship with its surroundings. As a result, it is impossible to effectively design an interactive product by decomposing it into a list of atomic requirements and coming up with a design solution for each. Even a relatively simple product must be considered in totality and in light of its context in the world. Again, we've found that a methodical approach helps provide the holistic perspective necessary to create products that people find useful and engaging.

A Brief History of Interaction Design

In the late 1970s and early 1980s, a dedicated and visionary set of researchers, engineers, and designers in the San Francisco Bay area were busy inventing how people would interact with computers in the future. At Xerox Parc, SRI, and eventually Apple Computer, people had begun discussing what it meant to create useful and usable "human interfaces" to digital products. In the mid-1980s, two industrial designers, Bill Moggridge and Bill Verplank, were working on the first laptop computer, the GRiD Compass. They coined the term *interaction design* for what they were doing. But it would be another 10 years before other designers rediscovered this term and brought it into mainstream use.

When *About Face* was first published in August 1995, the landscape of interaction design was still a frontier wilderness. A small cadre of people brave enough to hold the title user interface designer operated in the shadow of software engineering, rather like the tiny, quick-witted mammals that lurked in the shadows of hulking tyrannosaurs. "Software design," as the first edition of *About Face* called it, was poorly understood and underappreciated. When it was practiced at all, it was usually practiced by developers. A handful of uneasy technical writers, trainers, and product support people, along with a rising number of practitioners from another nascent field—usability—realized that something needed to change.

The amazing growth and popularity of the web drove that change, seemingly overnight. Suddenly, the phrase "ease of use" was on everyone's lips. Traditional design professionals, who had dabbled in digital product design during the short-lived popularity of "multimedia" in the early '90s, leapt to the web en masse. Seemingly new design titles sprang up like weeds: information designer, information architect, user experience strategist, interaction designer. For the first time, C-level executive positions were established to focus on creating user-centered products and services, such as chief experience officer. Universities scrambled to offer programs to train designers in these disciplines. Meanwhile, usability and human-factors practitioners also rose in stature and are now recognized as advocates for better-designed products.

Although the web knocked back interaction design idioms by more than a decade, it inarguably placed user requirements on the radar of the corporate world permanently. After the second edition of *About Face* was published in 2003, the *user experience* of digital products became front-page news in the likes of *Time* and *BusinessWeek*. And institutions such as Harvard Business School and Stanford recognized the need to train the next generation of MBAs and technologists to incorporate design thinking into their business and development plans. People are tired of new technology for its own sake. Consumers are sending a clear message that they want *good* technology: technology that is *designed* to provide a compelling and effective user experience.

In August 2003, five months after the second edition of *About Face* proclaimed the existence of a new design discipline called *interaction design*, Bruce "Tog" Tognazzini made an impassioned plea to the nascent community to create a nonprofit professional organization. A mailing list and steering committee were founded shortly thereafter by Challis Hodge, David Malouf, Rick Cecil, and Jim Jarrett.

In September 2005, IxDA, the Interaction Design Association (www.ixda.org), was incorporated. In February 2008, less than a year after the publication of the third edition of *About*

Face, IxDA hosted its first international design conference, Interaction08, in Savannah, Georgia. In 2012, IxDA presented its first annual Interaction Awards for outstanding designs submitted from all over the world. IxDA currently has over 70,000 members living in more than 20 countries. We're pleased to say that interaction design has truly come into its own as both a design discipline and a profession.

IxD and User Experience

The first edition of *About Face* described a discipline called software design and equated it with another discipline called user interface design. Of these two terms, user interface design has enjoyed more longevity. We still use it occasionally in this book, specifically to connote the layout of widgets on a screen. However, this book discusses a discipline broader than the design of user interfaces. In the world of digital technology, form, function, content, and behavior are so inextricably linked that many of the challenges of designing an interactive product go right to the heart of what a digital product *is* and *does*.

As we've discussed, interaction designers have borrowed practices from more-established design disciplines but also have evolved beyond them. Industrial designers have attempted to address the design of digital products. But like their counterparts in graphic design, their focus traditionally has been on the design of static form, not the design of interactivity, or form that changes and reacts to input over time. These disciplines do not have a language with which to discuss the design of rich, dynamic behavior and changing user interfaces.

A term that has gained particular popularity in the last decade is *user experience (UX) design*. Many people have advocated for the use of this term as an umbrella under which several different design and usability disciplines collaborate to create products, systems, and services. This is a laudable goal with great appeal, but it does not in itself directly address the core concern of interaction design as discussed in this book: how to specifically *design the behavior* of complex interactive systems. It's useful to consider the similarities and synergies between creating a customer experience at a physical store and creating one with an interactive product. However, we believe specific methods are appropriate for designing for the world of bits.

We also wonder whether it is truly possible to *design* an experience. Designers of all stripes hope to manage and *influence* people's experiences, but this is done by carefully manipulating the variables intrinsic to the medium at hand. A graphic designer creating a poster arranges fonts, photos, and illustrations to help create an experience; a furniture designer working on a chair uses materials and construction techniques to help create an experience; an interior designer uses layout, lighting, materials, and even sound to help create an experience.

Extending this thinking into the world of digital products, we find it useful to think that we influence people's experiences by designing the mechanisms for interacting with a product. Therefore, we have chosen Moggridge's term *interaction design* (now abbreviated by many in the industry as IxD) to denote the kind of design this book describes.

Of course, often a design project requires careful attention to the orchestration of a number of design disciplines to achieve an appropriate user experience, as shown in Figure 1. It is in these situations that we feel the term *user experience design* is most applicable.

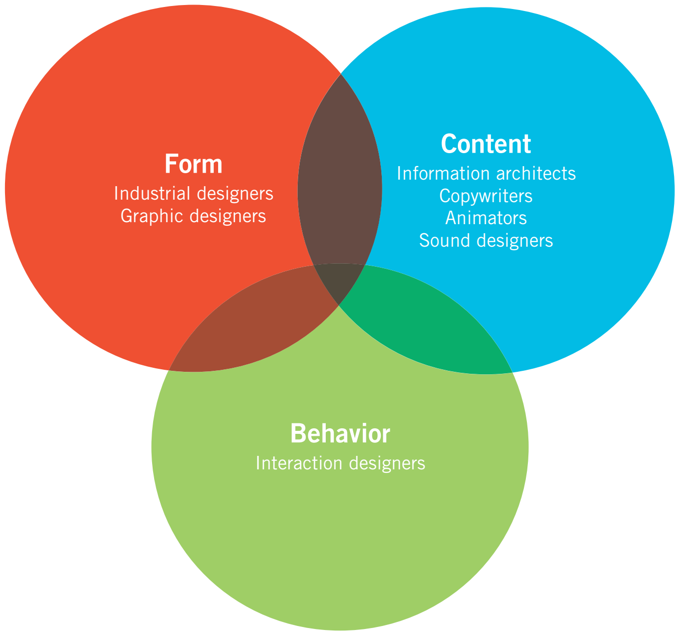


Figure 1: User experience (UX) design has three overlapping concerns: form, behavior, and content. Interaction design focuses on the design of behavior but also is concerned with how that behavior relates to form and content. Similarly, information architecture focuses on the structure of content but also is concerned with behaviors that provide access to content and how the content is presented to the user. Industrial design and graphic design are concerned with the form of products and services but also must ensure that their form supports use, which requires attention to behavior and content.

What This Book Is and What It Is Not

In this book, we attempt to give you effective and practical tools for interaction design. These tools consist of *principles*, *patterns*, and *processes*. Design *principles* encompass broad ideas about the practice of design, as well as rules and hints about how to best use specific user interface and interaction design idioms. Design *patterns* describe sets of interaction design idioms that are common ways to address specific user requirements and design concerns. Design *processes* describe how to understand and define user requirements, how to then translate those requirements into the framework of a design, and finally how to best apply design principles and patterns to specific contexts.

Although books are available that discuss design principles and design patterns, few books discuss design processes, and even fewer discuss all three of these tools and how they work together to create effective designs. Our goal was to create a book that unites all three of these tools. While helping you design more effective and useful dialog boxes and menus, this book also helps you understand how users comprehend and interact with your digital product. In addition, it helps you understand how to use this knowledge to drive your design.

Integrating design principles, processes, and patterns is the key to designing effective product interactions and interfaces. There is no such thing as an objectively good user interface. Quality depends on the context: who the user is, what she is doing, and what her motivations are. Applying a set of one-size-fits-all principles makes user interface creation *easier*, but it doesn't necessarily make the end result *better*. If you want to create good design solutions, there is no avoiding the hard work of really understanding the people who will actually interact with your product. Only then is it useful to have at your command a toolbox of principles and patterns to apply in specific situations. We hope this book will both encourage you to deepen your understanding of your product's users and teach you how to translate that understanding into superior product designs.

This book does *not* attempt to present a style guide or set of interface standards. In fact, you'll learn in Chapter 17 about the limitations of such tools. That said, we hope that the process and principles described in this book are compatible companions to the style guide of your choice. Style guides are good at answering *what* but generally are weak at answering *why*. This book attempts to address these unanswered questions.

This book discusses four main steps to designing interactive systems: researching the domain, understanding the users and their requirements, defining a solution's framework, and filling in the design details. Many practitioners would add a fifth step: *validation*—testing a solution's effectiveness with users. This is part of a discipline widely known as *usability*.

Although this is an important and worthwhile component to many interaction design initiatives, it is a discipline and practice in its own right. We briefly discuss design validation and usability testing in Chapter 5. We also urge you to refer to the significant and ever-growing body of usability literature for more detailed information about conducting and analyzing usability tests.

How This Book Is Structured

This book is organized in a way that presents its ideas in an easy-to-use reference structure. The book is divided into three parts:

- Part I introduces and describes the Goal-Directed Design process in detail, as well as building design teams and integrating with project teams.
- Part II deals with high-level interaction design principles that can be applied to any interaction design problem on almost any platform.
- Part III covers lower-level and platform-specific interface design principles and idioms for mobile, desktop, the web, and more.

Changes Since the Third Edition

In June 2007, just two months after the third edition of *About Face* was published, Apple changed the digital landscape forever with the introduction of the iPhone and iOS. In 2010 Apple followed with the first commercially successful tablet computer, the iPad. These touch-based, sensor-laden products and the competitors that followed in their footsteps have added an entirely new lexicon of idioms and design patterns to the language of interaction. This fourth edition of *About Face* addresses these and other modern interaction idioms directly.

This new edition retains what still holds true, updates things that have changed, and provides new material reflecting how the industry has changed in the last seven years. It also addresses new concepts we have developed in our practices to address the changing times.

Here are some highlights of the major changes you will find in this edition of *About Face*:

- The book has been reorganized and streamlined to present its ideas in a more concise and easy-to-use structure and sequence. Some chapters have been rearranged for better flow, others have been merged, a few have been condensed, and several new chapters have been added.
- Terminology and examples have been updated to reflect the current state of the art in the industry. The text as a whole has been thoroughly edited to improve clarity and readability.
- Part I describes the Goal-Directed Design process in additional detail and more accurately reflects the most current practices at Cooper. It also includes additional information on building a design team and integrating with development and project teams.
- Part II has been significantly reorganized to more clearly present its concepts and principles, and includes newly updated information on integrating visual design.

Part III has been extensively rewritten, updated, and extended to reflect new mobile and touch-based platforms and interaction idioms. It also offers more detailed coverage of web interactions and interactions on other types of devices and systems. We hope you will find that these additions and changes make *About Face* a more relevant and useful reference than ever before.

Examples Used in This Book

This book is about designing all kinds of interactive digital products. However, because interaction design has its roots in software for desktop computers, and the vast majority of today's PCs run Microsoft Windows, there is certainly a bias in the focus of our discussions of desktop software. Similarly, the first focus of many developers of native mobile apps is iOS, so the bulk of our mobile examples are from this platform.

Having said that, most of the material in this book transcends platform. It is equally applicable across platforms—Mac OS, Windows, iOS, Android, and others. The majority of the material is relevant even for more divergent platforms such as kiosks, embedded systems, 10-foot interfaces, and the like.

A number of the desktop examples in this book are from the Microsoft Office suite and from Adobe Photoshop and Illustrator. We have tried to stick with examples from these mainstream applications for two reasons. First, you're likely to be at least somewhat familiar with the examples. Second, it's important to show that the user interface design of even the most finely honed products can be significantly improved with a goal-directed approach. This edition also contains many new examples from mobile apps and the web, as well as several more-exotic applications.

A few examples in this new edition come from now-moribund software or OS versions. These examples illustrate particular points that we felt were useful enough to retain in this edition. The vast majority of examples are from contemporary software and OS releases.

Who Should Read This Book

While the subject matter is broadly aimed at students and practitioners of interaction design, anyone concerned about users interacting with digital technology will gain insights from reading this book. Developers, designers of all stripes involved with digital product design, usability professionals, and project managers will all find something useful in this book. If you've read earlier editions of *About Face* or *The Inmates*

Are Running the Asylum (Sams, 2004), you will find new and updated information about design methods and principles here.

We hope this book informs you and intrigues you. Most of all, we hope it makes you think about the design of digital products in new ways. The practice of interaction design is constantly evolving, and it is new and varied enough to generate a wide variety of opinions on the subject. If you have an interesting opinion, or if you just want to talk, we'd be happy to hear from you. E-mail us at `alan@cooper.com`, `rmreimann@gmail.com`, `davcron@gmail.com`, or `chrisnoessel@gmail.com`.

PART

I

Goal-Directed Design

CH 1

A DESIGN PROCESS FOR DIGITAL PRODUCTS

This book has a simple premise: If we design and develop digital products in such a way that the people who use them can easily achieve their goals, they will be satisfied, effective, and happy. They will gladly pay for our products—and recommend that others do the same. Assuming that we can do so in a cost-effective manner, this will translate into business success.

On the surface, this premise seems obvious: Make people happy, and your products will be a success. Why, then, are so many digital products so difficult and unpleasant to use? Why aren't we all happy and successful when we use them? Why, despite the steady march of faster, cheaper, and more accessible technology, are we still so often frustrated?

The answer, in short, is the *absence of design* as a fundamental and equal part of the product planning and development process.

Design, according to industrial designer Victor Papanek, is *the conscious and intuitive effort to impose meaningful order*. We propose a somewhat more detailed definition of human-oriented design activities:

- Understanding the desires, needs, motivations, and contexts of people using products
- Understanding business, technical, and domain opportunities, requirements, and constraints

- Using this knowledge as a foundation for plans to create products whose form, content, and behavior are useful, usable, and desirable, as well as economically viable and technically feasible

This definition is useful for many design disciplines, although the precise focus on form, content, and behavior varies depending on what is being designed. For example, an informational website may require particular attention to *content*, whereas the design of a simple TV remote control may be concerned primarily with *form*. As discussed in the Introduction, interactive digital products are uniquely imbued with complex *behavior*.

When performed using the appropriate methods, design can, and does, provide the missing human connection in technological products. But most current approaches to the design of digital products don't work as advertised.

The Consequences of Poor Product Behavior

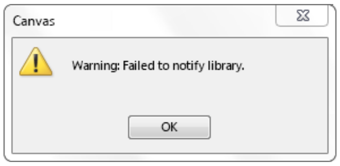
In the nearly 20 years since the publication of the first edition of *About Face*, software and interactive digital products have greatly improved. Many companies have begun to focus on serving people's needs with their products and are spending the time and money needed to support the design process. However, many more still fail to do so—at their peril. As long as businesses continue to focus solely on *technology* and *market data* while shortchanging design, they will continue to create the kind of products we've all grown to despise.

The following sections describe a few of the consequences of creating products that lack appropriate design and thus ignore users' needs and desires. How many of your digital products exhibit some of these characteristics?

Digital products are rude

Digital products often blame users for making mistakes that are not their fault, or should not be. Error messages like the one shown in Figure 1-1 pop up like weeds, announcing that the user has failed yet again. These messages also demand that the user acknowledge his failure by confirming it: OK.

Digital products and software frequently interrogate users, peppering them with a string of terse questions that they are neither inclined nor prepared to answer: "Where did you hide that file?" Patronizing questions like "Are you sure?" and "Did you really want to delete that file, or did you have some other reason for pressing the Delete key?" are equally irritating and demeaning.


Canvas

Warning: Failed to notify library.

Figure 1-1: Thanks for sharing. Why didn't the application notify the library? Why did it want to notify the library? Why is it telling us? And what are we OKing, anyway? It is not OK that the application failed!

Our software-enabled products also fail to act with a basic level of decency. They forget information we tell them and don't do a very good job of anticipating our needs. Even the iPhone—generally the baseline for good user experience on a digital device—doesn't anticipate that someone might not want to be pestered with a random phone call when he is in the middle of a business meeting *that is sitting right there in the iPhone's own calendar*. Why can't it quietly put a call that isn't from a family member into voicemail?

Digital products require people to think like computers

Digital products regularly assume that people are technology literate. For example, in Microsoft Word, if a user wants to rename a document she is editing, she must know that she must either close the document or use the "Save As…" menu command (and remember to delete the file with the old name). These behaviors are inconsistent with how a normal person thinks about renaming something; rather, they require that a person change her thinking to be more like the way a computer works.

Digital products are also often obscure, hiding meaning, intentions, and actions from users. Applications often express themselves in incomprehensible jargon that cannot be fathomed by normal users ("What is your SSID?") and are sometimes incomprehensible even to experts ("Please specify IRQ.").

Digital products have sloppy habits

If a 10-year-old boy behaved like some software apps or devices, he'd be sent to his room without supper. These products forget to shut the refrigerator door, leave their shoes in the middle of the floor, and can't remember what you told them only five minutes earlier. For example, if you save a Microsoft Word document, print it, and then try to close it, the application again asks you if you want to save it! Evidently the act of printing caused the application to think the document had changed, even though it did not. Sorry, Mom, I didn't hear you.

Software often requires us to step out of the main flow of tasks to perform functions that shouldn't require separate interfaces and extra navigation to access. Dangerous commands, however, are often presented right up front where users can accidentally trigger them. Dropbox, for example, sandwiches Delete between Download and Rename on its context menus, practically inviting people to lose the work they've uploaded to the cloud for safekeeping.

Furthermore, the appearance of software—especially business and technical applications—can be complex and confusing, making navigation and comprehension unnecessarily difficult.

Digital products require humans to do the heavy lifting

Computers and their silicon-enabled brethren are purported to be labor-saving devices. But every time we go out into the field to watch real people doing their jobs with the assistance of technology, we are struck by how much work they are forced to do simply to manage the proper operation of software. This work can be anything from manually copying (or, worse, retyping) values from one window into another, to attempting (often futilely) to paste data between applications that otherwise don't speak to each other, to the ubiquitous clicking and pushing and pulling of windows and widgets around the screen to access hidden functionality that people use every day to do their job.

The evidence is everywhere that digital products have a lot of explaining to do when it comes to their poor behavior.

Why Digital Products Fail

Most digital products emerge from the development process like a sci-fi monster emerging from a bubbling tank. Instead of planning and executing with a focus on satisfying the needs of the people who use their products, companies end up creating solutions that—while technically advanced—are difficult to use and control. Like mad scientists, they fail because they have not imbued their creations with sufficient humanity.

Why is this? What is it about the technology industry as a whole that makes it so inept at designing the interactive parts of digital products? What is so broken about the current process of creating software-enabled products that it results in such a mess?

There are four main reasons why this is the case:

- **Misplaced priorities** on the part of both product management and development teams
- **Ignorance about real users** of the product and what their baseline needs are for success
- **Conflicts of interest** when development teams are charged with both designing and building the user experience
- **Lack of a design process** that permits knowledge about user needs to be gathered, analyzed, and used to drive the development of the end experience

Misplaced priorities

Digital products come into the world subject to the push and pull of two often-opposing camps—marketers and developers. While marketers are adept at understanding and quantifying a marketplace opportunity, and at introducing and positioning a product within that market, their input into the product design process is often limited to lists of requirements. These requirements often have little to do with what users actually *need* or *desire* and have more to do with chasing the competition, managing IT resources with to-do lists, and making guesses based on market surveys—what people say they'll *buy*. (Contrary to what you might suspect, few users can clearly articulate their needs. When asked direct questions about the products they use, most tend to focus on low-level tasks or workarounds to product flaws. And, what they think they'll buy doesn't tell you much about how—or if—they will use it.)

Unfortunately, reducing an interactive product to a list of a hundred features doesn't lend itself to the kind of graceful orchestration that is required to make complex technology useful. Adding "easy to use" as a checklist item does nothing to improve the situation.

Developers, on the other hand, often have no shortage of input into the product's final form and behavior. Because they are in charge of construction, they decide exactly what gets built. And they too have a different set of imperatives than the product's eventual audience. Good developers are focused on solving challenging technical problems, following good engineering practices, and meeting deadlines. They often are given incomplete, myopic, confusing, and sometimes contradictory instructions and are forced to make significant decisions about the user experience with little time or knowledge of how people will actually use their creations.

Thus, the people who are most often responsible for creating our digital products rarely take into account the users' *goals*, needs, or motivations. At the same time, they tend to be highly reactive to market trends and technical constraints. This can't help but result in products that lack a coherent user experience. We'll soon see why goals are so important in addressing this issue.

The results of poor product vision are, unfortunately, digital products that irritate rather than please, reduce rather than increase productivity, and fail to meet user needs. Figure 1-2 shows the evolution of the development process and where, if at all, design has historically fit in. Most of digital product development is stuck in the first, second, or third step of this evolution, where design either plays no real role or becomes a surface-level patch on shoddy interactions—"lipstick on the pig," as one of our clients called it. The core activities in the design process, as we will soon discuss, should *precede* coding and testing to ensure that products truly meet users' needs.

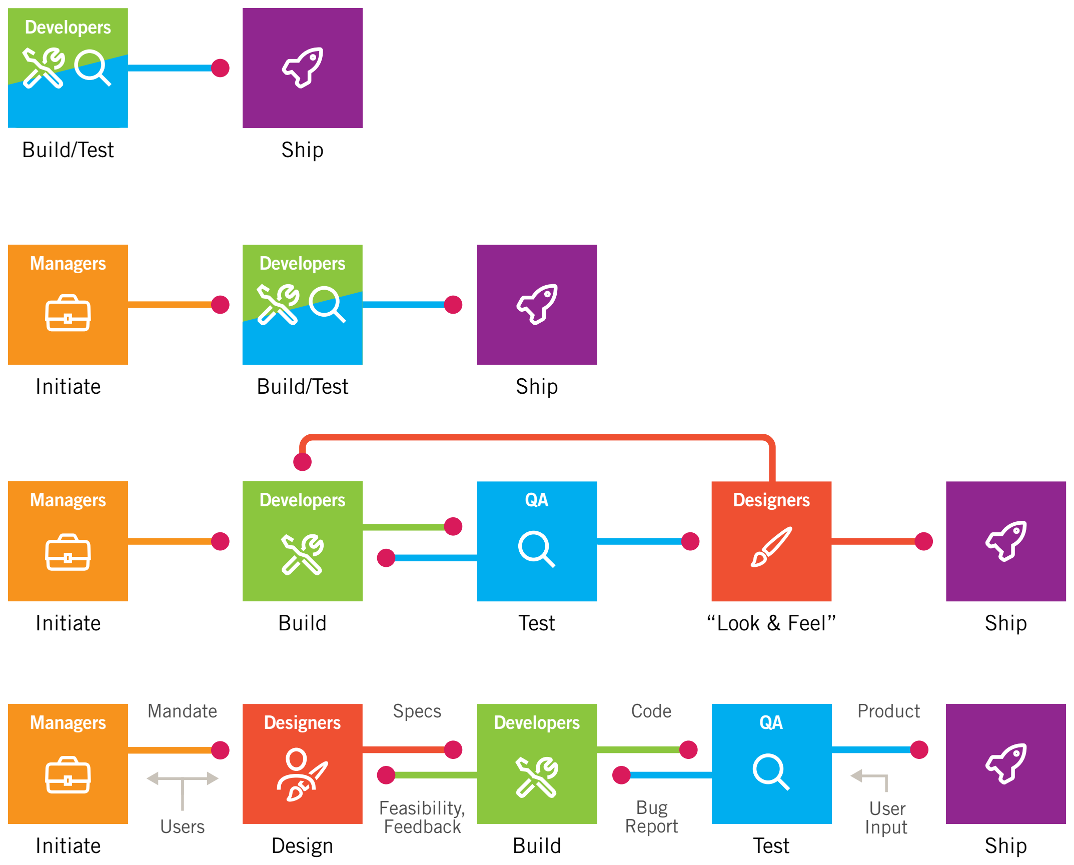


Figure 1-2: The evolution of the software development process. The first diagram depicts the early days of the software industry, when smart developers dreamed up products and then built and tested them. Inevitably, professional managers were brought in to help facilitate the process by translating market opportunities into product requirements. As depicted in the third diagram, the industry matured, and testing became a discipline in its own right. With the popularization of the graphical user interface (GUI), graphic designers were brought in to create icons and other visual elements. The final diagram shows the Goal-Directed approach to software development, where decisions about a product's capabilities, form, and behavior are made before the expensive and challenging construction phase.

Ignorance about real users

It's an unfortunate truth that the digital technology industry doesn't have a good understanding of what it takes to make users happy. In fact, most technology products get built without much understanding of users. We might know what *market segment* our users are in, how much money they make, how they like to spend their weekends, and what sorts of cars they buy. We might even have a vague idea of what kind of jobs they have and some of the major tasks they regularly perform. But does any of this tell us how to make them happy? Does it tell us *how* they will actually use the product we're building? Does it tell us *why* they are doing whatever it is they might need our product for, *why* they might want to choose our product over our competitors, or *how* we can make sure they do? No, it does not.

However, we should not give up hope. It is possible to understand our users well enough to make excellent products they will love. We'll see how to address the issue of understanding users and their behaviors with products in Chapters 2 and 3.

Conflicts of interest

A third problem affects the ability of vendors and manufacturers to make users happy. The world of digital product development has an important conflict of interest: The people who build the products—developers—are often also the people who design them. They are are also, quite understandably, the people who usually have the final say on what does and doesn't get built. Thus, developers often are required to choose between ease of coding and ease of use. Because developers' performance is typically judged by their ability to code efficiently and meet incredibly tight deadlines, it isn't difficult to figure out what direction most software-enabled products take. Just as we would never permit the prosecutor in a legal trial to also adjudicate the case, we should make sure that the people designing a product are not the same people building it. Even with appropriate skills and the best intentions, it simply isn't possible for a developer (or anyone, for that matter) to advocate effectively for the user, the business, and the technology all at the same time.

We'll see how to address the issue of building design teams and fitting them into the planning and development process in Chapter 6.

Lack of a design process

The last reason the digital product industry isn't cranking out successful, well-designed products is that it has no reliable *process* for doing so. Or, to be more accurate, it doesn't have a *complete* process for doing so. Engineering departments follow—or should follow—rigorous engineering methods that ensure the *feasibility* and quality of the technology. Similarly, marketing, sales, and other business units follow their own well-established methods for ensuring the commercial *viability* of new products. What's left out is a repeatable, predictable, and analytical process for ensuring **desirability**: *transforming an understanding of users into products that meet their professional, personal, and emotional needs.*

In the worst case, decisions about what a digital product will do and how it will communicate with users are simply a by-product of its construction. Developers, deep in their thoughts of algorithms and code, end up "designing" product behaviors in the same way that miners end up "designing" a landscape filled with cavernous pits and piles of rubble. In unenlightened development organizations, the digital product interaction design process alternates between the accidental and the nonexistent.

Sometimes organizations do adopt a design process, but it isn't quite up to the task. Many companies embrace the notion that integrating customers (or their theoretical proxies, domain experts) directly into the development process can solve human interface design problems. Although this has the salutary effect of sharing the responsibility for design with the user, it ignores a serious methodological flaw: confusing domain knowledge with design knowledge.

Customers, although they might be able to articulate the problems with an interaction, often cannot visualize the solutions to those problems. Design is a specialized skill, just like software development. Developers would never ask users to help them *code*; design problems should be treated no differently. In addition, customers who *purchase* a product may not be the same people who *use* it from day to day, a subtle but important distinction. Finally, experts in a domain may not be able to easily place themselves in the shoes of less-expert users when defining tasks and flows. Interestingly, the two professions that seem to most frequently confuse domain knowledge with design knowledge when building information systems—law and medicine—have notoriously difficult-to-use products. Coincidence? Probably not.

Of course, designers should indeed get feedback on their proposed solutions, both from users and the product team. But hearing about the problems is much more useful to designers—and better for the product—than taking proposed solutions from users at face value. In interpreting feedback, the following analogy is useful: Imagine a patient who visits his doctor with acute stomach pain. "Doctor," he says, "it *really* hurts. I think it's my appendix. You've got to take it out as soon as possible." A responsible physician wouldn't perform surgery based solely on a patient request, even an earnest one. The patient can describe the symptoms, but it takes the doctor's professional knowledge to make the correct diagnosis and prescribe the treatment.

Planning and Designing Product Behavior

The planning of complex digital products, especially ones that interact directly with humans, requires a significant upfront effort by professional designers, just as the planning of complex physical structures that interact with humans requires a significant upfront effort by professional architects. In the case of architects, that planning involves understanding how the humans occupying the structure live and work, and designing spaces to support and facilitate those behaviors. In the case of digital products, the planning involves understanding how the humans using the product live and work, and designing product behavior and form that support and facilitate the human behaviors. Architecture is an old, well-established field. The design of product and system behavior—**interaction design**—is quite new, and only in recent years has it begun to come of age as a discipline. And this new design has fundamentally changed how products succeed in the marketplace.

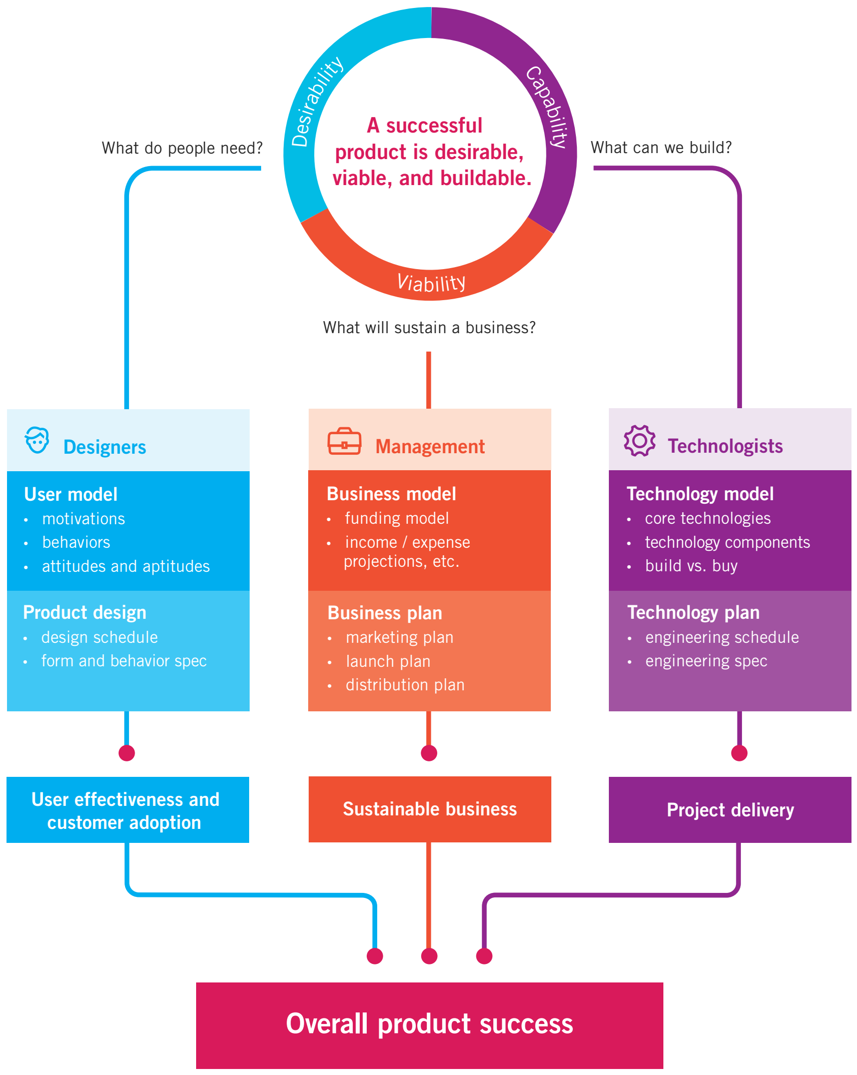
In the early days of industrial manufacturing, engineering and marketing processes alone were sufficient to produce *desirable* products: It didn't take much more than good engineering and reasonable pricing to produce a hammer, diesel engine, or tube of toothpaste that people would readily purchase. As time progressed, manufacturers of consumer products realized that they needed to differentiate their products from functionally identical products made by competitors, so design was introduced as a means to increase user desire for a product. Graphic designers were employed to create more effective packaging and advertising, and industrial designers were engaged to create more comfortable, useful, and exciting forms.

The conscious inclusion of design heralded the ascendance of the modern triad of product development concerns identified by Larry Keeley of the Doblin Group: capability, viability, and desirability (see Figure 1-3). If any of these three foundations is weak, a product is unlikely to stand the test of time.

Now enter the general-purpose computer, the first machine capable of almost limitless *behavior* via software programming. The interesting thing about this complex behavior, or interactivity, is that it completely alters the nature of the products it touches. Interactivity is compelling to humans—so compelling that the other aspects of an interactive product become marginal. Who pays attention to the black PC tower that sits under your desk? It is the screen, keyboard, and mouse to which users pay attention. With touchscreen devices like the iPad and its brethren, the only apparent hardware is the interactive surface. Yet the behaviors of software and other digital products, which should receive the majority of design attention, all too frequently receive no attention.

The traditions of design that corporations have relied on to provide the critical pillar of desirability for products don't provide much guidance in the world of interactivity. *Design of behavior* is a different kind of problem that requires greater knowledge of *context*, not just rules of visual composition and brand. It requires an understanding of the user's relationship with the product from before purchase to end of life. Most important is the understanding of how the user wants to use the product, in what ways, and to what ends.

Interaction design isn't merely a matter of aesthetic choice; rather, it is based on an understanding of users and cognitive principles. This is good news, because it makes the design of behavior quite amenable to a repeatable process of analysis and synthesis. It doesn't mean that the design of behavior can be automated, any more than the design of form or content can be automated, but it *does* mean that a systematic approach is possible. Rules of form and aesthetics mustn't be discarded, of course. They must work in harmony with the larger concern of achieving user goals via appropriately designed behaviors.



You can apply this to companies that have struggled to find the balance:

Apple	Microsoft	Novell
Apple has emphasized desirability but has made many business blunders. Nevertheless, it is sustained by the loyalty created by its attention to user experience.	Microsoft is one of the best run businesses ever, but it has not been able to create highly desirable products. This provides an opening for competition.	Novell emphasized technology and gave little attention to desirability. This made it vulnerable to competition.

Figure 1-3: Building successful digital products. Three major processes need to be followed in tandem to create successful technology products. This book addresses the first and foremost issue: how to create a product people will desire.

This book presents a set of methods to address this new kind of behavior-oriented design, providing a complete process for understanding users' *goals*, needs, and motivations: **Goal-Directed Design**. To understand the process of Goal-Directed Design, we first need to understand the nature of user goals, the *mental models* from which they arise, and how they are the key to designing appropriate interactive behavior.

Recognizing User Goals

So what are user goals? How can we identify them? How do we know that they are real goals, rather than tasks users are forced to perform by poorly designed tools or business processes? Are they the same for all users? Do they change over time? We'll try to answer those questions in the remainder of this chapter.

Users' goals are often quite different from what we might guess them to be. For example, we might think that an accounting clerk's goal is to process invoices efficiently. This is probably not true. Efficient invoice processing is more likely the goal of the clerk's employer. The clerk probably concentrates on goals like appearing competent at his job and keeping himself engaged with his work while performing routine and repetitive tasks—although he may not verbally (or even consciously) acknowledge this.

Regardless of the work we do and the tasks we must accomplish, most of us share these simple, personal goals. Even if we have higher aspirations, they are still more personal than work-related: winning a promotion, learning more about our field, or setting a good example for others, for instance.

Products designed and built to achieve business goals alone will eventually fail; users' personal goals need to be addressed. When the design meets the user's personal goals, business goals are achieved far more effectively, for reasons we'll explore in more detail in later chapters.

If you examine most commercially available software, websites, and digital products, you will find that their user interfaces fail to meet user goals with alarming frequency. They routinely:

- Make users feel stupid.
- Cause users to make big mistakes.
- Require too much effort to operate effectively.
- Don't provide an engaging or enjoyable experience.

Most of the same software is equally poor at achieving its business purpose. Invoices don't get processed all that well. Customers don't get serviced on time. Decisions don't get properly supported. This is no coincidence.

The companies that develop these products have the wrong priorities. Most focus far too narrowly on implementation issues, which distract them from users' needs.

Even when businesses become sensitive to their users, they are often powerless to change their products. The conventional development process assumes that the user interface should be addressed after coding begins—sometimes even after it ends. But just as you cannot effectively design a building after construction begins, you cannot easily make an application serve users' goals as soon as a significant and inflexible code base is in place.

Finally, when companies *do* focus on the users, they tend to pay too much attention to the *tasks* users engage in and not enough attention to their *goals* in performing those tasks. Software can be technologically superb and perform each business task with diligence, yet still be a critical and commercial failure. We can't ignore technology or tasks, but they play only a part in a larger schema that includes designing to meet user goals.

Goals versus tasks and activities

Goals are not the same as tasks or activities. A goal is an expectation of an end condition, whereas both activities and tasks are intermediate steps (at different levels of organization) that help someone to reach a goal or set of goals.

Donald Norman[1] describes a hierarchy in which activities are composed of tasks, which in turn are composed of actions, which are themselves composed of operations. Using this scheme, Norman advocates Activity-Centered Design (ACD), which focuses first and foremost on understanding activities. He claims humans adapt to the tools at hand and that understanding the activities people perform with a set of tools can more favorably influence the design of those tools. The foundation of Norman's thinking comes from Activity Theory, a Soviet-era Russian theory of psychology that emphasizes understanding who people are by understanding how they interact with the world. In recent years this theory has been adapted to the study of human-computer interaction, most notably by Bonnie Nardi.[2]

Norman concludes, correctly, that the traditional task-based focus of digital product design has yielded inadequate results. Many developers and usability professionals still approach interface design by asking what the tasks are. Although this may get the job done, it won't produce much more than an incremental improvement: It won't provide a solution that differentiates your product in the market, and very often it won't really satisfy the user.

While Norman's ACD takes some important steps in the right direction by highlighting the importance of the user's context, we do not believe it goes quite far enough. A method like ACD can be very useful in properly breaking down the "what" of user behaviors, but it really doesn't address the first question any designer should ask: *Why* is a user performing an activity, task, action, or operation in the first place? Goals motivate people to perform activities; understanding goals allows you to understand your users' expectations and aspirations, which in turn can help you decide which activities are truly relevant to your design. Task and activity analysis is useful at the detail level, but only after user goals have been analyzed. Asking, "What are the user's goals?" lets you understand the *meaning* of activities to your users and thus create more appropriate and satisfactory designs.

If you're still unsure about the difference between goals and activities or tasks, there is an easy way to tell the difference between them. Since goals are driven by human motivations, they change very slowly—if at all—over time. Activities and tasks are much more transient, because they are based almost entirely on whatever technology is at hand. For example, when someone travels from St. Louis to San Francisco, his *goals* are likely to include traveling quickly, comfortably, and safely. In 1850, a settler wishing to travel quickly and comfortably would have made the journey in a covered wagon; in the interest of safety, he would have brought along his trusty rifle. Today, a businessman traveling from St. Louis to San Francisco makes the journey in a jet and, in the interest of safety, he is required to leave his firearms at home. The *goals* of the settler and businessman remain unchanged, but their activities and tasks have changed so completely with the changes in technology that they are, in some respects, in direct opposition.

Design based solely on understanding activities or tasks runs the risk of trapping the design in a model imposed by an outmoded technology, or using a model that meets a corporation's goals without meeting the users' goals. Looking through the lens of goals allows you to leverage available technology to eliminate irrelevant tasks and to dramatically streamline activities. Understanding users' goals can help designers eliminate the tasks and activities that better technology renders unnecessary for humans to perform.

Designing to meet goals in context

Many designers assume that making user interfaces and product interactions easier to learn should always be a design target. Ease of learning is an important guideline, but in reality, the design target really depends on the context—who the users are, what they are doing, and their goals. You simply can't create good design by following rules disconnected from the goals and needs of the users of your product.

Consider an automated call-distribution system. The people who use this product are paid based on how many calls they handle. Their most important concern is not ease of learning, but the efficiency with which they can route calls, and the rapidity with which those calls can be completed. Ease of learning is also important, however, because it affects employees' happiness and, ultimately, turnover rate, so both ease and throughput should be considered in the design. But there is no doubt that throughput is the dominant demand placed on the system by the users, so, if necessary, ease of learning should take a backseat. An application that walks the user through the call-routing process step by step each time merely frustrates him after he's learned the ropes.

On the other hand, if the product in question is a kiosk in a corporate lobby helping visitors find their way around, ease of use for first-time users is clearly a major goal.

A general guideline of interaction design that seems to apply particularly well to productivity tools is that *good design makes users more effective*. This guideline takes into account the universal human goal of not looking stupid, along with more particular goals of business throughput and ease of use that are relevant in most business situations.

It is up to you as a designer to determine how you can make the users of your product more effective. Software that enables users to perform their tasks *without addressing their goals* rarely helps them be truly effective. If the task is to enter 5,000 names and addresses into a database, a smoothly functioning data-entry application won't satisfy the user nearly as much as an automated system that extracts the names from the invoicing system.

Although it is the user's job to focus on her tasks, the designer's job is to look beyond the task to identify *who* the most important users are, and then to determine *what* their goals might be and *why*.

Implementation Models and Mental Models

The computer industry still makes use of the term *computer literacy*. Pundits talk about how some people have it and some don't, how those who have it will succeed in the information economy, and how those who lack it will inevitably fall between the socioeconomic cracks. Computer literacy, however, is really a euphemism for forcing human beings to stretch their thinking to understand the inner workings of application logic, rather than having software-enabled products stretch to meet people's usual ways of thinking.

Let's explore what's really going on when people try to use digital products, and what the role of design is in translating coded functions into an understandable and pleasurable experience for users.

Implementation models

Any machine has a mechanism for accomplishing its purpose. A motion picture projector, for example, uses a complicated sequence of intricately moving parts to create its illusion. It shines a very bright light through a translucent, miniature image for a fraction of a second. It then blocks the light for a split second while it moves another miniature image into place. Then it unblocks the light again for another moment. It repeats this process with a new image 24 times per second. Software-enabled products don't have mechanisms in the sense of moving parts; these are replaced with algorithms and modules of code that communicate with each other. The representation of how a machine or application actually works has been called the *system model* by Donald Norman and others; we prefer the term *implementation model* because it describes the details of how an application is implemented in code.

It is much easier to design software that reflects its implementation model. From the developer's perspective, it's perfectly logical to provide a button for every function, a field for every data input, a page for every transaction step, and a dialog box for every code module. But while this adequately reflects the infrastructure of engineering efforts, it does little to provide coherent mechanisms for a user to achieve his goals. In the end, what is produced alienates and confuses the user, rather like the ubiquitous external ductwork in the dystopian setting of Terry Gilliam's movie *Brazil* (which is full of wonderful tongue-in-cheek examples of miserable interfaces).

Mental models

From the moviegoer's point of view, it is easy to forget the nuance of sprocket holes and light interrupters while watching an absorbing drama. Many moviegoers, in fact, have little idea how the projector works, or how this differs from the way a television works. The viewer imagines that the projector merely throws a picture that moves onto the big screen. This is his *mental model*, or *conceptual model*.

People don't need to know all the details of how a complex mechanism actually works in order to use it, so they create a cognitive shorthand for explaining it. This explanation is powerful enough to cover their interactions with it but doesn't necessarily reflect its actual inner mechanics. For example, many people imagine that, when they plug in their vacuum cleaner and blender, the electricity flows like water from the wall into the appliances through the little black tube of the electrical cord. This mental model is perfectly adequate for using household appliances. The fact that the implementation model of household electricity involves nothing resembling a fluid traveling through a tube and that there is a reversal of electrical potential 120 times per second is irrelevant to the user, although the power company needs to know the details.

In the digital world, however, the differences between a user's mental model and the implementation model are often quite distinct. We tend to ignore the fact that our cell phone doesn't work like a landline phone; instead, it is actually a radio transceiver that might swap connections between a half-dozen different cellular base antennas in the course of a two-minute call. Knowing this doesn't help us understand how to *use* the phone.

The discrepancy between implementation and mental models is particularly stark in the case of software applications, where the complexity of implementation can make it nearly impossible for the user to see the mechanistic connections between his actions and the application's reactions. When we use a computer to digitally edit sound or create video special effects like morphing, we are bereft of analogy to the mechanical world, so our mental models are necessarily different from the implementation model. Even if the connections were visible, they would remain inscrutable to most people.

Striving toward perfection: represented models

Software (and any digital product that relies on software) has a behavioral face it shows to the world that is created by the developer or designer. This representation is not necessarily an accurate description of what is really going on inside the computer, although unfortunately, it frequently is. This ability to *represent* the computer's functioning independent of its true actions is far more pronounced in software than in any other medium. It allows a clever designer to hide some of the more unsavory facts of how the software really gets the job done. This disconnection between what is implemented and what is offered as explanation gives rise to a *third* model in the digital world, the designer's *represented model*—how the designer chooses to represent an application's functioning to the user. Donald Norman calls this the *designer's model*.

In the world of software, an application's represented model can (and often should) be quite different from an application's actual processing structure. For example, an operating system can make a network file server look as though it were a local disk. The model does not represent the fact that the physical disk drive may be miles away. This concept of the represented model has no widespread counterpart in the mechanical world. Figure 1-4 shows the relationship between the three models.

The closer the represented model comes to the user's mental model, the easier he will find the application to use and understand. Generally, offering a represented model that follows the implementation model too closely significantly reduces the user's ability to learn and use the application. This occurs because the user's mental model of his tasks usually differs from the software's implementation model.

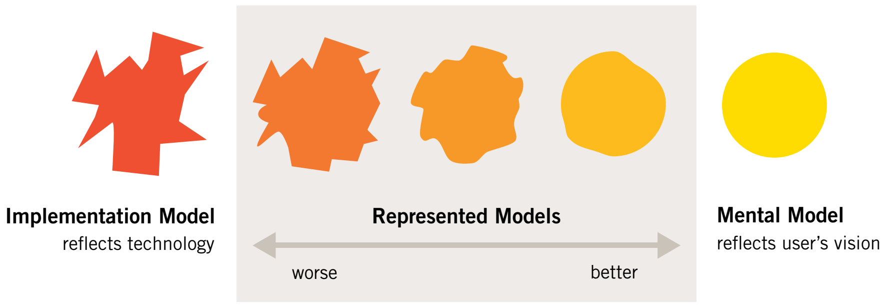


Figure 1-4: A comparison of the implementation model, mental model, and represented model. The way engineers must build software is often a given, dictated by various technical and business constraints. The model for how the software actually works is called the *implementation model*. The way users perceive the jobs they need to do and how the application helps them do so is their *mental model* of interaction with the software. It is based on their own ideas of how they do their jobs and how computers might work. The way designers choose to represent the working of the application to the user is called the *represented model*. Unlike the other two models, it is an aspect of software over which designers have great control. One of the designer's most important goals should be to make the represented model match a user's mental model as closely as possible. Therefore, it is critical that designers understand in detail how their target users think about the work they do with the software.

We tend to form mental models that are simpler than reality. So, if we create represented models that are simpler than the implementation model, we help the user achieve better understanding. In software, we imagine that a spreadsheet scrolls new cells into view when we click the scrollbar. Nothing of the sort actually happens. There is no sheet of cells out there, but a tightly packed data structure of values, with various pointers between them, from which the application synthesizes a new image to display in real time.

One of the most significant ways in which computers can assist human beings is presenting complex data and operations in a simple, easily understandable form. As a result, user interfaces that are consistent with users' mental models are vastly superior to those that are merely reflections of the implementation model.

DESIGN PRINCIPLE

User interfaces should be based on user mental models rather than implementation models.

In Adobe Photoshop Express on the iPad, users can adjust a set of ten different visual filters, including noise, contrast, exposure, and tint. Instead of offering numeric fields or many banks of controls for entering filter values—the implementation model—the interface instead shows a set of thumbnail images of the edited photo, each with a different filter applied (see Figure 1-5). A user can tap the image that best represents the desired result, and can tweak it with a single large slider. The interface more closely follows his mental model, because the user—likely an amateur photographer—is thinking in terms of how his photo looks, not in terms of abstract numbers.

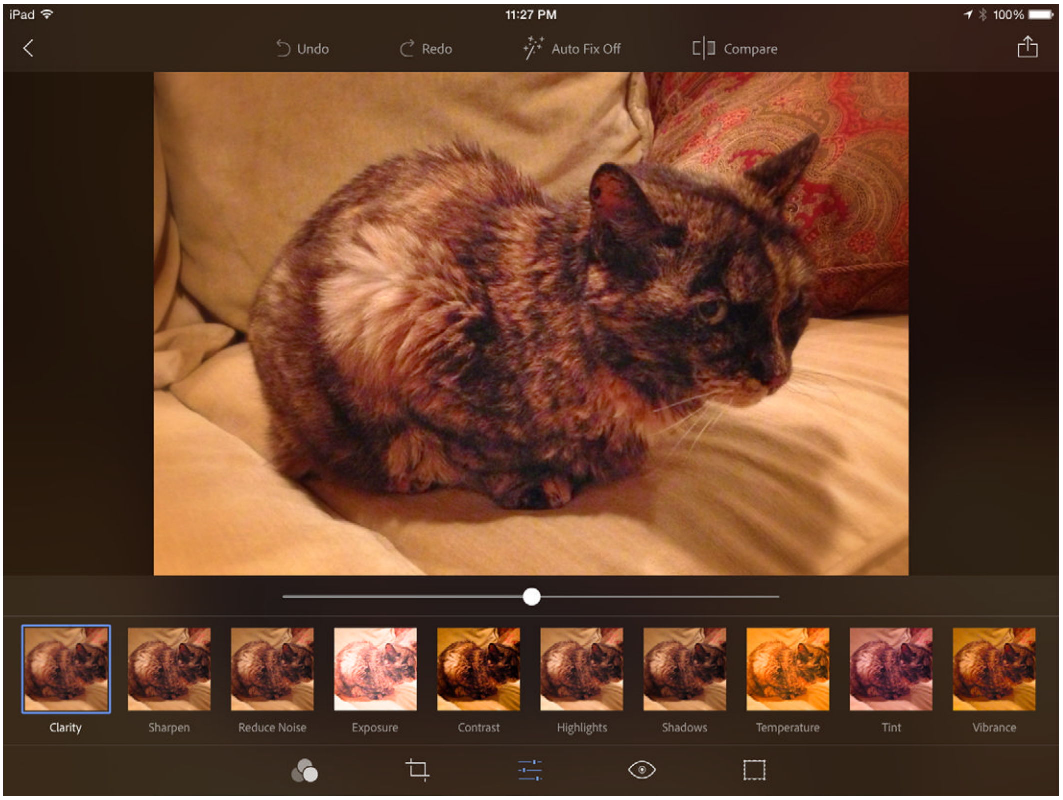


Figure 1-5: Adobe Photoshop Express for iPad has a great example of software design to match user mental models. The interface shows a set of thumbnail images of the photo being edited. A user can tap the thumbnail that best represents the desired setting, which can then be tweaked using the single large slider below the photo. The interface follows mental models of photographers who are after a particular look, not a set of abstract numeric values.

If the represented model for software closely follows users' mental models, it eliminates needless complexity from the user interface by providing a cognitive framework that makes it evident to the user how his goals and needs can be met.

DESIGN PRINCIPLE

Goal-directed interactions reflect user mental models.

So, now we know that a missing link prevents the majority of digital products from being truly successful. A design process translates the implementation of features into intuitive and desirable product behaviors that match how people think about performing tasks toward achieving their goals. But how do we actually do it? How do we know what our users' goals are and what mental models they have of their activities and tasks?

The Goal-Directed Design process, which we describe in the remainder of this chapter and in the remainder of Part I, provides a structure for determining the answers to these questions—a structure by which solutions based on this information can be systematically achieved.

An Overview of Goal-Directed Design

Most technology-focused companies don't have an adequate process for product design, if they have a process at all. But even the more enlightened organizations—those that can boast of an established process—come up against some critical issues that result from traditional ways of approaching the problems of research and design.

In recent years, the business community has come to recognize that user research is necessary to create good products, but the proper nature of that research is still in question in many organizations. Quantitative market research and market segmentation are quite useful for *selling* products but fall short of providing critical information about *how people actually use products*—especially products with complex behaviors. (See Chapter 2 for a more in-depth discussion of this topic.) A second problem occurs after the results have been analyzed: Most traditional methods don't provide a means of *translating research results into design solutions*. A hundred pages of user survey data don't easily translate into a set of product requirements. They say even less about how those requirements should be expressed in terms of a meaningful and appropriate interface structure. Design remains a black box: "A miracle happens here…" This gap between research results and the ultimate design solution is the result of a process that doesn't connect the dots from user to final product. We'll soon see how to address this problem with Goal-Directed methods.

Bridging the gap

As we have briefly discussed, the role of design in the development process needs to change. We need to start thinking about design in new ways and start thinking differently about how product decisions are made.

Design as product definition

Design has, unfortunately, become a limiting term in the technology industry. For many developers and managers, the word stands for what happens in the third process diagram shown in Figure 1-2: a visual facelift of the *implementation model*. But design, when properly deployed (as in the fourth process diagram shown in Figure 1-2), both identifies user requirements and defines a detailed plan for the behavior and appearance of products. In other words, design provides true *product definition*, based on user goals, business needs, and technology constraints.

Designers as researchers

If design is to become product definition, designers need to take on a broader role than that assumed in traditional practice, particularly when the object in question is complex, interactive systems.

One of the problems with the current development process is that roles in the process are overspecialized: Researchers perform research, and designers perform design (see Figure 1-6). The results of user and market research are analyzed by the usability and market researchers and then thrown over the transom to designers or developers. What is missing in this model is a systematic means of translating and synthesizing the research into design solutions. One of the ways to address this problem is for designers to learn to be researchers.

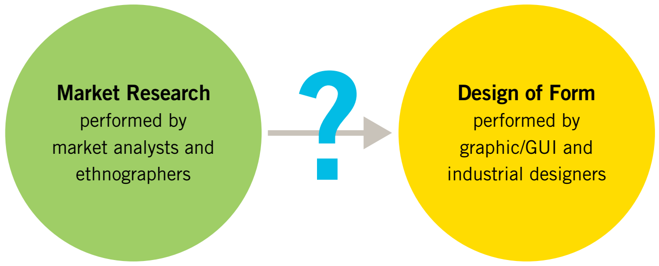


Figure 1-6: A problematic design process. Traditionally, research and design have been separated, with each activity handled by specialists. *Research* has, until recently, referred primarily to market research, and *design* is too often limited to visual design or skin-deep industrial design. More recently, user research has expanded to include qualitative, ethnographic data. Yet, without including designers in the research process, the connection between research data and design solutions remains tenuous at best.

There is a compelling reason to involve designers in the research process. One of the most powerful tools designers offer is empathy: the ability to feel what others are feeling. The direct and extensive exposure to users that proper user research entails immerses designers in the users' world and gets them thinking about users long before they propose solutions. One of the most dangerous practices in product development is isolating designers from the users, because doing so eliminates empathic knowledge.

Additionally, it is often difficult for pure researchers to know what user information is really important from a design perspective. Involving designers directly in research addresses both issues.

In the authors' practice, designers are trained in the research techniques described in Chapter 2 and perform their research without further support or collaboration. This is a satisfactory solution, provided that your team has the time and resources to train your designers fully in these techniques. If not, a cross-disciplinary team of designers and dedicated user researchers is appropriate.

Although research practiced by designers takes us part of the way to Goal-Directed Design solutions, a translation gap still exists between research results and design details. The puzzle is missing several pieces, as we will discuss next.

Between research and blueprint: Models, requirements, and frameworks

Few design methods in common use today incorporate a means of effectively and systematically translating the knowledge gathered during research into a detailed design specification. Part of the reason for this has already been identified: Designers have historically been out of the research loop and have had to rely on third-person accounts of user behaviors and desires.

The other reason, however, is that few methods capture user behaviors in a manner that appropriately directs the definition of a product. Rather than providing information about user goals, most methods provide information at the task level. This type of information is useful for defining layout, work flow, and translation of functions into interface controls. But it's less useful for defining the basic framework of what a product *is*, what it *does*, and how it should meet the user's broad needs.

Instead, we need an explicit, systematic process to bridge the gap between research and design for defining user models, establishing design requirements, and translating those into a high-level interaction framework (see Figure 1-7). Goal-Directed Design seeks to bridge the gap that currently exists in the digital product development process—the gap between user research and design—through a combination of new techniques and known methods brought together in more effective ways.

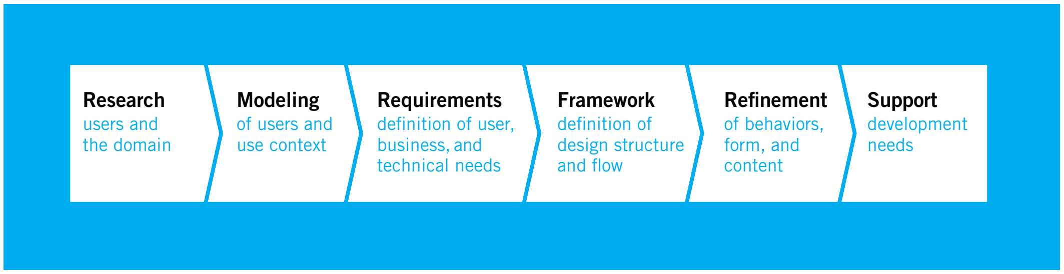

Research
users and the domain
Modeling
of users and use context
Requirements
definition of user, business, and technical needs
Framework
definition of design structure and flow
Refinement
of behaviors, form, and content
Support
development needs

Figure 1-7: The Goal-Directed Design process

A process overview

Goal-Directed Design combines techniques of ethnography, stakeholder interviews, market research, detailed user models, scenario-based design, and a core set of interaction principles and patterns. It provides solutions that meet users' needs and goals while also addressing business/organizational and technical imperatives. This process can be roughly divided into six phases: Research, Modeling, Requirements Definition, Framework Definition, Refinement, and Support (see Figure 1-7). These phases follow the five component activities of interaction design identified by Gillian Crampton Smith and Philip Tabor—understanding, abstracting, structuring, representing, and detailing—with a greater emphasis on modeling user behaviors and defining system behaviors.

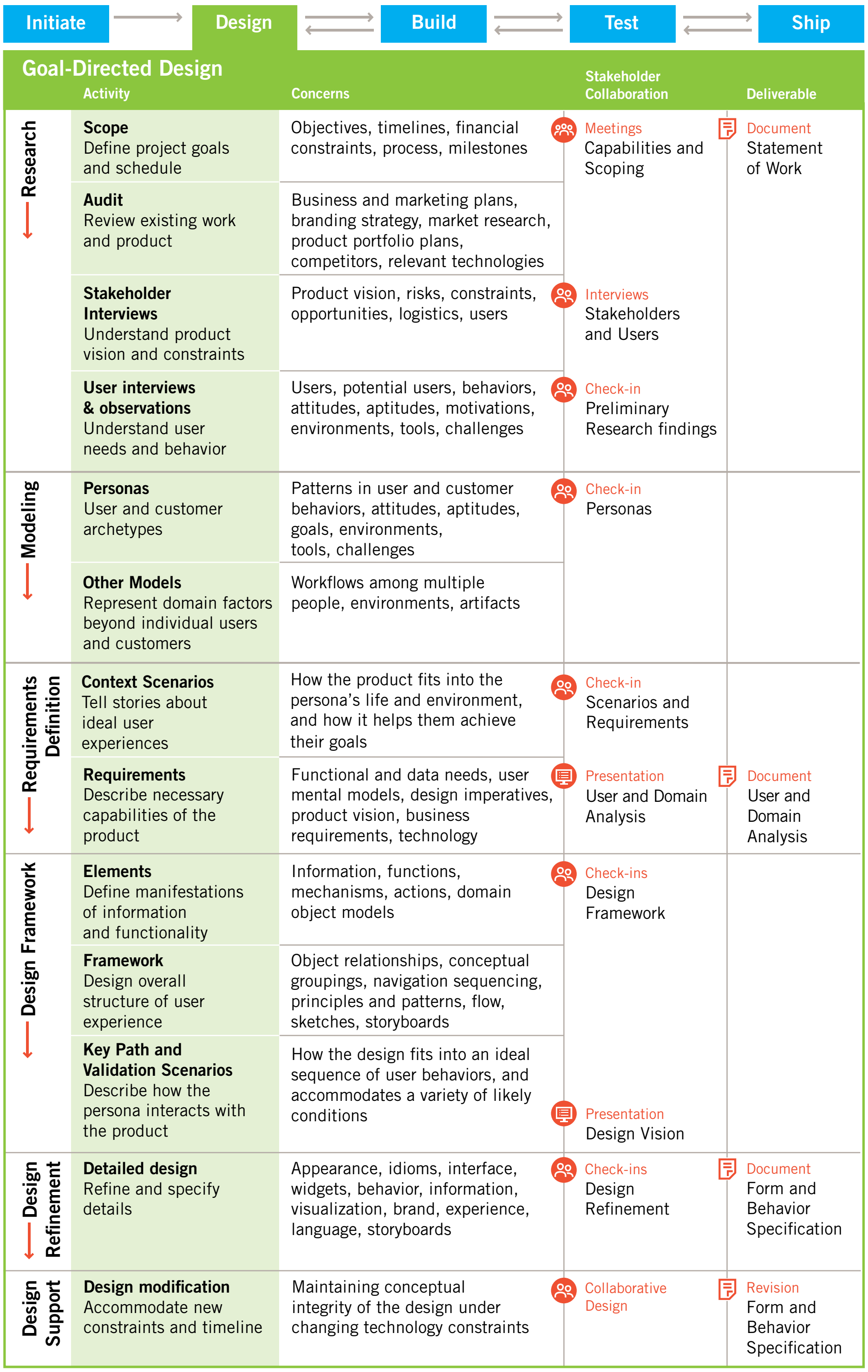
The remainder of this chapter provides a high-level view of the six phases of Goal-Directed Design, and Chapters 2 through 6 provide a more detailed discussion of the methods involved in each of these phases. Figure 1-8 shows a more detailed diagram of the process, including key collaboration points and design concerns.

Research

The Research phase employs ethnographic field study techniques (observation and contextual interviews) to provide qualitative data about potential and/or actual users of the product. It also includes competitive product audits as well as reviews of market research, technology white papers, and brand strategy. It also includes one-on-one interviews with stakeholders, developers, subject matter experts (SMEs), and technology experts as suits the particular domain.

One of the principal outcomes of field observation and user interviews is an emergent set of *behavior patterns*—identifiable behaviors that help categorize modes of use of a potential or existing product. These patterns suggest goals and motivations (specific and general desired outcomes of using the product). In business and technical domains, these behavior patterns tend to map into professional roles; for consumer products, they tend to correspond to lifestyle choices. Behavior patterns and the goals associated with them drive the creation of *personas* in the Modeling phase. Market research helps select and filter valid personas that fit business models. Stakeholder interviews, literature reviews, and product audits deepen the designers' understanding of the domain and elucidate business goals, brand attributes, and technical constraints that the design must support.

Chapter 2 provides a more detailed discussion of Goal-Directed research techniques.


Initiate
Design
Build
Test
Ship

Goal-Directed Design

	Activity	Concerns	Stakeholder Collaboration	Deliverable
Research	**Scope** Define project goals and schedule	Objectives, timelines, financial constraints, process, milestones	Meetings Capabilities and Scoping	Document Statement of Work
	Audit Review existing work and product	Business and marketing plans, branding strategy, market research, product portfolio plans, competitors, relevant technologies		
	Stakeholder Interviews Understand product vision and constraints	Product vision, risks, constraints, opportunities, logistics, users	Interviews Stakeholders and Users	
	User interviews & observations Understand user needs and behavior	Users, potential users, behaviors, attitudes, aptitudes, motivations, environments, tools, challenges	Check-in Preliminary Research findings	
Modeling	**Personas** User and customer archetypes	Patterns in user and customer behaviors, attitudes, aptitudes, goals, environments, tools, challenges	Check-in Personas	
	Other Models Represent domain factors beyond individual users and customers	Workflows among multiple people, environments, artifacts		
Requirements Definition	**Context Scenarios** Tell stories about ideal user experiences	How the product fits into the persona's life and environment, and how it helps them achieve their goals	Check-in Scenarios and Requirements	
	Requirements Describe necessary capabilities of the product	Functional and data needs, user mental models, design imperatives, product vision, business requirements, technology	Presentation User and Domain Analysis	Document User and Domain Analysis
Design Framework	**Elements** Define manifestations of information and functionality	Information, functions, mechanisms, actions, domain object models	Check-ins Design Framework	
	Framework Design overall structure of user experience	Object relationships, conceptual groupings, navigation sequencing, principles and patterns, flow, sketches, storyboards		
	Key Path and Validation Scenarios Describe how the persona interacts with the product	How the design fits into an ideal sequence of user behaviors, and accommodates a variety of likely conditions	Presentation Design Vision	
Design Refinement	**Detailed design** Refine and specify details	Appearance, idioms, interface, widgets, behavior, information, visualization, brand, experience, language, storyboards	Check-ins Design Refinement	Document Form and Behavior Specification
Design Support	**Design modification** Accommodate new constraints and timeline	Maintaining conceptual integrity of the design under changing technology constraints	Collaborative Design	Revision Form and Behavior Specification

Figure 1-8: A more detailed look at the Goal-Directed Design process

Modeling

During the Modeling phase, behavior and work flow patterns discovered by analyzing the field research and interviews are synthesized into domain and user models. Domain models can include information flow and work flow diagrams. User models, or personas, are detailed, composite *user archetypes* that represent distinct groupings of behaviors, attitudes, aptitudes, goals, and motivations observed and identified during the Research phase.

Personas are the main characters in a narrative, scenario-based approach to design. This approach iteratively generates design concepts in the Framework Definition phase. It provides feedback that enforces design coherence and appropriateness in the Refinement phase. It also is a powerful communication tool that helps developers and managers understand design rationale and prioritize features based on user needs. In the Modeling phase, designers employ a variety of methodological tools to synthesize, differentiate, and prioritize personas, exploring different *types* of goals and mapping personas across ranges of behavior to ensure that no gaps or duplications exist.

Specific design targets are chosen from the cast of personas through a process of comparing goals and assigning priorities based on how broadly each persona's goals encompass the goals of other personas. A process of designating persona types determines how much influence each persona has on the design's eventual form and behavior.

A detailed discussion of persona and goal development can be found in Chapter 3.

Requirements Definition

Design methods employed by teams during the Requirements Definition phase provide the much-needed connection between user and other models and design's framework. This phase employs scenario-based design methods with the important innovation of focusing the scenarios not on user tasks in the abstract, but first and foremost on meeting the goals and needs of specific user personas. Personas help us understand which tasks are truly important and why, leading to an interface that minimizes necessary tasks (effort) while maximizing return. Personas become the main characters of these scenarios, and the designers explore the design space via a form of role playing.

For each interface/primary persona, the process of design in the Requirements Definition phase involves analyzing persona data and functional needs (expressed in terms of objects, actions, and contexts), prioritized and informed by persona goals, behaviors, and interactions with other personas in various contexts.

This analysis is accomplished through an iteratively refined *context scenario*. It starts with a "day in the life" of the persona using the product, describing high-level product touch points, and thereafter successively defining detail at ever-deepening levels. In addition to these scenario-driven requirements, designers consider the personas' skills and physical capabilities as well as issues related to the usage environment. Business goals, desired brand attributes, and technical constraints are also considered and balanced with persona goals and needs. The output of this process is a *requirements definition* that balances user, business, and technical requirements of the design to follow.

Chapter 4 covers the process of establishing requirements through the use of *scenarios*.

Framework Definition

In the Framework Definition phase, designers create the overall product concept, defining the basic frameworks for the product's behavior, visual design, and, if applicable, physical form. Interaction design teams synthesize an *interaction framework* by employing two other critical methodological tools in conjunction with context scenarios. The first is a set of general *interaction design principles* that provide guidance in determining appropriate system behavior in a variety of contexts. Part II of this book is devoted to high-level interaction design principles appropriate to the Framework Definition phase.

The second critical methodological tool is a set of *interaction design patterns* that encode general solutions (with variations dependent on context) to classes of previously analyzed problems. These patterns bear close resemblance to the concept of architectural design patterns first developed by Christopher Alexander[3] and more recently brought to the programming field by Erich Gamma and others.[4] Interaction design patterns are hierarchically organized and continuously evolve as new contexts arise. Rather than stifling designer creativity, they often provide needed leverage to approach difficult problems with proven design knowledge.

After data and functional needs are described at this high level, they are translated into design elements according to interaction principles and then organized, using patterns and principles, into design sketches and behavior descriptions. The output of this process is an *interaction framework definition*, a stable design concept that provides the logical and hi-level formal structure for the detail to come. Successive iterations of more narrowly focused scenarios provide this detail in the Refinement phase. The approach is often a balance of top-down (pattern-oriented) design and bottom-up (principle-oriented) design.

When the product takes physical form, interaction designers and industrial designers begin by collaborating closely on various *input methods* and approximate *form factors* the product might take, using scenarios to consider the pros and cons of each. As this is narrowed to a couple of options that seem promising, industrial designers begin producing early physical prototypes to ensure that the overall interaction concept will work. It's critical at this early stage that industrial designers not create concepts independent of the product's behavior.

When working to design a service, we will collaborate with service designers to draft a *service map* and a *blueprint* that coordinates touchpoints and experiences across channels, both "backstage" with the service providers and "frontstage" experiences from the users' point of view.

As soon as an interaction framework begins to emerge, visual interface designers produce several options for a *visual framework*, which is sometimes also called a *visual language strategy*. They use brand attributes as well as an understanding of the overall interface structure to develop options for typography, color palettes, and visual style.

Refinement

The Refinement phase proceeds similarly to the Framework Definition phase, but with increasing focus on detail and implementation. Interaction designers focus on task coherence, using *key path scenarios* (walkthroughs) and *validation scenarios* focused on storyboarding paths through the interface in great detail. Visual designers define a system of type styles and sizes, icons, and other visual elements that provide a compelling experience with clear affordances and visual hierarchy. Industrial designers, when appropriate, finalize materials and work closely with engineers on assembly schemes and other technical issues. The culmination of the Refinement phase is the detailed documentation of the design—a *form and behavior specification or blueprint*, delivered in either paper or interactive media form as the context dictates.

Chapter 5 discusses in more detail the use of personas, scenarios, principles, and patterns in the Framework Definition and Refinement phases.

Development support

Even a very well-conceived and validated design solution can't possibly anticipate every development challenge and technical question. In our practice, we've learned that it's important to be available to answer developers' questions as they arise during the construction process. It is often the case that as the development team prioritizes their work and makes trade-offs to meet deadlines, the design must be adjusted, requiring scaled-down design solutions. If the interaction design team is not available to create these solutions, developers are forced to do this under time pressure, which has the potential to gravely compromise the integrity of the product's design.

Chapter 6 discusses how interaction design activities and processes can be integrated with the larger product team.

Goals, not features, are the key to product success

Developers and marketers often use the language of features and functions to discuss products. But reducing a product's definition to a list of features and functions ignores the real opportunity—orchestrating technological capability to serve human needs and goals. Too often the features of our products are a patchwork of nifty technological innovations structured around a marketing requirements document or organization of the development team, with too little attention paid to the overall user experience.

The successful interaction designer must maintain her focus on users' goals amid the pressures and chaos of the product-development cycle. Although we discuss many other techniques and tools of interaction in this book, we always return to users' goals. They are the bedrock upon which interaction design should be practiced.

The Goal-Directed process, with its clear rationale for design decisions, makes collaboration with developers and businesspeople easier. It also ensures that the design in question isn't guesswork, the whim of a creative mind, or just a reflection of the team members' personal preferences.

DESIGN PRINCIPLE

Interaction design is not guesswork.

Goal-Directed Design is a powerful tool for answering the most important questions that crop up during the definition and design of a digital product:

- Who are my users?
- What are my users trying to accomplish?
- How do my users think about what they're trying to accomplish?
- What kind of experiences do my users find appealing and rewarding?
- How should my product behave?
- What form should my product take?
- How will users interact with my product?
- How can my product's functions be most effectively organized?
- How will my product introduce itself to first-time users?
- How can my product put an understandable, appealing, and controllable face on technology?
- How can my product deal with problems that users encounter?

- How will my product help infrequent and inexperienced users understand how to accomplish their goals?
- How can my product provide sufficient depth and power for expert users?

The remainder of this book is dedicated to answering these questions. We share tools tested by years of experience with hundreds of products that can help you identify key users of your products, understand them and their goals, and translate this understanding into effective and appealing design solutions.

Notes

1. Norman, 2005
2. Nardi, 1996
3. Alexander, 1979
4. Gamma, et al, 1994