About Face 3

The Essentials of Interaction Design

Alan Cooper, Robert Reimann, and Dave Cronin

Wiley Publishing, Inc.
About Face 3
The Essentials of Interaction Design

Alan Cooper, Robert Reimann, and Dave Cronin

Wiley Publishing, Inc.
For Sue, my best friend through all the adventures of life.

For Maxwell Aaron Reimann.

For Gretchen.

And for Cooperistas past, present, and future; and for those visionary IxD practitioners who have helped create a new design profession.
About the Authors

Alan Cooper is a pioneering software inventor, programmer, designer, and theorist. He is credited with having produced “probably the first serious business software for microcomputers” and is well known as the “Father of Visual Basic.” For the last 15 years his software design consulting company, Cooper, has helped many companies invent new products and improve the behavior of their technology. At Cooper, Alan led the development of a new methodology for creating successful software that he calls the Goal-Directed process. Part of that effort was the invention of personas, a practice that has been widely adopted since he first published the technique in his second book, *The Inmates are Running the Asylum*, in 1998. Cooper is also a well known writer, speaker, and enthusiast for humanizing technology.

Robert Reimann has spent the past 15 years pushing the boundaries of digital products as a designer, writer, lecturer, and consultant. He has led dozens of interaction design projects in domains including e-commerce, portals, desktop productivity, authoring environments, medical and scientific instrumentation, wireless, and handheld devices for startups and Fortune 500 clients alike. As director of design R&D at Cooper, Reimann led the development and refinement of many of the Goal-Directed Design methods described in *About Face*. In 2005, Reimann became the first President of IxDA, the Interaction Design Association (www.ixda.org), a global nonprofit professional organization for Interaction Designers. He is currently manager of user experience at Bose Corporation.

Dave Cronin is the director of interaction design at Cooper, where he’s helped design products to serve the needs of people such as surgeons, museum visitors, marketers, investment portfolio managers, online shoppers, hospital staff, car drivers, dentists, financial analysts, manufacturing planners, the elderly, and the infirm. At Cooper, he has also contributed substantially to the ongoing process of developing and refining the Goal-Directed Design methods described in this book.
Credits

Executive Editor
Chris Webb

Development Editors
Sara Shlaer
Sydney Jones

Production Editor
Eric Charbonneau

Copy Editor
Foxxe Editorial Services

Editorial Manager
Mary Beth Wakefield

Production Manager
Tim Tate

Vice President and Executive Group Publisher
Richard Swadley

Project Coordinator
Erin Smith

Graphics and Production Specialists
Sean Decker, Brooke Graczyk,
Stephanie D. Jumper,
Jennifer Mayberry, Barbara Moore,
Ronald Terry

Quality Control Technician
Christy Pingleton

Book Designers
Rebecca Bortman and Nick Myers

Illustrators
Rebecca Bortman and Nick Myers

Proofreading and Indexing
Aptara

Anniversary Logo Design
Richard Pacifico

Cover Design
Rebecca Bortman and Nick Myers
## Contents

About the Authors vi  
Foreword: The Postindustrial World xxi  
Acknowledgments xxv  
Introduction to the Third Edition xxvii  

### Part I: Understanding Goal-Directed Design 1  

#### Chapter 1: Goal-Directed Design 3  
- Digital Products Need Better Design Methods 3  
- The creation of digital products today 4  
- Why are these products so bad? 8  
- The Evolution of Design in Manufacturing 11  
- Planning and Designing Behavior 13  
- Recognizing User Goals 13  
- Goals versus tasks and activities 15  
- Designing to meet goals in context 16  
- The Goal-Directed Design Process 17  
- Bridging the gap 18  
- A process overview 20  
- Goals, not features, are the key to product success 25  

#### Chapter 2: Implementation Models and Mental Models 27  
- Implementation Models 27  
- User Mental Models 28  
- Represented Models 29  
- Most Software Conforms to Implementation Models 32  
- User interfaces designed by engineers follow the implementation model 32  
- Mathematical thinking leads to implementation model interfaces 34  
- Mechanical-Age versus Information-Age Represented Models 35  
- Mechanical-Age representations 35  
- New technology demands new representations 36  
- Mechanical-Age representations degrade user interaction 36  
- Improving on Mechanical-Age representations: An example 37
## Chapter 3  Beginners, Experts, and Intermediates

- Perpetual Intermediates 42
- Designing for Different Experience Levels 44
  - What beginners need 45
  - Getting beginners on board 46
  - What experts need 47
  - What perpetual intermediates need 47

## Chapter 4  Understanding Users: Qualitative Research

- Qualitative versus Quantitative Research 50
- The value of qualitative research 50
- Types of qualitative research 52
- Ethnographic Interviews: Interviewing and Observing Users 58
  - Contextual inquiry 58
  - Improving on contextual inquiry 59
  - Preparing for ethnographic interviews 59
  - Conducting ethnographic interviews 63
- Other Types of Research 68
  - Focus groups 69
  - Market demographics and market segments 69
  - Usability and user testing 70
  - Card sorting 72
  - Task analysis 72

## Chapter 5  Modeling Users: Personas and Goals

- Why Model? 76
- Personas 77
  - Strengths of personas as a design tool 78
  - Personas are based on research 80
  - Personas are represented as individual people 81
  - Personas represent groups of users 82
  - Personas explore ranges of behavior 83
  - Personas must have motivations 83
  - Personas can also represent nonusers 84
  - Personas and other user models 84
  - When rigorous personas aren’t possible: Provisional personas 86
- Goals 88
  - Goals motivate usage patterns 88
  - Goals should be inferred from qualitative data 88
  - User goals and cognitive processing 89
  - The three types of user goals 92
  - User goals are user motivations 94
  - Types of goals 94
  - Successful products meet user goals first 96
- Constructing Personas 97
  - Step 1: Identify behavioral variables 98
  - Step 2: Map interview subjects to behavioral variables 99
  - Step 3: Identify significant behavior patterns 99
<table>
<thead>
<tr>
<th>Chapter 6</th>
<th>The Foundations of Design: Scenarios and Requirements</th>
<th>109</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenarios: Narrative as a Design Tool</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>Scenarios in design</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>Using personas in scenarios</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>Different types of scenarios</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>Persona-based scenarios versus use cases</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>Requirements: The “What” of Interaction Design</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>Requirements Definition Using Personas and Scenarios</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>Step 1: Creating problem and vision statements</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Step 2: Brainstorming</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>Step 3: Identifying persona expectations</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>Step 4: Constructing context scenarios</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>Step 5: Identifying requirements</td>
<td>122</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 7</th>
<th>From Requirements to Design: The Framework and Refinement</th>
<th>125</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Design Framework</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>Defining the interaction framework</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>Defining the visual design framework</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>Defining the industrial design framework</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>Refining the Form and Behavior</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>Design Validation and Usability Testing</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>When to test: Summative and formative evaluations</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>Conducting formative usability tests</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>Designer involvement in usability studies</td>
<td>145</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part II</th>
<th>Designing Behavior and Form</th>
<th>147</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 8</td>
<td>Synthesizing Good Design: Principles and Patterns</td>
<td>149</td>
</tr>
<tr>
<td>Interaction Design Principles</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Principles operate at different levels of detail</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Behavioral and interface-level principles minimize work</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>Design Values</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>Ethical interaction design</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>Purposeful interaction design</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>Pragmatic interaction design</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>Elegant interaction design</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>Interaction Design Patterns</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>Architectural patterns and interaction design</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>Recording and using interaction design patterns</td>
<td>157</td>
<td></td>
</tr>
<tr>
<td>Types of interaction design patterns</td>
<td>158</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 9 Platform and Posture 161
Posture 162
Designing Desktop Software 163
Designing for the Web 174
  Informational Web sites 175
  Transactional Web sites 177
  Web applications 178
  Internet-enabled applications 181
  Intranets 181
Other Platforms 182
  General design principles 182
  Designing for handhelds 189
  Designing for kiosks 191
  Designing for television-based interfaces 195
  Designing for automotive interfaces 197
  Designing for appliances 198
  Designing for audible interfaces 199

Chapter 10 Orchestration and Flow 201
Flow and Transparency 201
Designing Harmonious Interactions 203

Chapter 11 Eliminating Excise 223
GUI Excise 224
  Excise and expert users 225
  Training wheels 225
  “Pure” excise 226
  Visual excise 226
  Determining what is excise 228
Stopping the Proceedings 228
  Errors, notifiers, and confirmation messages 228
  Making users ask permission 230
Common Excise Traps 231
  Navigation Is Excise 232
    Navigation among multiple screens, views, or pages 233
    Navigation between panes 233
    Navigation between tools and menus 235
    Navigation of information 236
Improving Navigation 237
  Reduce the number of places to go 238
  Provide signposts 238
  Provide overviews 241
  Provide appropriate mapping of controls to functions 242
  Inflect your interface to match user needs 245
  Avoid hierarchies 247
## Chapter 12 Designing Good Behavior

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designing Considerate Products</td>
<td>250</td>
</tr>
<tr>
<td>Considerate products take an interest</td>
<td>251</td>
</tr>
<tr>
<td>Considerate products are deferential</td>
<td>252</td>
</tr>
<tr>
<td>Considerate products are forthcoming</td>
<td>252</td>
</tr>
<tr>
<td>Considerate products use common sense</td>
<td>253</td>
</tr>
<tr>
<td>Considerate products anticipate human needs</td>
<td>253</td>
</tr>
<tr>
<td>Considerate products are conscientious</td>
<td>253</td>
</tr>
<tr>
<td>Considerate products don’t burden you with their personal problems</td>
<td>254</td>
</tr>
<tr>
<td>Considerate products keep us informed</td>
<td>255</td>
</tr>
<tr>
<td>Considerate products are perceptive</td>
<td>255</td>
</tr>
<tr>
<td>Considerate products are self-confident</td>
<td>256</td>
</tr>
<tr>
<td>Considerate products don’t ask a lot of questions</td>
<td>256</td>
</tr>
<tr>
<td>Considerate products fail gracefully</td>
<td>256</td>
</tr>
<tr>
<td>Considerate products know when to bend the rules</td>
<td>257</td>
</tr>
<tr>
<td>Considerate products take responsibility</td>
<td>259</td>
</tr>
<tr>
<td>Designing Smart Products</td>
<td>260</td>
</tr>
<tr>
<td>Putting the idle cycles to work</td>
<td>260</td>
</tr>
<tr>
<td>Smart products have a memory</td>
<td>261</td>
</tr>
<tr>
<td>Task coherence</td>
<td>263</td>
</tr>
<tr>
<td>Actions to remember</td>
<td>265</td>
</tr>
<tr>
<td>Applying memory to your applications</td>
<td>266</td>
</tr>
</tbody>
</table>

## Chapter 13 Metaphors, Idioms, and Affordances

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Paradigms</td>
<td>270</td>
</tr>
<tr>
<td>Implementation-centric interfaces</td>
<td>270</td>
</tr>
<tr>
<td>Metaphoric interfaces</td>
<td>271</td>
</tr>
<tr>
<td>Idiomatic interfaces</td>
<td>273</td>
</tr>
<tr>
<td>Further Limitations of Metaphors</td>
<td>276</td>
</tr>
<tr>
<td>Finding good metaphors</td>
<td>276</td>
</tr>
<tr>
<td>The problems with global metaphors</td>
<td>276</td>
</tr>
<tr>
<td>Macs and metaphors: A revisionist view</td>
<td>279</td>
</tr>
<tr>
<td>Building Idioms</td>
<td>280</td>
</tr>
<tr>
<td>Manual Affordances</td>
<td>282</td>
</tr>
<tr>
<td>Semantics of manual affordances</td>
<td>284</td>
</tr>
<tr>
<td>Fulfilling user expectations of affordances</td>
<td>284</td>
</tr>
</tbody>
</table>

## Chapter 14 Visual Interface Design

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Art, Visual Interface Design, and Other Design Disciplines</td>
<td>288</td>
</tr>
<tr>
<td>Graphic design and user interfaces</td>
<td>289</td>
</tr>
<tr>
<td>Visual information design</td>
<td>289</td>
</tr>
<tr>
<td>Industrial design</td>
<td>290</td>
</tr>
<tr>
<td>The Building Blocks of Visual Interface Design</td>
<td>290</td>
</tr>
<tr>
<td>Shape</td>
<td>291</td>
</tr>
<tr>
<td>Size</td>
<td>291</td>
</tr>
<tr>
<td>Value</td>
<td>291</td>
</tr>
<tr>
<td>Hue</td>
<td>292</td>
</tr>
</tbody>
</table>
Orientation 292
Texture 292
Position 293
Principles of Visual Interface Design 293
Use visual properties to group elements and provide clear hierarchy 294
Provide visual structure and flow at each level of organization 296
Use cohesive, consistent, and contextually appropriate imagery 302
Integrate style and function comprehensively and purposefully 306
Avoid visual noise and clutter 307
Keep it simple 308
Text in visual interfaces 310
Color in visual interfaces 311
Visual interface design for handhelds and other devices 312
Principles of Visual Information Design 313
Enforce visual comparisons 314
Show causality 314
Show multiple variables 314
Integrate text, graphics, and data in one display 315
Ensure the quality, relevance, and integrity of the content 315
Show things adjacent in space, not stacked in time 316
Don't de-quantify quantifiable data 317
Consistency and Standards 317
Benefits of interface standards 317
Risks of interface standards 318
Standards, guidelines, and rules of thumb 318
When to violate guidelines 319
Consistency and standards across applications 319

Part III Designing Interaction Details 321
Chapter 15 Searching and Finding: Improving Data Retrieval 323
Storage and Retrieval Systems 324
Storage and Retrieval in the Physical World 324
Everything in its place: Storage and retrieval by location 324
Indexed retrieval 325
Storage and Retrieval in the Digital World 326
Relational Databases versus Digital Soup 330
Organizing the unorganizable 330
Problems with databases 331
The attribute-based alternative 332
Natural Language Output: An Ideal Interface for
Attribute-Based Retrieval 333

Chapter 16 Understanding Undo 335
Users and Undo 335
User mental models of mistakes 336
Undo enables exploration 336
Designing an Undo Facility 337
Types and Variants of Undo
  Incremental and procedural actions
  Blind and explanatory Undo
  Single and multiple Undo
  Redo
  Group multiple Undo
Other Models for Undo-Like Behavior
  Comparison: What would this look like?
  Category-specific Undo
  Deleted data buffers
  Versioning and reversion
  Freezing
  Undo-Proof Operations

Chapter 17 Rethinking Files and Save
What's Wrong with Saving Changes to Files?
Problems with the Implementation Model
  Closing documents and removing unwanted changes
  Save As
  Archiving
Implementation Model versus Mental Model
Dispensing with the Implementation Model
Designing with a Unified File Model
  Automatically saving
  Creating a copy
  Naming and renaming
  Placing and moving
  Specifying the stored format
  Reversing changes
  Abandoning all changes
  Creating a version
  A new File menu
  A new name for the File menu
  Communicating status
Are Disks and File Systems a Feature?
Time for Change

Chapter 18 Improving Data Entry
Data Integrity versus Data Immunity
  Data immunity
  What about missing data?
  Data entry and fudgeability
Auditing versus Editing

Chapter 19 Pointing, Selecting, and Direct Manipulation
Direct Manipulation
  Pointing Devices
    Using the mouse
    Mouse buttons
Combutcons 447
List controls 449
Combo boxes 455
Tree controls 457
Entry Controls 457
  Bounded and unbounded entry controls 457
  Spinners 459
  Dials and Sliders 460
  Thumbwheels 462
  Other bounded entry controls 462
  Unbounded entry: Text edit controls 463
Display Controls 468
  Text controls 468
  Scrollbars 469
  Splitters 471
  Drawers and levers 472

Chapter 22 Menus 473
  A Bit of History 473
    The command-line interface 474
    Sequential hierarchical menus 474
    The Lotus 1-2-3 interface 476
    Drop-down and pop-up menus 478
  Menus Today: The Pedagogic Vector 479
    Standard menus for desktop applications 481
    File (or document) 482
    Edit 482
    Windows 483
    Help 483
  Optional Menus 484
    View 484
    Insert 484
    Settings 484
    Format 484
    Tools 485
  Menu Idioms 485
    Cascading menus 485
    Menus 486
    The ribbon 487
    Bang menus 488
    Disabled menu items 489
    Checkmark menu items 489
    Icons on menus 490
    Accelerators 490
    Access keys 491
    Menus on other platforms 492
## Chapter 23  Toolbars
- Toolbars: Visible, Immediate Commands 493
- Toolbars versus Menus 494
- Toolbars and Toolbar Controls 495
  - Icons versus text on toolbars 495
  - The problem with labeling butcons 496
- Explaining Toolbar Controls 496
  - Balloon help: A first attempt 497
  - ToolTips 497
- Disabling toolbar controls 498
- Evolution of the Toolbar 499
  - State-indicating toolbar controls 499
  - Menus on toolbars 499
  - Movable toolbars 500
  - Customizable toolbars 501
  - The ribbon 502
  - Contextual toolbars 503

## Chapter 24  Dialogs
- Appropriate Uses for Dialog Boxes 505
- Dialog Box Basics 507
- Modal Dialog Boxes 509
- Modeless Dialog Boxes 509
  - Modeless dialog issues 510
  - Two solutions for better modeless dialogs 510
- Four Different Purposes for Dialogs 516
  - Property dialog boxes 516
  - Function dialog boxes 517
  - Process dialog boxes 518
  - Eliminating process dialogs 520
  - Bulletin dialog boxes 522
- Managing Content in Dialog Boxes 523
  - Tabbed dialogs 523
  - Expanding dialogs 526
  - Cascading dialogs 527

## Chapter 25  Errors, Alerts, and Confirmation
- Error Dialogs 529
  - Why we have so many error messages 530
  - What’s wrong with error messages 530
  - Eliminating error messages 534
  - Aren’t there exceptions? 536
  - Improving error messages: The last resort 537
- Alert Dialogs: Announcing the Obvious 539
- Confirmation Dialog 541
  - The dialog that cried “Wolf!” 542
  - Eliminating confirmations 543
Chapter 26 Designing for Different Needs 551
Command Vectors and Working Sets 551
Immediate and pedagogic vectors 552
Working sets and personas 552
Graduating Users from Beginners to Intermediates 553
World vectors and head vectors 553
Memorization vectors 554
Personalization and Configuration 555
Idiosyncratically Modal Behavior 557
Localization and Globalization 558
Galleries and Templates 559
Help 560
The index 560
Shortcuts and overview 561
Not for beginners 561
Modeless and interactive help 561
Wizards 561
"Intelligent" agents 562
Afterword: On Collaboration 565
Appendix A Design Principles 569
Appendix B Bibliography 575
Index 581
Foreword: The Postindustrial World

The industrial age is over. Manufacturing, the primary economic driver of the past 175 years, no longer dominates. While manufacturing is bigger than ever, it has lost its leadership to digital technology, and software now dominates our economy. We have moved from atoms to bits. We are now in the postindustrial age.

More and more products have software in them. My stove has a microchip in it to manage the lights, fan, and oven temperature. When the deliveryman has me sign for a package, it’s on a computer, not a pad of paper. When I shop for a car, I am really shopping for a navigation system.

More and more businesses are utterly dependent on software, and not just the obvious ones like Amazon.com and Microsoft. Thousands of companies of all sizes that provide products and services across the spectrum of commerce use software in every facet of their operations, management, planning, and sales. The back-office systems that run big companies are all software systems. Hiring and human resource management, investment and arbitrage, purchasing and supply chain management, point-of-sale, operations, and decision support are all pure software systems these days. And the Web dominates all sales and marketing. Live humans are no longer the front line of businesses. Software plays that role instead. Vendors, customers, colleagues, and employees all communicate with companies via software or software-mediated paths.

The organizational structures and management techniques that have worked so well in the past for manufacturing-based companies are failing us today in the postindustrial age. They fail because they focus on the transformation and movement of things made out of atoms. There are only finite amounts of desirable atoms and it takes lots of energy to transform and transport them. Software—made out of bits, not atoms—is qualitatively different. There is an infinite quantity of bits and virtually no energy is needed to transform, transport, or even replicate them.
The people who make software are different as well. The average computer programmer and the average assembly line worker are qualitatively different in their aptitude, attitude, training, language, tools, and value systems. The most effective ways of supervising, tracking, and managing programmers are dramatically different from those used so successfully with blue-collar workers of an earlier age. Getting programmers to do what is best for the company requires skills unknown to the industrial-age executive.

Reducing the cost of manufacturing was the essential contribution of industrialization. Thus the best and brightest minds of an earlier age applied themselves to reducing the amount of money spent creating products. In the postindustrial age, the costs of raw materials, product assembly, and shipping are equally low for all players. The only significant leverage to lower manufacturing costs comes through automation, planning, and business intelligence: that is, software. In other words, instead of saving a dollar on the construction of each widget, you save a million dollars by making the precisely needed quantity of the most desirable product.

Once a software program has been successfully written, it can be reproduced an unlimited number of times for virtually nothing. There is little benefit in reducing the cost of writing it. Reducing the amount one spends on software construction usually means compromising the quality, so the primary business equation of the industrial age is reversed today. The best and brightest minds of today apply themselves to increasing the effectiveness of software and the quality of its behavior. Keep in mind that all modern financial accounting systems focus on tracking manufacturing costs and no longer accurately represent the state of our software-dominated businesses. Making executive decisions on these erroneous numbers causes significant waste of time, money, and opportunity.

It’s no wonder that companies struggle with software. Very capable executives find that their intentions are subtly but significantly altered somewhere along the path from conception to release. What appeared to be a sound plan turns out to be inadequate for shepherding the software construction process. It’s time to let go of obsolete industrial-age management methods and adopt interaction design as the primary tool for designing and managing software construction.

Since About Face was first published in 1995, the practice of interaction design has grown and matured enormously. Dominated for so long by simple ex post facto, trial-and-error methods, interaction design—along with its many siblings and variants—has matured into a clear, dependable, effective tool for determining what behavior will succeed. The invention and development of personas, the refinement of written behavioral blueprints, and the entire practice of Goal-Directed™ Design, have made high-quality software behavior achievable by any organization with the will to create it.
What’s more, interaction design has emerged as an incredibly powerful software construction management tool. Because it is a description of the software as it will be when it is finally written, it acts as a blueprint, not only helping programmers know what to build but also helping managers measure the progress of the programmers.

Interaction design has also shown its power as a marketing tool, communicating with great clarity and specificity about exactly whom will be using the product and why. Getting to the root of customer motivations is manna for marketers, and the qualitative research and analysis aspects of Goal-Directed Design provide significant market insight.

Especially since the Web revolution—when tossing common sense overboard seemed to be the path to instant riches—I’ve heard many intelligent people who really should know better say, “It is simply not possible to know what the user wants!” While this assertion certainly absolves them of not, in fact, knowing what the user wants, it is boldly, obviously, incredibly false. At my company, Cooper, clients bring our designers into the complex worlds of finance, health care, pharmaceuticals, human resources, programming tools, museums, consumer credit, and any number of disparate fields. Our teams, none of whom have any training in—or typically even any exposure to—the particular subject matter at hand, routinely become sufficiently expert in only a few weeks to astonish our clients. We can do this because our point of departure is relentlessly human-centered, rather than technology-centered.

Interaction design is a tool for “Knowing what the user wants.” Armed with that knowledge, you can create better, more successful, bit-empowered products, and you can sell them for more money. What’s more, you will reach your market with a loyalty-inducing, better solution. Time and time again we have seen feature-loaded products early to market get trounced by later entries whose behavior has been better thought out. Imagine getting that thinking done before the first release ever has a chance to commit you to a nonoptimal strategy.

Nothing succeeds like success, and the success of the practical application of the principles and methods put forth in this book—and others like it—are clearly demonstrating that software isn’t really as soft as many people first thought, and that thorough user research and detailed planning are more necessary than ever in the postindustrial age.

If you are committed to improving the world by improving the behavior of digital products and services, then I welcome you to the world of About Face.

—Alan Cooper
Acknowledgments

We’d like to express our deepest gratitude to the following individuals, without whom this new edition of About Face would not have been possible: Chris Webb at Wiley, who saw the time was right for a new edition; Sue Cooper, who shared that vision; and Sara Shlaer at Wiley, who has patiently helped us shape multiple editions of this book.

We would also like to thank the following colleagues and Cooper designers for their contributions to this volume and the previous, for which we are greatly indebted: Kim Goodwin, who has contributed significantly to the development and expression of the concepts and methods described in these pages; Rebecca Bortman and Nick Myers who overhauled the book and cover designs, as well as the illustrations; Hugh Dubberly, for his help in developing the principles at the end of Chapter 8 and for his assistance in clarifying the Goal-Directed process with early versions of the diagrams found in Chapter 1; Gretchen Anderson, Elaine Montgomery, and Doug LeMoine for their contributions on user and market research in Chapter 4; Rick Bond for his many insights about usability testing featured in Chapter 7; Ernest Kinsolving and Joerg Beringer at SAP for their contributions on the posture of Web portals in Chapter 9; Chris Weeldreyer for his insights into the design of embedded systems in Chapter 9; Wayne Greenwood for his contributions on control mapping in Chapter 10; and Nate Fortin and Nick Myers for their contributions on visual interface design and branding in Chapter 14. 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, Doug LeMoine, Tim McCoy, Elaine Montgomery, Nick Myers, Chris Noessel, Ryan Olshavsky, Angela Quail, Suzy Thompson, and Chris Weeldreyer for their contributions to the Cooper designs and illustrations featured in this volume.

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 wish also to thank the many other clients who have had the vision and the foresight to work with us and support us in their organizations.

We would also 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.

Finally, it should be noted that the parts of Chapter 5 concerned with cognitive processing originally appeared in an article by Robert Reimann on UXMatters.com, and are used with permission.
Introduction to the Third Edition

This book is about interaction design—the practice of designing interactive digital products, environments, systems, and services. Like many 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—we routinely create products that exhibit complex behavior.

Take something as basic as an oven. Before the digital age, the operation of an oven was quite simple—it involved turning a single knob to the correct position. There was one position for off, and one position for any oven temperature one might want to use. Every single time a person turned that knob to a given position, the same thing happened. One might call this “behavior,” but it is certainly quite simple and mechanistic behavior. Compare this to our modern-day ovens with silicon chips and LCD screens. They are endowed with buttons that say non-cooking-related things like Start, Cancel, Program, as well as the more expected Bake and Broil. What happens when you press any one of these buttons is quite a lot less predictable than what happened when you turned the knob on your old gas range. In fact, the results of pressing one of the buttons is entirely dependent on the state of the oven and what other buttons you might have pressed previously. 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 a sum of its parts, with its own unique methods and practices. And to be clear—it is very much a design
discipline, quite different from science and engineering. While it should always be practiced in a rational and considered manner, interaction design is about synthesis and imagining things as they might be, not necessarily as they currently are.

Interaction design is also 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. In this book we describe a particular approach to interaction design that we call the Goal-Directed method. We’ve found that when a designer focuses 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 powerful and pleasurable.

As even the most casual observer of developments in technology must have noticed, interactive products can become very complex very quickly. While a mechanical device may be capable of a dozen visible states, a digital product may be capable of being in 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 very 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 of assessing 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 isn’t possible 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, who were working on the first laptop computer, the GRiD Compass, 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.
At the time *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 under the shadow of software engineering, rather like the tiny, quick-witted mammals that scrambled under the shadows of hulking tyrannosaurs. “Software design,” as the first edition of *About Face* referred to it, was poorly understood and underappreciated, and, when it was practiced at all, it was usually practiced by programmers. 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, “ease of use” was a term on everyone’s lips. Traditional design professionals, who had dabbled in digital product design during the short-lived popularity of “multimedia” in the early nineties, leapt to the Web en masse. Seemingly new design titles sprang up like weeds: information designer, information architect, user experience strategist, and interaction designer. For the first time ever, C-level executive positions were established to focus on creating user-centered products and services, such as the 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 interaction design idioms back by more than a decade, it inarguably placed user requirements on the radar of the corporate world for good. Since the second edition of *About Face* was published in 2003, the user experience of digital products has become front page news in the likes of *Time* magazine and *BusinessWeek*, and institutions such as Harvard Business School and Stanford have 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 what they want is good technology: technology that has been 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, and a mailing list and steering committee were founded shortly thereafter by Challis Hodge, David Heller, Rick Cecil, and Jim Jarrett. In September of 2005, IxDA, the Interaction Design Association ([www.ixda.org](http://www.ixda.org)) was officially incorporated. At the time of writing, it has over 2000 members in over 20 countries. We’re pleased to say that Interaction Design is finally beginning to come into its own as both a discipline and a profession.
Why Call It Interaction Design?

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 certainly had better longevity. We still use it occasionally in this book, specifically to connote the arrangement of widgets on the screen. However, what is discussed in this book is 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 what it does.

As we’ve discussed, interaction designers have borrowed practices from more established design disciplines, but have also evolved beyond them. Industrial designers have attempted to address the design of digital products, but like their counterparts in graphic design, their focus has traditionally 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.

In recent years, a number of new terms have been proposed for this type of design. As the World Wide Web gained prominence, information architecture (IA) emerged as a discipline dedicated to solving problems dealing with navigation to and the “findability” of content, mostly (though not exclusively) within the context of Web sites. While clearly a close relative of interaction design, mainstream IA still retains a somewhat limited, Web-centric view of organizing and navigating content using pages, links, and minimally interactive widgets. However, recent industry trends such as Web 2.0 and rich Internet applications have begun to open the eyes of Web designers, causing them to look beyond archaic browser interaction idioms. We believe this awakening is bringing information architects’ concerns ever more closely in alignment with those of interaction designers.

Another term that has gained popularity is *user experience* (UX). There are many who advocate for the use of this term as an umbrella under which many 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 volume: how specifically to design the behavior of complex interactive systems. While 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, we believe there are specific methods appropriate to designing for the world of bits.

We also wonder whether it is actually possible to *design* an experience. Designers of all stripes hope to manage and *influence* the experiences people have, but this is done...
by carefully manipulating the variables intrinsic to the medium at hand. A graphic designer creating a poster uses an arrangement of type, 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, and an interior designer uses layout, lighting, materials, and even sound to help create an experience.

Extending this thinking to 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, there are many cases where a design project requires careful attention to the orchestration of a number of design disciplines to achieve an appropriate user experience (see Figure 1). It is to these situations that we feel the term experience design is most applicable.

Figure 1: One can think of user experience design (UX) of digital products as consisting of three overlapping concerns: form, behavior, and content. Interaction design is focused on the design of behavior, but is also concerned with how that behavior relates to form and content. Similarly, information architecture is focused on the structure of content, but is also concerned with behaviors that provide access to content, and the way 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.
Working with the Product Team

In addition to defining interaction design in terms of its primary concern with behavior and its relationships with other design disciplines, we also often find it necessary to define how interaction design should fit within an organization. We believe that establishing a rigorous product development process that incorporates design as an equal partner with engineering, marketing, and business management, and that includes well-defined responsibilities and authority for each group, greatly increases the value a business can reap from design. The following division of responsibilities, balanced by an equal division of authority, can dramatically improve design success and organizational support of the product throughout the development cycle and beyond:

- **The design team** has responsibility for users’ satisfaction with the product. Many organizations do not currently hold anyone responsible for this. To carry out this responsibility, designers must have the authority to decide how the product will look, feel, and behave. They also need access to information: They must observe and speak to potential users about their needs, to engineers about technological opportunities and constraints, to marketing about opportunities and requirements, and to management about the kind of product to which the organization will commit.

- **The engineering team** has responsibility for the implementation and fabrication of the product. For the design to deliver its benefit, engineering must have the responsibility for building, as specified, the form and behaviors that the designers define, while keeping on budget and on schedule. Engineers, therefore, require a clear description of the product’s form and behaviors, which will guide what they build and drive their time and cost estimates. This description must come from the design team. Engineers must also contribute to discussions of technical constraints and opportunities, as well as the feasibility of proposed design solutions.

- **The marketing team** has responsibility for convincing customers to purchase the product, so they must have authority over all communications with the customer, as well as input into the product definition and design. In order to do this, the team members need access to information, including the results of designers’ research, as well as research of their own. (It’s worth noting that, as we discuss further in Chapters 4 and 5, customers and users are often different people with different needs.)

- **Management** has responsibility for the profitability of the resulting product, and therefore has the authority to make decisions about what the other groups will work on. To make those decisions, management needs to receive clear information from the other groups: design’s research and product definition, marketing’s research and sales projections, and engineering’s estimations of the time and cost to create the product.
What This Book Is and What It Is Not

In this book, we attempt to provide readers with 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 go about understanding and defining 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 with this volume has been to create a book that weaves all three of these tools together. While helping you design more effective and useful dialog boxes and menus, this book will simultaneously help you understand how users comprehend and interact with your digital product, and 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 14 why the utility of such tools is limited and relevant only to specific circumstances. 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 weak at answering why. This book attempts to address these unanswered questions.

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

While 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 7, but urge you to refer to the significant and ever-growing body of usability literature for more detailed information about conducting and analyzing usability tests.

Changes from the Previous Editions

Much in the world of interface design has changed since the first edition of About Face was published in 1995. However, much remains the same. The third edition of About Face retains what still holds true, updates those things that have changed, and provides new material reflecting not only how the industry has changed in the last 11 years but also new concepts that we have developed in our practice to address the changing times.

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

- The book has been reorganized to present its ideas in a more easy-to-use reference structure. The book is divided into three parts: The first deals with process and high-level ideas about users and design, the second deals with high-level interaction design principles, and the third deals with lower-level interface design principles.
- The first part describes the Goal-Directed Design process in much greater detail than in the second edition, and more accurately reflects current practices at Cooper, including research techniques, the creation of personas, and how to use personas and scenarios to synthesize interaction design solutions.
- Throughout the book, we attempt to more explicitly discuss visual interface design concepts, methods and issues, as well as issues related to a number of platforms beyond the desktop.
- Terminology and examples in the book have been updated to reflect the current state of the art in the industry, and the text as a whole has been thoroughly edited to improve clarity and readability.

We hope that readers will find these additions and changes provide a fresh look at the topics at hand.
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—this is where the greatest need exists for understanding how to create effective, Goal-Directed user interfaces.

Having said this, most of the material in this book transcends platform. It is equally applicable to all desktop platforms—Mac OS, Linux, and others—and the majority of it is relevant even for more divergent platforms such as kiosks, handhelds, embedded systems, and others.

A good portion of examples in this book are from the Microsoft Word, Excel, PowerPoint, Outlook, and Internet Explorer, and Adobe Photoshop and Illustrator. We have tried to stick with examples from these mainstream applications for two reasons. First, readers are likely to be at least slightly 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. We have included a few examples from more exotic applications as well, in places where they were particularly illustrative.

A few examples in this new edition come from now moribund software or OS versions. These examples illustrate particular points that the authors 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 of this book 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. Programmers, designers of all stripes involved with digital product design, usability professionals, and project managers will all find something useful in this volume. People who have read earlier editions of About Face or The Inmates Are Running the Asylum will find new and updated information about design methods and principles here.

We hope this book informs you and intrigues you, but 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 just want to talk to us, we’d be happy to hear from you at alan@cooper.com, rmreimann@gmail.com, and dave@cooper.com.
Part I

Understanding Goal-Directed Design

Chapter 1
Goal-Directed Design

Chapter 2
Implementation Models and Mental Models

Chapter 3
Beginners, Experts, and Intermediates

Chapter 4
Understanding Users: Qualitative Research

Chapter 5
Modeling Users: Personas and Goals

Chapter 6
The Foundations of Design: Scenarios and Requirements

Chapter 7
From Requirements to Design: The Framework and Refinement
Goal-Directed Design

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

On the surface, this premise sounds quite obvious and straightforward: 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?

Digital Products Need Better Design Methods

Most digital products today emerge from the development process like a creature emerging from a bubbling tank. Developers, instead of planning and executing with a mind towards satisfying the needs of the people who purchase and use their products, end up creating technologically focused solutions that are difficult to use and control. Like mad scientists, they fail because they have not imbued their creations with humanity.
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 users’ desires, needs, motivations, and contexts
- 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 is 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 will vary depending on what is being designed. For example, an informational Web site may require particular attention to content, whereas the design of a chair is primarily concerned with form. As we discussed in the Introduction, interactive digital products are uniquely imbued with complex behavior.

When performed using the appropriate methods, design can provide the missing human connection in technological products. But clearly, most current approaches to the design of digital products aren’t working as advertised.

The creation of digital products today

Digital products come into this world subject to the push and pull of two, often opposing, forces — developers and marketers. 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 are able to 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.) Unfortunately, reducing an interactive product to a list of hundreds of features doesn’t lend itself to the kind of graceful orchestration that is required to make complex technology useful. Adding “easy to use” to the list of requirements 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 users. Good developers are focused on solving challenging technical problems, following good engineering practices, and meeting deadlines. They are often given incomplete, confusing, and sometimes contradictory instructions and are forced to make significant decisions about the user experience with little time or background.

Thus, the people who are most often responsible for the creation of our digital products rarely take into account the users’ goals, needs, or motivations, and at the same time 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, reduce productivity, and fail to meet user needs. Figure 1-1 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 it becomes a surface-level patch on shoddy interactions — “lipstick on the pig,” as one of our clients once referred to it. The design process, as we will soon discuss, should precede coding and testing to ensure that products truly meet the needs of users.

In the dozen years since the publication of the first edition of this book, software and interactive products have certainly improved. Many companies have begun to focus on serving the needs of people with their products, and are spending the time and money to do upfront design. Many more companies are still failing to do this, and as they maintain their focus on technology and marketing data, they continue to create the kind of digital products we’ve all grown to despise. Here are a few symptoms of this affliction.

**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 in Figure 1-2 pop up like weeds announcing that the user has failed yet again. These messages also demand that the user acknowledge his failure by agreeing: OK.
The evolution of the software development process. The first diagram depicts the early days of the software industry when smart programmers 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, testing became a discipline in its own right, and 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.
Figure 1-2 Thanks for sharing. Why didn't the program notify the library? What did it want to notify the library about? Why is it telling us? And what are we OKing, anyway? It is not OK that the program failed!

Digital products and software frequently interrogate users, peppering them with a string of terse questions that they are neither inclined or 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.

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. For example, the feature-rich Palm Treo smartphone doesn’t anticipate that a user might want to add the phone number of someone who has just called to an existing contact. It doesn’t take a lot of research or imagination to deduce that this is something that many users will want to do, but nevertheless one is forced to go through a complicated maze involving copying the phone number, navigating to the contact in question, and pasting into the appropriate field.

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 not consistent with the way 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. Programs often express themselves in incomprehensible jargon that cannot be fathomed by normal users (“How many stop bits?”) and are sometimes incomprehensible even to experts (“Please specify IRQ.”).
Digital products exhibit poor behavior
If a 10-year-old child behaved like some software programs or devices, he’d be sent to his room without supper. Programs forget to shut the refrigerator door, leave 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 program once again asks you if you want to save it! Evidently the act of printing caused the program to think the document had changed, even though it did not. Sorry, Mom, I didn’t hear you.

Programs often require us to step out of the main flow of tasks to perform functions that should fall immediately to hand. Dangerous commands, however, are often presented right up front where unsuspecting users can accidentally trigger them. The overall appearance of many programs is overly complex and confusing, making navigation and comprehension difficult.

Digital products require humans to do the heavy lifting
Computers and their silicon-enabled brethren are supposed 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 and horrified by how much work they are forced to do to manage the operation of software. This work can be anything from manually keying values from one window into another, to copying and pasting between applications that don’t otherwise speak to each other, to the ubiquitous clicking and pushing and pulling of windows around the screen to access hidden functionality that people use every day to do their job.

Why are these products so bad?
So what, then, is the real problem? Why is the technology industry generally so inept at designing the interactive parts of digital products? There are three primary reasons: ignorance about users, a conflict of interest between serving human needs and construction priorities, and the lack of a process for understanding human needs as an aid to developing appropriate product form and behavior.

Ignorance about users
It’s a sad 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 the users. We might know what market segment our users are in, how much money they make, how much money they like to
spend on weekends, and what sort of cars they buy. Maybe we even have a vague idea what kind of jobs they have and some of the major tasks that 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? Unfortunately, it does not.

We’ll soon see how to address the issue of understanding users and their behaviors with products.

**Conflicting interests**

A second problem affects the ability of vendors and manufacturers to make users happy. There is an important conflict of interest in the world of digital product development: The people who build the products — programmers — are usually also the people who design them. Programmers are often required to choose between ease of coding and ease of use. Because programmers’ 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 programmer to advocate effectively for the user, the business, and the technology all at the same time.

**The lack of a process**

The third reason the digital technology industry isn’t cranking out successful 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 *transforming an understanding of users into products that both meet their needs and excite their imaginations*.

When we think about complex mechanical devices, we take for granted that they have been carefully designed for use, in addition to being engineered. Most manufactured objects are quite simple, and even complex mechanical products are quite
simple when compared to most software and software-enabled products that can be compiled from over one million lines of code (compare this to a mechanical artifact of overwhelming complexity such as the space shuttle, which has 250,000 parts, only a small percentage of which are moving parts). Yet most software has never undergone a rigorous design process from a user-centered perspective.

In the worst case, decisions about what a digital product will do and how it will communicate with users is simply a byproduct of its construction. Programmers, deep in their thoughts of algorithms and code, end up “designing” product behaviors and user interfaces the same way that miners end up “designing” the landscape 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 programmers today embrace the notion that integrating customers directly into the development process on a frequent basis 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: a confusion of domain knowledge with design knowledge. Customers, although they might be able to articulate the problems with an interaction, are not often capable of visualizing the solutions to those problems. Design is a specialized skill, just like programming. Programmers 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.

This doesn’t mean that designers shouldn’t be interested in getting feedback on their proposed solutions. However, each member of the product team should respect the others’ areas of expertise. Imagine a patient who visits his doctor with a horrible stomachache. “Doctor,” he says, “it really hurts. I think it’s my appendix. You’ve got to take it out as soon as possible.” Of course, a responsible physician wouldn’t perform the surgery without question. The patient can express the symptoms, but it takes the doctor’s professional knowledge to make the correct diagnosis.

To better understand how to create a workable process that brings user-centered design to digital products, it’s useful to understand a bit more about the history of design in manufacturing and about how the challenges of interactive products have substantially changed the demands on design.
The Evolution of Design in Manufacturing

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 one of these three foundations is significantly weak in a product, it is unlikely to stand the test of time.

Now enter the computer, the first machine created by humans that is capable of almost limitless behavior when properly coded into software. 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 other aspects of an interactive product become marginal. Who pays attention to the black box that sits under your desk — it is the interactive screen, keyboard, and mouse to which users pay attention. Yet, the interactive behaviors of software and other digital products, which should be receiving the lion’s share of design attention, all too frequently receive no attention at all.

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. Design of behavior requires an understanding of the user’s relationship with the product from before purchase to end-of-life. Most important of all is the understanding of how the user wishes to use the product, in what ways, and to what ends.
Building successful digital products. The diagram indicates the three major processes that 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.

**Figure 1-3** Building successful digital products. The diagram indicates the three major processes that 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.
Planning and Designing 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 supports and facilitates 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.

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, but they must work in harmony with the larger concern of achieving user goals via appropriately designed behaviors.

This book presents a set of methods to address the needs of this new kind of behavior-oriented design, which addresses the goals (Rudolf, 1998) and motivations of users: Goal-Directed Design. To understand Goal-Directed Design, we first need to better understand user goals and how they provide 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 they are forced to do 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 is more likely concentrating 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; personal goals of users need to be addressed. When the user’s personal goals are met by the design, business goals are far more effectively achieved, for reasons we’ll explore in more detail in later chapters.

If you examine most commercially available software, Web sites, and digital products today, 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 don’t have the right priorities. Most focus their attention far too narrowly on implementation issues, which distract them from the needs of users.

Even when businesses become sensitive to their users, they are often powerless to change their products because 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 a program serve users’ goals once there is a significant and inflexible code base in place.

Finally, when companies do focus on the users, they tend to pay too much attention to the tasks that 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 describes a hierarchy in which activities are composed of tasks, which are in turn composed of actions, which are then themselves composed of operations. Using this scheme, Norman advocates “Activity-Centered Design” (ACD), which focuses first and foremost on understanding activities. His claim is that humans adapt to the tools at hand, and understanding the activities that 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, and which has in recent years been adapted to the study of human-computer interaction, most notably by Bonnie Nardi.

Norman concludes, correctly, that the traditional task-based focus of digital product design has yielded inadequate results. Many developers and usability professionals still approach the design of interfaces by asking, “What are the tasks?” 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 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 that it goes quite far enough. While a method like ACD can be very useful in properly breaking down the “what” of user behaviors, it really doesn’t address what should be the first question asked by any designer: 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 the expectations and aspirations of your users, which can in turn 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, since they are based almost entirely on whatever technology is
at hand. For example, when traveling from St. Louis to San Francisco, a person’s 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 aircraft 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 the goals of a corporation without meeting the goals of their users. 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 interfaces easier to learn should always be a design target. Ease of learning is an important guideline, but in reality, as Brenda Laurel notes, the design target really depends on the context — who the users are, what they are doing, and what goals they have. You simply can’t create good design by following rules disconnected from the goals and needs of the users of your product.

Let us illustrate: Take 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 users can route calls, and the rapidity with which those calls can be completed. Ease of learning is also important because it affects the happiness and, ultimately, the turnover rate of employees, 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 and, if necessary, ease of learning should take a back seat. A program 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 5000 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. The design process, which we describe in the remainder of this chapter and detail in the remaining chapters 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.

The Goal-Directed Design Process

Most technology-focused companies don’t have an adequate process for user-centered 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 is quite useful for selling products but falls short of providing critical information about how people actually use products — especially products with complex behaviors (see Chapter 4 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, and 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 has come to mean what happens in the third process diagram shown in Figure 1-1: a visual facelift on the implementation model (see Chapter 2). But design, when properly deployed (as in the fourth process diagram shown in Figure 1-1), 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 goals of users, needs of business, and constraints of technology.

Designers as researchers

If design is to become product definition, designers need to take on a broader role than that assumed in traditional design, particularly when the object of this design 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 1-4). The results of user and market research are analyzed by the usability and market researchers and then thrown over the transom to designers or programmers. 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 1-4 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 for involving designers in the research process. One of the most powerful tools designers bring to the table 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 4 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, there is still a translation gap between research results and design details. The puzzle is missing several pieces, as we will discuss next.

**Between research and design: 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, workflow, and translation of functions into interface controls, but is less useful for defining the basic framework of what a product is, what it does, and how it should meet the broad needs of the user.

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 1-5). 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.
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 the needs and goals of users, 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 1-5). 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.

![Figure 1-5 The Goal-Directed Design process.](image)

The remainder of this chapter provides a high-level view of the five phases of Goal-Directed Design, and Chapters 4–7 provide more detailed discussion of the methods involved in each of these phases. See Figure 1-6 for 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, reviews of market research and technology white papers and brand strategy, as well as 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 4 provides a more detailed discussion of Goal-Directed research techniques.

Modeling
During the Modeling phase, behavior and workflow patterns discovered through analysis of the field research and interviews are synthesized into domain and user models. Domain models can include information flow and workflow 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 serve as the main characters in a narrative, scenario-based approach to design that iteratively generates design concepts in the Framework Definition phase, provides feedback that enforces design coherence and appropriateness in the Refinement phase, and represents a powerful communication tool that helps developers and managers to understand design rationale and to 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 there are no gaps or duplications.

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

A detailed discussion of persona and goal development can be found in Chapter 5.

Requirements Definition
Design methods employed by teams during the Requirements Definition phase provide the much-needed connection between user and other models and the framework of the design. 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 provide an understanding of 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 an analysis of 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 that 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.

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. Chapters 2 and 3 and the whole of Part II are 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, and more recently brought to the programming field by Erich Gamma, et al. 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 gross 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 vectors** 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 go off and create concepts independent of the product’s behavior.

As soon as an interaction framework begins to emerge, visual interface designers produce several options for a **visual framework**, which is sometimes also referred to as 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** (walkthrough) and **validation scenarios** focused on storyboarding paths through the interface in high 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**, delivered in either paper or interactive media as context dictates. Chapter 6 discusses in more detail the use of personas, scenarios, principles, and patterns in the Requirements Definition, 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.
## Part I: Understanding Goal-Directed Design

![Goal-Directed Design Process Diagram](image)

### Figure 1-6: A more detailed look at the Goal-Directed Design process.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Concerns</th>
<th>Stakeholder Collaboration</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope</strong></td>
<td>define project goals &amp; schedule</td>
<td>Objectives, timelines, financial constraints, process, milestones</td>
<td>Meetings</td>
</tr>
<tr>
<td><strong>Audit</strong></td>
<td>Review existing work &amp; product</td>
<td>Business &amp; marketing plans, branding strategy, market research, product portfolio plans, competitors, relevant technologies</td>
<td>Capabilities &amp; Scoping</td>
</tr>
<tr>
<td><strong>Stakeholder Interviews</strong></td>
<td>Understand product vision &amp; constraints</td>
<td>Product vision, risks opportunities, constraints, logistics, users</td>
<td>Interviews with stakeholders &amp; users</td>
</tr>
<tr>
<td><strong>User interviews &amp; observations</strong></td>
<td>Understand user needs &amp; behavior</td>
<td>Users, potential users, behaviors, attitudes, aptitudes, motivations, environments, tools, challenges</td>
<td>Check-in</td>
</tr>
<tr>
<td><strong>Persons</strong></td>
<td>User &amp; customer archetypes</td>
<td>Patterns in user &amp; customer behaviors, attitudes, aptitudes, goals, environments, tools, challenges</td>
<td>Check-in</td>
</tr>
<tr>
<td><strong>Other Models</strong></td>
<td>Represent domain factors beyond individual users &amp; customers</td>
<td>Workflows among multiple people, environments, artifacts</td>
<td></td>
</tr>
<tr>
<td><strong>Context Scenarios</strong></td>
<td>Tell stories about ideal user experiences</td>
<td>How the product fits into the personas life &amp; environment &amp; helps them achieve their goals</td>
<td>Check-in</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>Describe necessary capabilities of the product</td>
<td>Functional &amp; data needs, user mental models, design imperatives, product vision, business requirements, technology</td>
<td>Presentation</td>
</tr>
<tr>
<td><strong>Design Framework</strong></td>
<td>Define manifestations of information &amp; functionality</td>
<td>Information, functions, mechanisms, actions, domain object models</td>
<td>Check-ins</td>
</tr>
<tr>
<td><strong>Framework</strong></td>
<td>Design overall structure of user experience</td>
<td>Object relationships, conceptual groupings, navigation sequencing, principles &amp; patterns, flow, sketches, storyboards</td>
<td></td>
</tr>
<tr>
<td><strong>Key Path &amp; Validation Scenarios</strong></td>
<td>Describe how the persona interacts with the product</td>
<td>How the design fits into an ideal sequence of user behaviors, &amp; accommodates a variety of likely conditions</td>
<td>Presentation</td>
</tr>
<tr>
<td><strong>Detailed design</strong></td>
<td>Refine &amp; specify details</td>
<td>Appearance, idioms, interface, widgets, behavior, information, visualization, brand, experience, language, storyboards</td>
<td>Check-ins</td>
</tr>
<tr>
<td><strong>Design modification</strong></td>
<td>Accommodate new constraints &amp; timeline</td>
<td>Maintaining conceptual integrity of the design under changing technology constraints</td>
<td>Collaborative</td>
</tr>
<tr>
<td><strong>Document</strong></td>
<td></td>
<td></td>
<td>Revision</td>
</tr>
</tbody>
</table>
Goals, not features, are the key to product success

Developers and marketers often use the language of features and functions to discuss products. This is only natural. Developers build software function by function, and a list of features is certainly one way to express a product’s value to potential customers (though this is clearly limiting, as well). The problem is that these are abstract concepts that only provide limited insight into how human beings can be effective and happy while using technology.

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.

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.
Implementation Models and Mental Models

The computer industry makes frequent 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 nothing more than a euphemism for forcing human beings to stretch their thinking to understand an alien, machine logic rather than having software-enabled products stretch to meet people’s ways of thinking. In this chapter, we discuss how a poor understanding of users and the specific ways they approach digital products has exacerbated the computer-literacy divide, and how software that better matches how people think and work can help solve the problem.

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 out 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 a program 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 the way a program is implemented in code.

User 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 actually 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 called the user’s 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, one that is powerful enough to cover their interactions with it, but that doesn’t necessarily reflect its actual inner mechanics. For example, many people imagine that, when they plug their vacuum cleaners and blenders into outlets in the wall, the electricity flows like water from the wall to 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 actually traveling up the cord 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 cellular telephone 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 to 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 program’s reactions. When we use a computer to digitally edit sound or to 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.

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 programmer 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 is really getting 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 — the way the designer chooses to represent a program’s functioning to the user. Donald Norman refers to this simply as the designer’s model.

In the world of software, a program’s represented model can (and often should) be quite different from the actual processing structure of the program. 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. The relationship between the three models is shown in Figure 2-1.

The closer the represented model comes to the user’s mental model, the easier he will find the program to use and to understand. Generally, offering a represented model that follows the implementation model too closely significantly reduces the user’s ability to learn and use the program, assuming (as is almost always the case) that the user’s mental model of his tasks differs from the implementation model of the software.
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 program helps them do it 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 program to the user is called the represented model, which, unlike the other two models, is an aspect of software over which designers have great control. One of the most important goals of the designer should be to make the represented model match the mental model of users as closely as possible. It is therefore critical that designers understand in detail the way 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 actual implementation model, we help the user achieve a better understanding. Pressing the brake pedal in your car, for example, may conjure a mental image of pushing a lever that rubs against the wheels to slow you down. The actual mechanism includes hydraulic cylinders, tubing, and metal pads that squeeze on a perforated disk, but we simplify all that out of our minds, creating a more effective, albeit less accurate, mental model. In software, we imagine that a spreadsheet scrolls new cells into view when we click on 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 program synthesizes a new image to display in real time.

Understanding how software actually works always helps someone to use it, but this understanding usually comes at a significant cost. One of the most significant ways in which computers can assist human beings is by putting a simple face on complex processes and situations. 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.
User interfaces should be based on user mental models rather than implementation models.

In Adobe Photoshop, users can adjust the color balance and brightness of an illustration using a feature called Variations. Instead of offering numeric fields for entering color data — the implementation model — the Variations interface shows a set of thumbnail images, each with a different color balance (see Figure 2-2). A user can click on the image that best represents the desired color setting. The interface more closely follows his mental model, because the user — likely a graphic artist — is thinking in terms of how his image looks, not in terms of abstract numbers.

Figure 2–2  Adobe Photoshop has a great example of software design to match user mental models. The Variations interface shows a set of thumbnail images, varying color balance and brightness by adjustable increments. A user can click on the image that best represents the desired color setting. This image then becomes the new default for more varied thumbnails. The interface follows the mental model of graphic artists who are after a particular look, not a set of abstract numerical 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.

Goal-directed interactions reflect user mental models.

A user’s mental model doesn’t necessarily have to be true or accurate, but it should enable him to work effectively. For example, most nontechnical computer users imagine that their video screen is the heart of their computer. This is only natural because the screen is what they stare at all the time and is the place where they see what the computer is doing. If you point out to a user that the computer is actually a little chip of silicon in that black box sitting under his desk, he will probably shrug and ignore this pointless (to him) bit of information. The fact that the CPU isn’t the same thing as the video display doesn’t help him think about how he interacts with his computer, even though it is a more technically accurate concept.

Most Software Conforms to Implementation Models

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 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).

User interfaces designed by engineers follow the implementation model

User interfaces and interactions designed by engineers, who know precisely how software works, quite often lead to a represented model that is very consistent with its implementation model. To the engineers, such models are logical, truthful, and accurate; unfortunately, they are not very intelligible or effective for users. The majority of users don’t much care how a program is actually implemented.
A good example of a digital product that conforms to implementation models is the typical component home theater system, which requires the user to know exactly how all of the components are wired together in order to switch, say, between viewing a DVD and tuning a cable TV channel. In most products, users need to switch video sources, and sometimes even switch between multiple remote controls, to access the functions they need simply to watch their television. A more mental-model-centered alternative, adopted by some newer products, is to keep track of the component configuration in the remote, so that the user can simply pick “Watch TV,” and the remote sends the appropriate commands to the TV, cable box, DVD player, and surround audio system without the user needing to know what’s going on behind the scenes.

Even the Windows user interface slips into the implementation model sometimes. If you drag a file between directories on the same hard drive, the program interprets this as a MOVE, meaning that the file is removed from the old directory and added to the new directory, closely following the mental model. However, if you drag a file from hard drive C to hard drive D, the action is interpreted as a COPY, meaning that the file is added to the new directory but not removed from the old directory. This behavior is rooted in the implementation model — the way the underlying file system actually works. When the operating system moves a file from one directory to another on the same drive, it merely relocates the file’s entry in the disk’s table of contents. It never actually erases and rewrites the file. But when it moves a file to another physical drive, it must physically copy the data onto the new drive. To conform to the user’s mental model, it should then erase the original even though that contradicts the implementation model.

This inconsistency in the computer’s response to two seemingly similar user actions has the potential to create significant cognitive dissonance (confusion resulting from two contradictory images of reality) for users, which in turn makes this simple interaction difficult to learn. For a user to ultimately be able to achieve a desired result, he must understand that the computer’s behavior depends on the physical nature of the particular storage devices.

Because treating the drag of a file from one disk to another as a COPY function can be desirable behavior, especially when copying files from a hard drive to removable media such as USB flash drives, many people aren’t aware that it is an inconsistent side effect of the implementation model. As computers mature and logical volumes represent more than just physical drives, the side effects stop being useful and instead become irritating because we have to memorize the idiosyncratic behavior of each volume type.
Mathematical thinking leads to implementation model interfaces

Interaction designers must shield users from implementation models. Just because a technique is well suited to solving a problem in software construction doesn’t necessarily mean that it is well suited to be a mental model for the user. Just because your car is constructed of welded metal parts doesn’t mean that you must be skilled with a welding torch to drive it.

Most of the data structures and algorithms used to represent and manipulate information in software are logic tools based on mathematical algorithms. All programmers are fluent in these algorithms, including such things as recursion, hierarchical data structures, and multithreading. The problem arises when the user interface attempts to accurately represent the concepts of recursion, hierarchical data, or multithreading.

Mathematical thinking is an implementation model trap that is particularly easy for programmers to fall into. They solve programming problems by thinking mathematically, so they naturally see these mathematical models as appropriate terms for inventing user interfaces. Nothing could be further from the truth.

Users don’t understand Boolean logic.

For example, one of the most durable and useful tools in the programmer’s toolbox is Boolean algebra. It is a compact mathematical system that conveniently describes the behavior of the strictly on-or-off universe inside all digital computers. There are only two main operations: AND and OR. The problem is that the English language has an and and an or and they are usually interpreted — by nonprogrammers — as the exact opposite of the Boolean AND and OR. If the program expresses itself with Boolean notation, the user can be expected to misinterpret it.

For example, this problem crops up frequently when querying databases. If we want to extract from a file of employees those who live in Arizona along with those who live in Texas, we would say to a human in English, “Find all my employees in Arizona and Texas.” To express this properly to a database in Boolean algebraic terms, we would say, “Find employees in Arizona OR Texas.” No employee lives in two states at once, so saying, “Find employees in Arizona AND Texas” is nonsensical. In Boolean, this will almost always return nothing.
Any application that interacts with users in Boolean is doomed to suffer severe user-interface problems. It is unreasonable to expect users to penetrate the confusion. They are well trained in English, so why should they have to express things in an unfamiliar language that, annoyingly, redefines key words.

**Mechanical-Age versus Information-Age Represented Models**

We are experiencing an incredible transformation from the age of industrial, mechanical artifacts to an age of digital, information objects. The change has only begun, and the pace is accelerating rapidly. The upheaval that society underwent as a result of industrialization will likely be dwarfed by that associated with the Information Age.

**Mechanical-Age representations**

It is only natural for us to try to draw the imagery and language of an earlier era that we are comfortable with into a new, less certain one. As the history of the industrial revolution shows, the fruits of new technology can often only be expressed at first with the language of an earlier technology. For example, we called railroad engines *iron horses* and automobiles were labeled *horseless carriages*. Unfortunately, this imagery and language colors our thinking more than we might admit.

Naturally, we tend to use old representations in our new environments. Sometimes, the usage is valid because the function is identical, even if the underpinning technology is different. For example, when we translate the process of typewriting with a typewriter into word processing on a computer, we are using a Mechanical-Age representation of a common task. Typewriters used little metal tabs to slew the carriage rapidly over several spaces until it came to rest on a particular column. The process, as a natural outgrowth of the technology, was called tabbing or setting tabs. Word processors also have tabs because their function is the same; whether you are working on paper rolled around a platen or on images on a video screen, you need to rapidly slew to a particular margin offset.

Sometimes, however, Mechanical-Age representations shouldn’t be translated verbatim into the digital world. We don’t use reins to steer our cars, or a tiller, although both of these were tried in the early days of autos. It took many years to develop a steering idiom that was appropriate for the car. In word processors, we don’t need to load a new blank page after we fill the previous one; rather, the document scrolls continuously, with visual markers for page breaks.
New technology demands new representations

Sometimes tasks, processes, concepts, and even goals arise solely because new technology makes them possible for the first time. With no reason to exist beforehand, they were not conceived of in advance. When the telephone was first invented, for example, it was, among other things, touted as a means to broadcast music and news, although it was personal communication that became the most popular and widely developed. Nobody at the time would ever have conceived of the telephone as being a ubiquitous personal object that people would carry in their pockets and purses and that would ring annoyingly in the midst of theater performances.

With our Mechanical-Age mindset, we have a hard time seeing appropriate Information-Age representations — at first. The real advantages of the software products that we create often remain invisible until they have a sizable population of users. For example, the real advantage of e-mail isn’t simply that it’s faster than postal mail — the Mechanical-Age view — but rather that it promotes the flattening and democratization of the modern business organization — the Information-Age advantage. The real advantage of the Web isn’t cheaper and more efficient communication and distribution — the Mechanical-Age view. Instead, it is the creation of virtual communities — the Information-Age advantage that was revealed only after it materialized in our grasp. Because we have a hard time seeing how digital products will be used, we tend to rely too much on representations from the past, Mechanical Age.

Mechanical-Age representations degrade user interaction

We encounter a problem when we bring our familiar Mechanical-Age representations over to the computer. Simply put, Mechanical-Age processes and representations tend to degrade user interactions in Information-Age products. Mechanical procedures are easier to perform by hand than they are with computers. For example, typing an individual address on an envelope using a computer requires significant overhead compared to addressing the envelope with pen and ink (although the former might look neater). The situation improves only if the process is automated for a large number of instances in batch — 500 envelopes that you need to address.

As another example, take a contact list on a computer. If it is faithfully rendered onscreen like a little bound book, it will be much more complex, inconvenient, and difficult to use than the physical address book. The physical address book, for example, stores names in alphabetical order by last name. But what if you want to find someone by her first name? The Mechanical-Age artifact doesn’t help you: You
have to scan the pages manually. So, too, does the faithfully replicated digital version: It can’t search by first name either. The difference is that, on the computer screen, you lose many subtle visual cues offered by the paper-based book (bent page corners, penciled-in notes). Meanwhile, the scrollbars and dialog boxes are harder to use, harder to visualize, and harder to understand than simply flipping pages.

Don’t replicate Mechanical-Age artifacts in user interfaces without Information-Age enhancements.

Real-world mechanical systems have the strengths and weaknesses of their medium, such as pen and paper. Software has a completely different set of strengths and weaknesses, yet when mechanical representations are replicated without change, they combine the weaknesses of the old with the weaknesses of the new. In our address book example, the computer could easily search for an entry by first name, but, by storing the names in exactly the same way as the mechanical artifact, we deprive ourselves of new ways of searching. We limit ourselves in terms of capabilities possible in an information medium, without reaping any of the benefits of the original mechanical medium.

When designers rely on Mechanical-Age representations to guide them, they are blinded to the far greater potential of the computer to provide sophisticated information management in a better, albeit different, way. The use of a Mechanical-Age representation in a user interface can operate as a metaphor that artificially constrains the design. For further discussion of the pitfalls surrounding reliance on metaphors in user interfaces, see Chapter 13.

Improving on Mechanical-Age representations: An example

Although new technologies can bring about entirely new concepts, they can also extend and build upon old concepts, allowing designers to take advantage of the power of the new technology on behalf of users through updated representations of their interface.

For example, take the calendar. In the nondigital world, calendars are made of paper and are usually divided up into a one-month-per-page format. This is a reasonable compromise based on the size of paper, file folders, briefcases, and desk drawers.
Programs with visual representations of calendars are quite common, and they almost always display one month at a time. Even if they can show more than one month, as Outlook does, they almost always display days in discrete one-month chunks. Why?

Paper calendars show a single month because they are limited by the size of the paper, and a month is a convenient breaking point. Computer screens are not so constrained, but most designers copy the Mechanical-Age artifact faithfully (see Figure 2-3). On a computer, the calendar could easily be a continuously scrolling sequence of days, weeks, or months as shown in Figure 2-4. Scheduling something from August 28 to September 4 would be simple if weeks were contiguous instead of broken up by the arbitrary monthly division.

Similarly, the grid pattern in digital calendars is almost always of a fixed size. Why couldn’t the width of columns of days or the height of rows of weeks be adjustable like a spreadsheet? Certainly you’d want to adjust the sizes of your weekends to reflect their relative importance in relation to your weekdays. If you’re a businessperson, your working-week calendar would demand more space than a vacation week. The idioms are as well known as spreadsheets — that is to say, universal — but the Mechanical-Age representations are so firmly entrenched that we rarely see software publishers deviate from it.

Figure 2-3  The ubiquitous calendar is so familiar that we rarely stop to apply Information-Age sensibilities to its design on the screen. Calendars were originally designed to fit on stacked sheets of paper, not interactive digital displays. How would you redesign it? What aspects of the calendar are artifacts of its old, Mechanical-Age platform?
Scrolling is a very familiar idiom to computer users. Why not replace the page-oriented representation of a calendar with a scrolling representation to make it better? This perpetual calendar can do everything the old one can, and it also solves the mechanical-representation problem of scheduling across monthly boundaries. Don’t drag old limitations onto new platforms out of habit. What other improvements can you think of?

The designer of the software in Figure 2-3 probably thought of calendars as canonical objects that couldn’t be altered from the familiar. Surprisingly, most time-management software handles time internally — in its implementation model — as a continuum, and only renders it as discrete months in its user interface — its represented model!

Some might counter that the one-month-per-page calendar is better because it is easily recognizable and familiar to users. However, the new model is not that different from the old model, except that it permits the users to easily do something they couldn’t do easily before — schedule across monthly boundaries. People don’t find it difficult to adapt to newer, more useful representations of familiar systems.

**Figure 2-4**  Scrolling is a very familiar idiom to computer users. Why not replace the page-oriented representation of a calendar with a scrolling representation to make it better? This perpetual calendar can do everything the old one can, and it also solves the mechanical-representation problem of scheduling across monthly boundaries. Don’t drag old limitations onto new platforms out of habit. What other improvements can you think of?

The designer of the software in Figure 2-3 probably thought of calendars as canonical objects that couldn’t be altered from the familiar. Surprisingly, most time-management software handles time internally — in its implementation model — as a continuum, and only renders it as discrete months in its user interface — its represented model!

Some might counter that the one-month-per-page calendar is better because it is easily recognizable and familiar to users. However, the new model is not that different from the old model, except that it permits the users to easily do something they couldn’t do easily before — schedule across monthly boundaries. People don’t find it difficult to adapt to newer, more useful representations of familiar systems.

**Significant change must be significantly better.**
Paper-style calendars in personal information managers (PIMs) and schedulers are mute testimony to how our language influences our designs. If we depend on words from the Mechanical Age, we will build software from the Mechanical Age. Better software is based on Information-Age thinking.
Beginners, Experts, and Intermediates

Most computer users know all too well that buying a new cell phone or opening the shrink-wrap on a new software product augurs several days of frustration and disappointment spent learning the new interface. On the other hand, many experienced users of a digital product may find themselves continually frustrated because that product always treats them like rank beginners. It seems impossible to find the right balance between catering to the needs of the first-timer and the needs of the expert.

One of the eternal conundrums of interaction and interface design is how to address the needs of both beginning users and expert users with a single, coherent interface. Some programmers and designers choose to abandon this idea completely, choosing instead to segregate the user experiences by creating wizards for beginners and burying critical functionality for experts deep in menus. Of course, no one wants to deal with the extra labor associated with moving through a wizard, but the leap from there to knowing what esoteric command to select from a series of long menus is usually a jump off a rather tall cliff into a shark-infested moat of implementation-model design. What, then, is the answer? The solution to this predicament lies in a different understanding of the way users master new concepts and tasks.