

Contents

How to Read This Book · 1

PART I: MOTIVATIONS

1 Why I Wrote This Book · 9

2 Discovering Concepts · 15

3 How Concepts Help · 29

PART II: ESSENTIALS

4 Concept Structure · 47

5 Concept Purposes · 59

6 Concept Composition · 79

7 Concept Dependence · 99

8 Concept Mapping · 109

PART III: PRINCIPLES

9 Concept Specificity · 127

10 Concept Familiarity · 147

11 Concept Integrity · 157

Questions to Remember · 167

Acknowledgments · 179

RESOURCES

Explorations & Digressions · 183

References · 299

Index of Applications · 309

Index of Concepts · 311

Index of Names · 315

Index of Topics · 317

1

Why I Wrote This Book

As an undergraduate in physics, I'd been entranced by the idea that the world could be captured by simple equations like $F = ma$. When I became a programmer, and later a computer science researcher, I gravitated towards the field of formal methods, because it promised to do something similar for software: to express its very essence in a succinct logic.

A Passion for Design

My main research contribution in the 30 years since my PhD has been Alloy,³ a language for describing software designs and analyzing them automatically. It's been an exciting and satisfying journey for me, but I came to realize over time that the essence of software doesn't lie in any logic or analysis. What really fascinated me wasn't the question that consumed most formal methods researchers—namely how to check that a program's behavior conforms exactly to its specification—but rather the question of *design*.⁴

I mean “design” here in the same sense that the word is used in other design disciplines: the shaping of some artifact to meet a human need. Design, as the architect Christopher Alexander put it, is about creating a *form* to fit a *context*. For software, that means determining what the behavior of the software should be: what controls it will offer, and what responses it will provide in return. These questions have no right or wrong answers, only better or worse ones.⁵

I wanted to know why some software products seem so natural and elegant, react predictably once you master the basics, and let you combine their features in powerful ways. And to pinpoint why other products just seem wrong: cluttered with needless complexity, and behaving in unexpected and inconsistent ways. Surely, I thought, there must be some essential principles, some theory of software design, that could explain all of this. It would not only explain why some software products are good and some are bad, but it would help you fix the problems and avoid them in the first place.

Design in Computer Science and Other Fields

I started to look around. Within my own subfield (formal methods, software engineering and programming languages), such a theory exists for what you might call “internal design”—namely the design of the structure of the code. Programmers have a rich language of design, and well-established criteria for what distinguishes good designs from bad ones. But no such language or criteria exist for software design in the user-facing sense, namely design that determines how software is experienced as a form in context.⁶

Internal code design is very important and influences primarily what software engineers call “maintainability,” which means how easy (or hard) the code is to change over time as needs evolve. It also influences performance and reliability. But the key decisions that determine whether a software application or system is useful and fulfills its users’ needs lie elsewhere, in the kind of software design in which the functionality and the patterns of interaction with the user are shaped.

These big questions were at one time more central in computer science. In the field of software engineering, they came up in workshops on software design, specification and requirements; in the field of human-computer interaction, they permeated early work on graphical user interfaces and computational models of user behavior.⁷

But as time passed, they became less fashionable, and they faded away. Research in software engineering narrowed, and eliminating defects—whether by testing or more sophisticated means such as program verification—became synonymous with software quality.⁸ But you can’t get there from here: if your software has the wrong design, there’s no amount of defect elimination that will fix it, short of going back to the very start and fixing the design itself.⁹

Research in human-computer interaction (HCI) shifted to novel interaction technologies, to tools and frameworks, to niche domains, and to other disciplines (such as ethnography and sociology). Both software engineering and HCI embraced empiricism enthusiastically, largely in the misguided hope that this would bring respectability. Instead, the demand for concrete measures of success seems to have led researchers towards less ambitious projects that admit easier evaluation, and has stymied progress on bigger and more important questions.¹⁰

1: WHY I WROTE THIS BOOK

Puzzlingly, even as interest in design seems to have waned, talk of “design” is everywhere. This is not in fact a contradiction. The talk, almost exclusively, is about the *process* of design, whether in the context of “design thinking” (a compelling packaging of iterative design processes), or of “agile” software development. These processes are undoubtedly valuable (so long as they are applied judiciously and not as panaceas), but they are for the most part content-free. I mean that not to disparage but to describe. Design thinking, for example, might tell you to develop your solution hand in hand with your understanding of the problem, or to engage in alternating phases of brainstorming (“divergence”) and reduction (“convergence”). But no design thinking book that I have read talks in depth about any particular designs and how the process sheds light on them. The very domain-independence of design thinking may be the key to its widespread appeal and applicability—but also the reason it has little to say about deeper challenges of design in a particular domain such as software.¹¹

Clarity & Simplicity in Design

When I began the Alloy project, with the goal of creating a design language that was amenable to automatic analysis, I was critical of existing modeling and specification languages whose lack of tool support rendered them “write-only.” This snide dismissal was not entirely unwarranted. After all, why would you go to the trouble of constructing an elaborate design model if you couldn’t then do anything with it? I argued, in particular, that the designer’s effort should be rewarded immediately with “push-button automation” that would instantly give you feedback in the form of surprising scenarios that would challenge you to think more deeply about your design.¹²

I don’t think I was wrong, and Alloy’s automation did indeed change the experience of design modeling. But I had underestimated the value of writing down a design. In fact, it was a not very well guarded secret amongst formal methods researchers (who were eager to demonstrate the efficacy of their tools by finding flaws in existing designs) that a high proportion of the flaws were detected *before* the tools were even run! Just transcribing the design into logic was enough to reveal serious problems. The software engineering researcher Michael Jackson credits not the logic per se but the very difficulty of using it, and once mischievously suggested that the quality of software systems might be improved if designers were simply required to record their designs in Latin.

THE ESSENCE OF SOFTWARE

Clarity is good not only for finding design flaws after the fact. It is also the key to good design in the first place. In teaching programming and software engineering over the last thirty years, I've become increasingly convinced that the determinant of success when you're developing software isn't whether you use the latest programming languages and tools, or the management process you follow (agile or otherwise), or even how you structure the code. It's simply whether you know what you are trying to do. If your goals are clear, and your design is clear—and it's clear how your design meets the goals—your code will tend to be clear too. And if something isn't working, it will be clear how to fix it.¹³

It is this clarity that distinguishes great software from the rest. When the Apple Macintosh came out in 1984, people could see immediately how to use folders to organize their files; the complexities of previous operating systems (such as Unix, which made even the command to move files between folders complicated) seemed to have evaporated.

But what exactly is this clarity, and how is it achieved? As early as the 1960s, the central role of “conceptual models” has been recognized. The challenge was not merely to *convey* the software's conceptual model to the user so that her internal version (“mental model”) was aligned with the programmers', but to treat it as a subject of design in its own right. With the right conceptual model, the software would be easy to understand and thus easy to use. This was a great idea, but nobody seems to have pursued it, and so until now “concepts” have remained a vague, if inspiring, notion.¹⁴

How This Project Came About

Convinced that conceptual models were indeed the essence of software, I started about eight years ago trying to figure out what they might be. I wanted to give them concrete expression, so that I could point to some software's conceptual model, compare it to others (and to the mental models of users), and have an explicit focus for design discussions.

That didn't seem so hard. After all, a plausible first cut at a conceptual model might be just a description of the software's behavior, made suitably abstract to remove incidental and “non-conceptual” aspects (such as the details of the physical user interface). What proved much harder was finding appropriate

1: WHY I WROTE THIS BOOK

structure in the model. I had an inkling that a conceptual model should be made up of concepts, but I didn't know what a concept was.

In a social media app such as Facebook, for example, it seemed to me that there should be a concept associated with liking things. This concept surely wasn't a function or action (such as the behavior bound to the button you click to like a post); there are too many of those, and they only tell part of the story. It also surely wasn't an object or entity (such as the "like" itself that your action produced), since at the very least the concept seemed to be about the *relationship* between things and their likes. It also seemed essential to me that the concept of liking was not associated with any particular kind of thing: you could like posts, comments, pages, and so on. The concept, in programming lingo, is "generic" or "polymorphic."

This Book: Opening a Conversation

This book is the result of my explorations to date. Driven by dozens of design issues in widely used applications, I've evolved a new approach to software design, refining and testing it along the way. A happy aspect of this project has been that every app failure or frustration had a silver lining: a chance to extend my repertoire of examples. It has also given me greater sympathy and respect for the designers when my analysis revealed the full complexity of the problem they faced.

Of course, the problem of software design is not solved. But as my friend Kirsten Olson wisely advised me: a book should aim to start a conversation, not to end one. In the course of giving many talks about this project, I've been thrilled to discover that it seems to resonate with audiences more than any of my previous ones. I suspect this is because software design is something we all want to talk about, but we have not known how to have that conversation.

So to you, my readers—fellow researchers, designers and users—I present this book as my opening gambit in what I hope to be a fruitful and enjoyable conversation.

Index of Applications

- Adobe Illustrator 53
- Adobe InDesign 31, 51, 53, 121–122, 161–162, 244, 265–266, 296
- Adobe Lightroom 95, 120–121, 138, 151–153, 247, 271–272, 281–282, 282, 290, 293
- Adobe Photoshop 33, 34, 68, 90, 211, 250, 267–269, 274, 286–288, 290
- Amazon Prime 113
- Apple Calendar 92–94
- Apple Color Picker 54, 236
- Apple Contacts 153–154, 294
- Apple Finder 243, 282
- Apple HyperCard 290
- Apple iCloud 40
- Apple iPhone 215
- Apple iPod 212, 214
- Apple iTunes 214–215
- Apple Keynote 73, 106, 129, 149–151, 209, 210, 274, 278
- Apple Lisa 33, 47
- Apple Macintosh 12, 24, 33, 47, 90, 188, 204, 206
- Apple Mail 50, 60, 118–119, 123, 128, 131, 279
- Apple Numbers 74–76
- Apple Pages 51, 53, 121–122, 161–162, 295–296
- Apple Photos 40
- Apple Podcasts 88
- Apple Preview 210
- Apple Safari 105, 278
- Apple TextEdit 160–162, 224, 296
- Apple Trash 47–51, 91–92, 212, 220, 243, 264, 270
- Atom 31
- Backblaze 16–17, 111–112, 116–118, 123, 280
- BBEdit 31, 291
- Bravo 237
- Calendly 34
- Carbonite 117
- change.org 112
- Chip and PIN 70, 258
- Crashplan 117
- CSS 256–257
- Dropbox 17–23, 40, 50, 203–205, 205–206
- Drupal 214, 289
- Emacs 31
- Epson (printer) 94, 137, 286
- Facebook 13, 29, 69, 104, 133–139, 141–143, 147, 148, 238, 251, 277, 288
- Fujifilm (camera) 138–139, 208
- Git 134, 204, 254
- Gitless 254
- Gmail 37, 50, 89, 93, 114–116, 123, 128, 130, 132, 136, 136–139, 143, 213, 243, 266, 271, 279–280, 283
- Google Apps 25, 93, 164, 271
- Google Calendar 92–94
- Google Docs 60, 61, 237
- Google Drive 50, 162–164, 296, 297
- Google Forms 95, 272
- Google Groups 94–96, 148
- Google Hangouts 73, 198
- Google Sheets 95
- Google Slides 274
- Hacker News 288
- Instagram 147
- LaTeX 32
- LinkedIn 41
- Linux 50
- Mastermatic 266

THE ESSENCE OF SOFTWARE

- Mercurial 134
Microsoft Outlook 270–271
Microsoft PowerPoint 53, 129, 149–151, 209,
274, 291–292, 292
Microsoft Publisher 31
Microsoft Teams 73
Microsoft Windows 24, 90, 208, 237, 270
Microsoft Word 31, 51, 53, 121–122, 211, 237,
243, 247, 265
Moirā 89, 266
Multics 205

Netflix 142, 273
NokNok 294

Open Office 31
OpenTable 251, 265
Oracle Java 109–111
OS X (macOS) 208
 El Capitan 50
 Lion 51, 64
 Mountain Lion 255

PayPal 114
Piazza 101
PostScript 160

QuarkXPress 31, 53
Quora 70, 101

Reddit 288

Scribus 31
Signal 142
Skype 198
Slack 142, 148
SnapChat 147
Squarespace 129
StackExchange 101
Sublime 31
Subversion 134

Teabox 267
Therac-25 96, 273
TikTok 274
Todoist 81
Tumblr 93
Twitter 29, 65, 93, 134, 147, 148, 255–256,
290

Unix 12, 203–205

VisiCalc 34, 289

WhatsApp 147, 148, 274
WordPerfect 31
WordPress 289

Zoom 73, 96, 131, 197, 272–273, 283
Zotero 120

Index of Concepts

- access control 86, 261
- account 239
- action 153, 290
- adjustment 240
- administrative group 89
- adverse interaction 216
- animation 106, 278, 279
- article 174
- aspect ratio 138, 139
- asset 239, 240
- audio mute 96
- auditing 41
- authentication 41, 239, 241
- authorization 41
- auto caption 274
- available funds 66

- backup 16, 17, 111, 128
- bank 239
- battery 73, 260
- BCC 136
- bookmark 61, 66, 105, 134, 278
- boxing 254
- branch 254
- breakout 283
- broadcast 131, 283

- cache 105
- calendar event 92, 272
- call 15, 274
- call forwarding 62, 87
- capability 41
- category 130, 170
- cellular 88
- certificate 35, 105
- channel 90, 148, 211, 240, 267–270
- chapter 31
- character 31
- character style 278
- chat 131, 283

- chatroom 148
- chip 70
- class (CSS) 53
- cleared check 67
- cloud app 164
- collateral 239
- collection 120
- color theme 53
- comment 29, 93, 104, 148, 174, 246, 246–247, 276–277, 277
- commit 135
- contact 87, 153, 154
- conversation 30, 114, 279
- cookie 105, 278
- correspondent 128, 169
- coupon 267
- cropping 170, 286, 288
- cursor 291

- delegate forwarding 63
- direct flight 71
- directory 203
- domain name 87, 210
- donation 113
- do not disturb 112
- dose 42, 216

- editor buffer 64, 254
- email 30, 85, 86, 176, 261, 263, 295
- email address 93
- equity 239
- event type 34
- existence coupling 84

- favorite 65–66, 105, 134, 148, 255, 278
- FDIC insurance 239
- filter 132, 281–282
- flag 118, 281, 283, 284
- folder 15, 30, 90–92, 120, 172, 176, 177, 203, 204, 243, 270, 271, 279
- follower 134, 148

THE ESSENCE OF SOFTWARE

- follow-me forwarding 63
- format 32, 121, 211, 225, 244
- format toggle 159–162, 164, 296
- formula 34
- free sample 89
- frequent flyer 36, 114, 212
- frequently visited 105, 278
- friend 30, 86, 104, 133, 134, 148, 277

- group 38, 44, 148, 237–238
- group directory 94

- hashtag 290
- holding 240
- HTML 34
- HTTP 87
- hypertext 290

- identification 101–104, 274, 277
- image quality 138
- image size 67–69, 138, 139, 256–257
- index 254
- install 110
- interbank transfer 239
- invitation 38, 92

- karma 288
- keyword 282

- label 30, 37, 81–85, 85–88, 88–90, 99, 114–116, 130, 168, 176, 213, 262, 263, 266, 279–280, 281
- layer 33, 34, 106, 167, 211, 240, 250, 268, 274, 290
- like 13, 29, 66, 105, 141–143, 170, 278, 288
- line 31
- link 34
- loan 239
- log 270

- mailbox 30
- mailing list 89
- markup 290
- mask 33, 34, 90, 211, 268, 290
- master 266

- master page 265
- master slide 106, 274
- message 148
- metadata 21, 241
- moderation 38, 39, 148

- nickname 153, 154, 294
- no-show 56
- notification 38, 69, 87, 97, 148, 158, 171, 243, 245

- object list view 209
- outline tree 151, 279

- page 32
- page cache 35
- paper feed 137, 286
- paper option 94
- paper size 137, 286
- paper source 94
- paragraph 31, 32, 106, 211, 225, 244
- paragraph style 278
- password 177, 241
- permission 94
- petition 112–114, 113
- phone call 87
- photo 277
- PIN 70
- pixel 240
- pixel array 67, 68, 267, 269
- podcast 88
- poke 63, 251
- post 29, 38, 86, 87, 93, 104, 105, 148, 167, 174, 232, 238, 243, 246, 247, 276, 277
- precis 135
- prescription 216
- preset 138, 151–153, 293–294
- preset station 253
- private browsing 35, 105, 278
- profile 238, 273
- profiling 142, 143
- publication 240
- push poll 71

- Q&A 101–102, 102–104, 275

INDEX OF CONCEPTS

- raised hand 96
- range 74, 74–76, 172
- rating 148, 233
- reaction 141–143, 278
- reading list 106, 278
- recent activity 148
- recommendation 142, 143, 288
- recording 101, 103, 104, 277
- reference 34
- reply 104, 277, 278
- request 38
- resampling 288
- reservation 15, 55–57, 58, 72, 158, 176, 201, 220, 223, 227, 229, 230, 231, 233, 234, 236, 237, 238, 239, 243, 245, 251, 263, 264, 265
- review 158, 265, 285
- rule 132

- save as 255
- seat 36
- seat allocation 229
- section 31, 60, 61, 150, 171, 212, 291
- selection 90, 268, 282, 291
- selection pane 209
- shape 106
- shape style 278
- shared song 274
- sharing 21
- shopping cart 89
- slide 106
- slide group 149–151
- slideshow 73
- social security number 237
- song 36, 212
- special block 106, 278, 279
- staging area 254
- star 283, 284
- stash 254
- status update 238
- stock 239
- style 32, 51–54, 106, 121, 129, 211, 220, 223, 224, 225, 229, 235, 236, 237, 238, 240, 241, 243, 244, 245, 253, 263, 265, 266, 274
- stylesheet 106
- style (TextEdit) 54
- subject line 135, 136
- subscription 148
- swatch (Illustrator) 53, 54
- synchronization 40, 163, 164, 214–215, 297

- table 176
- tag 69, 102, 104, 277
- target 73
- template 129, 140
- text block 106
- text flow 32
- text style 106
- theme 106, 129
- this object 255
- threaded comment 278
- timeline 238
- title 93
- todo 81, 82, 83, 84, 85, 86, 88, 99, 234, 262
- transition 278
- trash 33, 34, 47–51, 61, 74, 89, 90–92, 172, 176, 177, 205, 219–220, 220–221, 223, 225, 228, 229, 230, 237, 238, 240, 243, 245, 262, 295, 297
- tweet 93
- two-factor authentication 41, 215
- typeface 159, 160, 161

- Unix folder 22, 270
- unread 87
- upvote 15, 39, 66, 87, 101, 102, 103, 104, 109, 141–143, 148, 241, 277, 278, 288, 289
- URL 34, 105, 167, 232, 290
- user 70, 101, 102, 103, 104, 105, 273, 275
- username 93

- VIP 60

- web service 210, 211
- Wi-Fi 88

- Zoom session 272

© Copyright Princeton University Press. No part of this book may be distributed, posted, or reproduced in any form by digital or mechanical means without prior written permission of the publisher.

For general queries contact webmaster@press.princeton.edu.

Index of Names

- Adams, President John 196
Aldrich, Jonathan 193
Alexander, Christopher 9, 194, 195, 258, 291
- Bachman, Charles 240
Batory, Don 247
Berners-Lee, Tim 290
Bjorner, Dines 203
Bossavit, Laurent 193
Bricklin, Dan 34, 289
Brignull, Harry 279
Bringham, Robert 195
Brooks, Fred 100, 199–200
Bruns, Glen 248
Buxton, Bill 192, 207
- Card, Stuart 188, 198, 199
Charles, Prince of Wales 153, 294
Chen, Peter 200
Codd, Edgar 289
Constantine, Larry 258
Cunha, Alcino 230
- Dijkstra, Edsger 197, 213, 264
Dorsey, Jack 41
- Eames, Charles 183
Eames, Ray 183
Egan, Kieran 286
Elizabeth II, Queen 153, 294
Evans, Eric 202
- Faste, Rolf 249
Floyd, Bob 189
Foley, James 206
Fowler, Martin 201
Fu, Kevin 215
- Greenberg, Saul 192
Green, Thomas 188, 207
Gunter, Carl 249
Gunter, Elsa 249
- Hoare, Tony 189, 197, 263
Hollan, Jim 252
Hutchins, Ed 252
- Ingram, Matthew 255
Isaacson, Walter 212
Ive, Jony 183
- Jackson, Michael 11, 195, 202, 213, 222, 249, 250, 259
Jackson, Tim 219
Jacobson, Ivar 224
Jen, Natasha 195
Jobs, Steve 36, 183, 212
- Kaashoek, Frans 203
Kapor, Mitchell 184, 185, 186
Kleer, Johan de 294
Ko, Amy 211
Koppel, Jimmy 275
- Lampson, Butler 203, 237
Latour, Bruno 264
Leveson, Nancy 273
Levy, Mattys 195
Liddle, David 186
Linnaeus, Carl 236
- Madrigal, Alexis 237
McIlroy, Doug 196
McKim, Robert 249
Menzies, Tim 187
Mitnick, Kevin 215
Molich, Rolf 188
Moran, Tom 188, 198, 199, 206
Mott, Tim 237
Mylopoulos, John 200
- Newell, Allen 188
Newton, Casey 255
Newton, Isaac 222
Nielsen, Jakob 188

THE ESSENCE OF SOFTWARE

- Norman, Don 26, 43, 188, 198, 216, 219, 251, 252, 293
- Olson, Kirsten 13
- Ostroy, Andy 65
- Ovide, Shira 197
- Parnas, David 192, 247, 274, 276
- Perez De Rosso, Santiago 214, 254
- Perrow, Charles 195
- Petroski, Henry 195
- Polanyi, Michael 195, 222
- Pomiane, Edouard de 1, 5, 183
- Rose, David 208
- Rumsfeld, Donald 259
- Saca, Chris 255
- Saltzer, Jerry 203
- Seely Brown, John 294
- Seward, Don 258
- Shneiderman, Ben 216
- Simon, Herb 258
- Simonyi, Charles 237
- Spang, Rebecca 237
- Suh, Nam 195
- Tesler, Larry 237
- Thimbleby, Harold 189, 197, 216, 223
- Tognazzini, Bruce 188
- Tschichold, Jan 195
- Turner, Clark 273
- Uhlich, Karl 284
- van Dam, Andries 206
- van Lamsweerde, Axel 250
- Victor, Bret 208
- Vincenti, Walter 289
- von Hippel, Eric 290
- Wheeler, David 241
- Whitney, Eli 196
- Winograd, Terry 184
- Yu, Eric 258
- Zave, Pamela 249, 251
- Zhang, Kelly 205

Index of Topics

- abstraction 25–26
- access control 86
- accessibility 24
- action 49
 - ambiguous 120
 - formalized 227
 - inferred 87
 - not in logic 231
 - overcomplicated 175
- affordance 188, 293
- Afghanistan 72
- agile development 11, 209, 224
- airlines 36, 113, 212, 236
- Alloy (language) 9, 184
 - automation and, 11
 - concepts and, 225
 - exploration and, 197
 - operational principle in, 230
- analytics 86
- anthropomorphism 264
- assurance case 259
- auditing 86
- authentication 35
- backup
 - cloud storage and, 40
 - confusing 16–17, 111–112
 - deletion and, 128
 - Google Drive 297
 - restore 116–118, 280
 - version control and, 134–139
- battery failure 72–73
- beneficent difficulty 250
- B (formal method) 196
- bird song app 100–101
- bloat, in applications 209
- bounded context 202
- Bringing Design to Software (book) 185
- browser 35, 105
- bugs, and verification 189
- call forwarding 62–63, 87, 251, 295
- cognitive dimensions of notations 188, 207
- commit graph 134–139
- compatibility 241
- composition 79–97
 - collaborative 85–88
 - conventional 79–80
 - free 81–85
 - preserves concept behavior 261
 - semantics 260–264
 - state-based 269–270
 - synchronization 80
 - synergistic 88–90, 114
 - views and, 269–270
- concept
 - abstract type vs. 246–247
 - actions 49, 175, 227
 - age 167
 - API design and, 178
 - basis of design critique 42
 - behavior 57, 240–241
 - catalog 173, 175
 - competitors' 169
 - complicated 145
 - composite 176
 - composition 79–97, 99
 - conformity 153–154
 - consistent subset 103
 - core 35–36, 167
 - costs & benefits 37–38
 - data model 58
 - defines app 29–31
 - defines app family 31–33
 - defines business 35–36
 - dependence 99–107, 102–104, 172, 174, 246, 275–277
 - dependence diagram 102–104
 - deterministic 228
 - dishonest 36
 - evolving 237–248

THE ESSENCE OF SOFTWARE

concept, continued

explanation order 104, 173
familiarity 147–155, 171
feature vs. 247–248
flaw in code 178
freestanding 99–107, 157, 245–246, 275,
294
generic 13, 101–102, 123, 164, 174, 176,
234, 243–245
handbook 39, 173, 175
happy and sad 170
identifying usability snags 40
implementation 174–175
integrity 157–164, 172, 261
inventive 236–238
inventory 36, 100–101
key characteristics 236–248
localizes data model 242–243
localizes design 47
mapping 109–123, 172, 199, 281–282
medical 216
mental concept vs. 248
misfit 154
missing 169
modularity 177–178
most valuable 168
name 48, 221
near miss 236
one-to-one with purpose 127–145
operational principle 49, 51, 56, 61, 74,
110, 113, 123, 158, 170, 221–222
overloaded 132–139, 144, 145, 170
product differentiator 33–34, 274
purpose 48, 59–77, 168, 173
purposeless 127, 144, 254
purposive 238–240
redundant 130–132, 144, 145, 170, 283
responds to problem 273–274
reuse 39, 147–151
security and safety 41–42, 215
seed 101
semantics 225–226, 225–231
separating concerns 38

concept, continued

shared across family 168
specificity 127–145, 153
splitting 171
state 48, 52
stateful 241–242
structure 47–58
synchronization 157, 175
synergy 172
tricky 34–35
troubled 168
vocabulary 169
conceptual integrity 199–200
conceptual modeling
field 200–201
origins 198–200
concreteness fading 243
consistency 25
correctness 189
critical systems 215
crowdsourcing 101, 103
CSP (Communicating Sequential
Processes) 263–264
dark pattern 112–114
data model 58
defect elimination
focus of research 10
parable 191–192
software quality and, 190–192
Demeter, Law of 276
dependability case 259
dependence
between concepts 99–107
concept vs. code 275–277
diagram 102–104, 174
emerges from details 277
module vs. concept 245
Parnas definition 274–275
primary vs. secondary 277, 278
significant idea 187
design
axiomatic 195, 285
clarity and, 11–12

INDEX OF TOPICS

- design, continued*
 - critique 42
 - democratization of 194
 - domain-driven 202–203
 - expert vs. novice 147
 - HCI research 188–189
 - importance of details 183, 195
 - inevitability 292–293
 - internal, of software 10, 186
 - levels 23–26, 206–207
 - meaning of term 9, 185–186
 - normal vs. radical 289–290
 - novelty 195
 - origins of software 184–186
 - other domains 195
 - pattern 194, 195, 291
 - principles 216–219
 - reuse and, 194
 - safety 41–42, 216
 - security 41–42, 215
 - software engineering research 187
 - teams 38
 - top-down 213
 - typography 195
 - user-centered 188
- design journal 173
- Design of Everyday Things (book) 198, 216, 219, 251, 252
- design thinking
 - aided by concepts 193–194
 - domain-independent 11, 193–194
 - qualms about 194–195
- desktop publishing apps 32–33
- determinism 228
- digital transformation 35–36
- discoverability 209
- domain modeling 202–203
- d.school 185, 249
- Electrum (language) 184, 227
- email 30, 132
- empiricism
 - research and, 10, 192–193
 - software design and, 3
- engineering, vs. science 222
- entity-relationship diagram 58
- failures, of software 195
- familiarity, of concepts 147–155, 171
- family, of applications 103, 168, 247–248
- faucet 63–64, 252
- feature
 - concept vs. 247–248
 - diagram 277
 - interaction 248, 294–295
- Fitts’s law 24, 208
- fonts, professional 160
- formal methods
 - automation and, 11
 - benefits of modeling alone 11
 - in general 9
 - model-based 196
 - specification languages 196–197
 - verification 189–190
 - write-only 11
- formal methods, specific
 - Alloy 9, 11, 184, 197, 225, 230, 235
 - B 196
 - Electrum 184, 227
 - Larch 196, 197
 - OBJ 196
 - TLA 231
 - VDM 196, 225
 - Z 184, 196, 225, 235
- generated inputs 262–263
- GenVoca (framework) 247
- goal, vs. purpose 219–220
- gulfs of execution and evaluation 252–254
- heuristic evaluation 43, 188
- human-computer interaction
 - design and, 10–11, 188–189
 - design principles 216–219
 - empiricism and, 10, 217
 - formal methods and, 197
 - mapping and, 109
 - psychology and, 208

THE ESSENCE OF SOFTWARE

- IDEO (company) 249
- i* framework 258
- incremental development 99–100
- indirection 241
- inevitability, in design 292–293
- inscription 264
- invariants, in programming 187
- JSD (Jackson System Development) 202
- Larch specification language 196, 197
- learnability 211
- levels of design 23–26
 - conceptual 25
 - linguistic 24
 - physical 23
- listserv 136
- liveness property 261
- logging 86
- logic
 - dynamic 229
 - first order 235
 - linear temporal 230
 - temporal 250
- mapping
 - live filtering 281–282
 - Norman's usage 43, 188
 - overview 109–123
 - principles 177
 - questions 172
- mental model 12, 26–27, 198, 248, 294
- metaphor
 - inscription vs. 264
 - misleading 173, 220
 - unhelpful 76, 250
- micromaniac 1, 5
- minimum viable product 99, 174
- misfit 71–76
 - from context 267
 - origins 258
 - safety and, 273
 - verification and, 259
- mobile phone 88
- model
 - conceptual 12, 210–211
 - data 221, 242–243
 - domain 242
 - entity-relationship 200, 201
 - localized data 202
 - mental 164, 169, 206, 294
 - role 240
 - semantic data 200
- Mythical Man Month (book) 199
- nail clipper 284
- names, role in design 221
- nanny scam 66–67
- near-miss concept 236
- needfinding 248–249
- no-function-in-structure principle 294
- none, as value 121–122
- Notes on the Synthesis of Form (book) 258–259
- object
 - classification 231–234
 - entity vs. value 202
 - identity 232
 - mutability 232
 - roles 231–234
- object-oriented programming 246–247, 275–277
- OBJ (specification language) 196
- OOram (Object-oriented Role Analysis and Modeling) 240
- operational principle 49
 - anthropomorphism and, 264
 - elaborate 51
 - explanation and, 56
 - formalizing 229–231
 - fulfills purpose 61
 - liveness and, 262
 - mapping and, 123
 - origins 195, 222
 - partial theorem and, 223
 - subtle 74
 - support materials and, 170

INDEX OF TOPICS

- theorem and, 221
- unclear 113
- use case vs. 224–225
- violated 158
- workflows and, 110
- overloading 132–139
 - consequences 144
 - denied purpose 132–133, 134–135
 - emergent purpose 132–133, 135–136
 - false convergence 132–133, 133–139, 286
 - mechanical design 284–285
 - piggybacking 132–133, 137–307
 - social concepts 285–286
 - suggests splitting 145
 - summarized 144
 - synergy and, 271
 - upvote concept 289
- partial theorem 223
- pattern
 - analysis 201
 - dark 212, 279
 - design 194, 195, 291
 - Gang of Four 291
- perceptual fusion 24
- permutation invariance 234
- phishing 35
- pleasantness problem 189
- PLGR (precision lightweight GPS receiver) 72–73, 259–260
- Post-its 195
- precondition 226–227
 - integrity and, 262
 - operator 229
- problem frames 202
- product line 103
- programming language
 - Algol-68 197
 - Fortran 186
 - Java 240, 254, 276
 - JavaScript 255
 - PL/1 197
- Psychology of Human-Computer Interaction (book) 188, 208
- purpose 48, 59–77
 - clarity of 60–61
 - coherence of 145
 - concept without 63–65
 - conflict of 140
 - confusing 66–67
 - criteria for 61–62
 - deceitful 70–71
 - essential 145
 - goal vs. 219–220
 - granularity & coherence 140
 - importance 238–240
 - kinds of 176
 - misleading 69–70
 - reformulation of 140
 - resolves design puzzles 62–63
 - stakeholder for 140
 - unfulfilled due to misfits 71–72
 - without concept 127–129, 144, 145
- radiotherapy 96
- redundant concepts 130–132
- refactoring 225
- refrigerator controls 251
- relational join 235–236
- representation exposure 187
- representation independence 187
- requirements
 - Alexander’s method 258
 - belong to domain 202
 - complete 258
 - negative 72, 258
 - overview 249–250
 - use cases and, 224
- reservation
 - conflicts 236
 - networking 57
 - railway 57
 - restaurant 56, 223, 239, 264
- resolution, of image 67–69
- revengeful restaurateur 158–159
- safety-critical systems 215
- safety property 261

THE ESSENCE OF SOFTWARE

- search 128
- security 41–42, 215
- semantics
 - composition 260–264
 - concept 225–231
 - usability and, 207
- separation of concerns 88, 213–214
- simplicity 40, 197–198
- slideshow 73–74
- SMV (model checker) 184
- software engineering
 - best quotes 197
 - body of knowledge 186
 - components 196
 - defect elimination and, 190
 - design vs. 10–11, 185
 - empiricism and, 10
 - empiricism in, 192
 - folklore beliefs in, 193
 - requirements 249–250
 - research narrowed 10, 187
- specification
 - abstraction and, 25
 - correctness and, 189
 - dependences and, 187
 - design and, 190, 196–197
 - essential knowledge 186
 - feature interaction and, 294
 - integrity and, 261
 - languages 11
 - liveness and, 261
 - misfit and, 259
 - operational principle vs. 50
 - pleasantness and, 189
 - preserving, of concept 157
 - requirement vs. 249
 - safety and, 261
 - top-down and, 213
 - verification and, 189
 - views 269
- spreadsheet 34
- staging 87
- state 48, 52
 - function 228
 - memory and, 221
 - mutable objects and, 233
- state machine 225–226
- style
 - inheritance 54, 223
 - linked to page master 266
 - next 244
 - override 54
 - partial 54, 121–122, 296
 - user-defined 237
- synchronization 80–97
 - actions enough? 177
 - complex 175
 - concept of 40
 - mitigation 87
 - over 92–94, 171, 271
 - removes executions 84–85
 - semantics 260–264
 - staging 87
 - suppression 86
 - under 94–96, 171
- synergy 88–90
 - Apple Trash 90, 177, 270
 - Gmail 89, 114, 266
 - Google Forms 95
 - imperfect 90, 91–92, 267
 - Netflix 273
 - Outlook 270
 - Photoshop 90, 267–269
 - proliferation suggests 106
 - questions 172
 - Safari 278
 - state-based 269–270
 - trade-off example 266
- system image 26
- Terminal World 208
- testing 10, 72
- text messaging 30
- TLA (Temporal Logic of Actions) 231

INDEX OF TOPICS

- toaster 222
- trace 228
- traffic light 259
- training materials 173
- transition relation 226–227
- trepanning 280, 294
- type
 - abstract data 246–247
 - interpreted vs. uninterpreted 233–234
- USB key 50, 74
- use cases and user stories 224–225
- user manual 169, 173, 293
- uses relation 274–275
- VDM (Vienna Development Method)
 - 196, 225
- verification
 - does not prevent misfits 259
 - equated to software quality 10
 - history 189–190
- waterfall process 193
- WIMP (windows, icons, menus, pointer)
 - 33, 253
- word processors 32–33
- World Wide Web 34, 290
- Xerox PARC 33, 198, 237, 289
- Z (specification language) 184, 196, 225