INDEX

A
ABSTRACT CORE, 435–437
ADAPTERS, 367
AGGREGATES
definition, 126–127
elements, 130–135, 170–171, 177–179
invariants, 128–129
local vs. global identity, 127
overview, 129–129
ownership relationships, 126
Agile design
distillation, 483
MODULES, 111
reducing dependencies, 265, 435–437, 463
supple design, 243–244, 260–264
AIDS Memorial Quilt Project, 479
Analysis models, 47–49
Analysis patterns. See also design patterns.
concept integrity, 306–307
definition, 293
elements, 295–306
overview, 294
UBIQUITOUS LANGUAGE, 306–307
ANTICORRUPTION LAYER
ADAPTERS, 367
considerations, 368–369
elements, 369–370
FAÇADES, 366–367
interface design, 366–369
overview, 364–366
relationships with external systems, 384–385
Application layer, 70, 76–79
Architectural frameworks, 70, 74, 156–157, 271–272, 495–496
ASSERTIONS, 255–259
Associations
bidirectional, 102–103
elements, 169–170
for practical design, 82–88
VALUE OBJECTS, 102–103
Astrolabe, 47
Awkwardness, concept analysis, 210–216
B
Bidirectional associations, 102–103
Blind men and the elephant, 377–381
Bookmark anecdote, 57–59
BOUNDARY CONTEXT. See also
CONTEXT MAP.
code reuse, 344
CONTINUOUS INTEGRATION, 341–343
defining, 382
duplicate concepts, 339–340
elements, 337–340
false cognates, 339–340
large-scale structure, 485–488
overview, 335–337
relationships, 352–353
splinters, 339–340
testing boundaries, 351
translation layers, 374. See also ANTICORRUPTION LAYER; PUBLISHED LANGUAGE.
vs. MODULES, 335
Brainstorming, 7–13, 207–216, 219
Breakthroughs, 193–200, 202–203
Business logic, in user interface layer, 77
Business rules, 17, 225

C
Callbacks, 73
Cargo shipping examples. See examples, cargo shipping.
Changing the design. See refactoring.
Chemical warehouse packer example, 235–241
Chemistry example, 377
Cleese, John, 5
Closure of operations, 268–270
Code as documentation, 40
Code reuse
bounded context, 344
generic subdomains, 412–413
reusing prior art, 323–324
Cohesion, modules, 109–110, 113
Cohesive mechanisms
and declarative style, 426–427
example, 425–427
overview, 422–425
vs. generic subdomains, 425
Common language. See published language; ubiquitous language.
Communication, speech. See ubiquitous language.
Communication, written. See documents; UML (Unified Modeling Language); ubiquitous language.
Complexity, reducing. See distillation; large-scale structure; layered architecture; supple design.
Composite pattern, 315–320
Composite specification, 273–282
Concept analysis. See also analysis patterns; examples, concept analysis.
awkwardness, 210–216
contradictions, 216–217
explicit constraints, 220–222
language of the domain experts, 206–207
missing concepts, 207–210
processes as domain objects, 222–223
researching existing resources, 217–219
specification, 223
trial and error, 219
Conceptual contours, 260–264
Conceptual layers, See layered architecture; responsibility layers
Configuring specification, 226–227
Conformist, 361–363, 384–385
Constructors, 141–142, 174–175. See also factories.
Context map. See also bounded context.
example, 346–351
organizing and documenting, 351–352
overview, 344–346
vs. large-scale structure, 446, 485–488
Context map, choosing a strategy anticorruption layer, 384–385
Conformist, 384–385
customer/supplier development teams, 356–360
defining bounded context, 382
deployment, 387
external systems, 383–385
integration, 384–385
merging open host service and published language, 394–396
merging separate ways and shared kernel, 389–391
merging shared kernel and continuous integration, 391–393
packaging, 387
phasing out legacy systems, 393–394
for a project in progress, 388–389
Separate ways, 384–385
Shared kernel, 354–355
specialized terminologies, 386–387
system under design, 385–386
team context, 382
trade-offs, 387
transformations, 389
transforming boundaries, 382–383
Context principle, 328–329. See also BOUNDED CONTEXT; CONTEXT MAP.

CONTINUOUS INTEGRATION, 341–343, 391–393. See also integration.

Continuous learning, 15–16

Contradictions, concept analysis, 216–217

CORE DOMAIN

DOMAIN VISION STATEMENT, 415–416
flagging key elements, 419–420
MECHANISMS, 425
overview, 400–405

Costs of architecture dictated MODULES, 114–115
Coupling MODULES, 109–110
Customer-focused teams, 492
CUSTOMER/SUPPLIER, 356–360

D

Database tuning, example, 102
Declarative design, 270–272
Declarative style of design, 273–282, 426–427
Decoupling from the client, 156
Deep models
distillation, 436–437
overview, 20–21
refactoring, 189–191
Deployment, 387. See also MODULES.
Design changes. See refactoring.
Design patterns. See also analysis patterns.
COMPOSITE, 315–320
FLYWIGHT, 320
overview, 309–310
STRATEGY, 311–314
vs. domain patterns, 309

Development teams. See teams.
Diagrams. See documents; UML (Unified Modeling Language).
Discovery, 191–192
Distillation. See also examples, distillation.
ABSTRACT CORE, 435–437
deep models, 436–437

DOMAIN VISION STATEMENT,
encapsulation, 422–427
HIGHLIGHTED CORE, 417–421
INTENTION-REVEALING INTERFACES,
422–427
large-scale structure, 483, 488–489
overview, 397–399
PCB design anecdote, 7–13
polymorphism, 435–437
refactoring targets, 437
role in design, 329
SEGREGATED CORE, 428–434
separating CORE concepts, 428–434
Distillation, COHESIVE MECHANISMS and declarative style, 426–427
overview, 422–425
vs. GENERIC SUBDOMAINS, 425
Distillation, CORE DOMAIN

DOMAIN VISION STATEMENT, 415–416
flagging key elements, 419–420
MECHANISMS, 425
overview, 400–405

Distillation, GENERIC SUBDOMAINS
adapting a published design, 408
in-house solution, 409–410
off-the-shelf solutions, 407
outsourcing, 408–409
overview, 406
reusability, 412–413
risk management, 413–414
vs. COHESIVE MECHANISMS, 425
Distillation document, 418–419, 420–421

Documents
code as documentation, 40
distillation document, 418–419, 420–421
DOMAIN VISION STATEMENT, 415–416
explanatory models, 41–43
keeping current, 38–40
in project activities, 39–40
purpose of, 37–40
validity of, 38–40
UBIQUITOUS LANGUAGE, 39–40
Domain experts
gathering requirements from. See concept analysis; knowledge crunching.
language of, 206–207. See also UBQUITOUS LANGUAGE.
Domain layer, 70, 75–79
Domain objects, life cycle, 123–124.
See also AGGREGATES;
FACTORIES; REPOSITORIES.
Domain patterns vs. design pattern, 309
DOMAIN VISION STATEMENT, 415–416
Domain-specific language, 272–273
Duplicate concepts, 339–340
Elephant and the blind men, 377–381
Encapsulation. See also FACTORIES.
COHESIVE MECHANISMS, 425–427
INTENTION-REVEALING INTERFACES, 246
REPOSITORIES, 154
ENTITIES. See also associations;
services; VALUE OBJECTS.
automatic IDs, 95–96
clustering. See AGGREGATES.
establishing identity, 90–93
example, 167–168
ID uniqueness, 96
identifying attributes, 94–96
identity tracking, 94–96
modeling, 93–94
referencing with VALUE OBJECTS, 98–99
vs. Java entity beans, 91
Evant, 504–505
EVOLVING ORDER, 444–446, 491
Examples
AGGREGATES, 130–135
analysis patterns, 295–306
ASSERTIONS, 256–259
breakthroughs, 202–203
chemical warehouse packer, 235–241
chemistry, PUBLISHED LANGUAGE, 377
CLOSURE OF OPERATIONS, 269–270
COHESIVE MECHANISMS, 425–427
composite SPECIFICATION, 278–282
CONCEPTUAL CONTOURS, 260–264
constructors, 174–175
Evant, 504–505
explanatory models, 41–43
extracting hidden concepts, 17–20
insurance project, 372–373
integration with other systems, 372–373
INTENTION-REVEALING INTERFACES, 423–424
introducing new features, 181–185
inventory management, 504–505
investment banking, 211–215
KNOWLEDGE LEVEL, 466–474
LAYERED ARCHITECTURE, 71–72
MODEL-DRIVEN DESIGN, 52–57
MODULES, 111–112
multiple teams, 358–360
online banking, 71–72
organization chart, 423–427
package coding in Java, 111–112
paint-mixing application, 247–249, 252–254, 256–259
payroll and pension, 466–474
PLUGGABLE COMPONENT FRAMEWORK, 475–479
procedural languages, 52–57
prototypes, 238–241
PUBLISHED LANGUAGE, 377
purchase order integrity, 130–135
refactoring, 247–249
RESPONSIBILITY LAYERS, 452–460
selecting from Collections, 269–270
SEMATECH CIM framework, 476–479
SIDE-EFFECT-FREE FUNCTIONS, 252–254, 285–286
SPECIFICATION, 235–241
supple design, 247–249
time zones, 410–412
tuning a database, 102
VALUE OBJECTS, 102
Examples, cargo shipping
AGGREGATES, 170–171, 177–179
allocation checking, 181–185
ANTICORRUPTION LAYER, 369–370
FACTORIES (continued)
example, 174–175
invariant logic, 143
overview, 136–139
placing, 139–141
reconstitution, 145–146
and repositories, 157–159
requirements, 139
FACTORY METHOD, 139–141
False cognates, 339–340
Film editing anecdote, 5
Flexibility. See supple design.
FLYWEIGHT pattern, 320
Functions, SIDE-EFFECT-FREE,
250–254, 285–286

G
GENERIC SUBDOMAINS
adapting a published design, 408
easy, 410–412
in-house solution, 409–410
off-the-shelf solutions, 407
outsourcing, 408–409
overview, 406
reusability, 412–413
risk management, 413–414
vs. COHESIVE MECHANISMS, 425
Granularity, 108

H
Hidden concepts, extracting,
17–20
HIGHLIGHTED CORE, 417–421
Holy Grail anecdote, 5

I
Identity
establishing, 90–93
local vs. global, 127
tracking, 94–96
Immutability of VALUE OBJECTS,
100–101
Implicit concepts
categories of, 219–223
recognizing, 206–219
Infrastructure layer, 70
Infrastructure-driven packaging,
112–116

In-house solution, GENERIC SUB-
DOMAINS, 409–410
Insurance project example,
372–373
Integration
ANTICORRUPTION LAYER, 364–370
CONTINUOUS INTEGRATION,
341–343, 391–393
cost/benefit analysis, 371–373
elephant and the blind men,
377–381
easy, 372–373
external systems, 384–385
OPEN HOST SERVICE, 374
SEPARATE WAYS, 371–373
translation layers, 374. See also
PUBLISHED LANGUAGE.
Integrity. See model integrity.
INTENTION-REVEALING INTERFACES,
246–249, 422–427
Interest calculator examples, 211–215,
217–219, 295–306
Internet Explorer bookmark anec-
dote, 57–59
Invariant logic, 128–129, 143
Inventory management example,
504–505
Investment banking example,
194–200, 211–215, 501
Isolated domain layer, 106–107
Isolating the domain. See ANTI-
CORRUPTION LAYER; distillation;
LAYERED ARCHITECTURE.
Iterative design process, 14, 188, 445

J
Jargon. See PUBLISHED LANGUAGE;
UBIQUITOUS LANGUAGE.
Java entity beans vs. ENTITIES, 91

K
Knowledge crunching, 13–15
Knowledge crunching, example, 7–12
KNOWLEDGE LEVEL, 465–474

L
Language of the domain experts,
206–207
Large-scale structure. See also distillation; examples, large-scale structure; LAYERED ARCHITECTURE; strategic design.

CONTEXT MAP, 446
definition, 442
development constraints, 445–446
EVOLVING ORDER, 444–446
flexibility, 480–481
KNOWLEDGE LEVEL, 465–474
minimalism, 481
naive metaphor, 448–449
overview, 439–443
PLUGGABLE COMPONENT FRAMEWORK, 475–479
refactoring, 481
role in design, 329
supple design, 482–483
SYSTEM METAPHOR, 447–449
team communication, 482

Large-scale structure, RESPONSIBILITY LAYERS
choosing layers, 460–464
overview, 450–452
useful characteristics, 461

LAYERED ARCHITECTURE. See also distillation; examples, LAYERED ARCHITECTURE; large-scale structure.
apPLICATION LAYER, 70, 76–79
callbacks, 73
contceptual layers, 70
connecting layers, 72–74
design dependencies, 72–74
diagram, 68
domain layer, 70, 75–79
frameworks, 74–75
infrastructure layer, 70
isolated domain layer, 106–107
MVC (MODEL-VIEW-CONTROLLER), 73
OBSERVERS, 73
partitioning complex programs, 70
separating user interface, application, and domain, 76–79
SERVICES, 73–74
SMART UI, 73
TRANSACTION SCRIPT, 79

user interface layer, 70, 76–79
value of, 69

LAYERED ARCHITECTURE, ANTI-CORRUPTION LAYER
ADAPTERS, 367
considerations, 368–369
FACADES, 366–367
interface design, 366–369
overview, 364–366
relationships with external systems, 384–385

LAYERED ARCHITECTURE, RESPONSIBILITY LAYERS
choosing layers, 460–464
overview, 450–452
useful characteristics, 461
Legacy systems, phasing out, 393–394
Life cycle of domain objects, 123–124.
See also AGGREGATES; FACTORIES; REPOSITORIES.
Loan management examples. See examples, loan management.
Local vs. Global identity, 127

M

Merging
OPEN HOST SERVICE and PUBLISHED LANGUAGE, 394–396
SEPARATE WAYS TO SHARED KERNEL, 389–391
SHARED KERNEL TO CONTINUOUS INTEGRATION, 391–393
METADATA MAPPING LAYERS, 149
Missing concepts, 207–210
Mistaken identity anecdote, 89
Model integrity. See also BOUNDED CONTEXT; CONTEXT MAP;
multiple models.
establishing boundaries, 333–334
multiple models, 333
overview, 331–334
recognizing relationships, 333–334
unification, 332. See also CONTINUOUS INTEGRATION.
Model layer. See domain layer.
Model-based language. See UBIQUIOUS LANGUAGE.
**MODEL-DRIVEN DESIGN**
correspondence to design, 50–51
modeling paradigms, 50–52
overview, 49
procedural languages, 51–54
relevance of model, 49
tool support, 50–52

**Modeling**
associations, 82–88
ENTITIES, 93–94
HANDS-ON MODELERS, 60–62
integrating with programming, 60–62
non-object, 119–122

**Models**

binding to implementation. See MODEL-DRIVEN DESIGN
and user understanding, 57–59
MODEL-VIEW-CONTROLLER (MVC), 73

**Modularity**, 115–116
MODULES
agile, 111
cohesion, 109–110, 113
costs of, 114–115
coupling, 109–110
determining meaning of, 110
designing for relational databases, 159–161
infrastructure-driven packaging, 112–116
mixing paradigms, 119–122
modeling paradigms, 116–119
modularity, 115–116
naming, 110
non-object models, 119–122
object paradigm, 116–119
overview, 109
packaging domain objects, 115
refactoring, 110, 111
vs. BOUNDED CONTEXT, 335

Monty Python anecdote, 5
Multiple models, 333, 335–340
MVC (MODEL-VIEW-CONTROLLER), 73

**Naming**

BOUNDING CONTEXTS, 345
conventions for supple design, 247
INTENTION-REVEALING INTERFACES, 247

MODULES, 110
SERVICES, 105
Non-object models, 119–122

**O**
Object references. See REPOSITORIES.
Objects. See also ENTITIES; VALUE OBJECTS.
associations, 82–88
creating, 234–235. See also constructors; FACTORIES.
defining, 81–82
designing for relational databases, 159–161
made up of objects. See AGGREGATES; COMPOSITE.
persistent, 150–151
OBERVERS, 73
Off-the-shelf solutions, 407
Online banking example, 71–72
OPEN HOST SERVICE, converting to PUBLISHED LANGUAGE, 394–396
Outsourcing, 408–409
Overbooking examples, 18–19, 222

**P**
Packaging. See deployment; MODULES.
Paint-mixing application, examples, 247–249, 252–254, 256–259
Partitioning complex programs. See large-scale structure; LAYERED ARCHITECTURE.

SERVICES into layers, 107
Patterns, 507–510. See also analysis patterns; design patterns; large-scale structure.

PCB design anecdote, 7–13, 501
Performance tuning, example, 185–186
Persistent objects, 150–151

**N**
Naive metaphor, 448–449
Pluggable component framework, 475–479
Policy pattern. See strategy pattern.
Polymorphism, 435–437
Presentation layer. See user interface layer.
Procedural languages, and model-driven design, 51–54
Processes as domain objects, 222–223
Prototypes, 238–241
Published language
  elephant and the blind men, 377–381
  example, 377
  merging with open host service, 394–396
  overview, 375–377
Q
Quilt project, 479
R
Reconstitution, 145–146, 148
Refactoring
  breakthroughs, 193–200
  during a crisis, 325–326
  deep models, 189–191
  definition, 188
  designing for developers, 324
  discovery, 191–192
  distillation, 437
  exploration teams, 322–323
  initiation, 321–322
  large-scale structure, 481
  levels of, 188–189
  modules, 110, 111
  to patterns, 188–189
  reusing prior art, 323–324
  supple design, 191
  timing, 324–325
Refactoring targets, 437
Reference objects. See entities.
Repositories
  advantages, 152
  architectural frameworks, 156–157
decoupling from the client, 156
designing objects for relational databases, 159–161
encapsulation, 154
example, 172–173
and factories, 157–159
global searches, 150–151
implementing, 155–156
metadata mapping layers, 149
object access, 149–151
overview, 147–152
persistent objects, 150–151
querying, 152–154
references to preexisting domain objects, 149
transaction control, 156
transient objects, 149
type abstraction, 155–156
Requirements gathering. See concept analysis; knowledge crunching;
ubiquitous language.
Responsibility layers
  choosing layers, 460–464
  example, 452–460
  overview, 450–452
  useful characteristics, 461
Reusing code
  bounded context, 344
  generic subdomains, 412–413
  reusing prior art, 323–324
Risk management, 413–414
S
Scenarios, examples, 173–177
Segregated core, 428–434
Selecting objects, 229–234, 269–270
Separate ways, 384–385, 389–391
Services. See also entities; value objects.
  access to, 108
  characteristics of, 105–106
  granularity, 108
  and the isolated domain layer, 106–107
  naming, 105
  overview, 104–105
  partitioning into layers, 107
**Shared kernel**
- example, 359
- merging with continuous integration, 391–393
- merging with separate ways, 389–391
- overview, 354–355

**Sharing value objects**, 100–101

**Shipping examples.** See examples, cargo shipping.

**Side effects**, 250. See also assertions.

**Side-effect-free functions**, 250–254, 285–286

**Simplifying your design.** See distillation; large-scale structure; layered architecture.

**Smart UI**, 73

**Specification.** See also analysis patterns; design patterns.
- applying, 227
- business rules, 225
- combining. See composite specification.
- composite, 273–281
- configuring, 226–227
- definition, 225–226
- example, 29, 235–241, 279–282
- generating objects, 234–235
- implementing, 227
- overview, 224–227
- purpose, 227
- selecting objects, 229–234
- validating objects, 227, 228–229

**Speech, common language.** See ubiquitous language.

**Speech, modeling through**, 30–32

**Standalone classes**, 265–267

**Strategic design.** See also large-scale structure.
- assessing the situation, 490
- customer-focused architecture teams, 492
- developers, role of, 494
- essential requirements, 492–495
- evolution, 493
- evolving order, 491
- feedback process, 493
- minimalism, 494–495
- multiple development teams, 491
- objects, role of, 494
- setting a strategy, 490–492
- team communication, 492
- team makeup, 494
- technical frameworks, 495–497
- Strategy pattern, 19, 311–314

**Supple design**
- approaches to, 282–292
- assertions, 255–259
- closure of operations, 268–270
- composite specification, 273–282
- conceptual contours, 260–264
- declarative design, 270–272
- declarative style of design, 273–282
- domain-specific language, 272–273
- example, 247–249
- intention-revealing interfaces, 246–249
- interdependencies, 265–267
- large-scale structure, 480–483
- naming conventions, 247
- overview, 243–245
- side-effect-free functions, 250–254, 285–286
- standalone classes, 265–267
- system metaphor, 447–449
- System under design, 385–386

**T**

**Team context**, 382

**Teams**
- choosing a strategy, 382
- communication, large-scale structure, 482
- customer-focused, 492
- defining bounded context, 382
- developer community, maturity of, 117–119
- exploration, 322–323
- Teams, and strategic design
- communication, 492
- customer-focused, 492
- developers, role of, 494
- makeup of, 494
- multiple teams, 491
- Teams, multiple
- Anticorruption layer, 364–370
CONFORMIST, 361–363
CUSTOMER/SUPPLIER, 356–360
example, 358–360
SHARED KERNEL, 354–355, 359
strategic design, 491
Terminology. See BOUNDED CONTEXT;
PUBLISHED LANGUAGE; UBQUITOUS LANGUAGE.
Testing boundaries, 351
Transaction control, 156
TRANSACTION SCRIPT, 79
Transformations, 389
Transforming boundaries, 382–383
Transient objects, 149
Translation layers, 374
Tuning a database, example, 102
U
UBQUITOUS LANGUAGE. See also PUBLISHED LANGUAGE.
analysis patterns, 306–307
cargo router example, 27–30
consistent use of, 32–35
designing objects for relational databases, 160–161
domain-specific language, 272–273
language of the domain experts, 206–207
overview, 24–27
refining the model, 30–32
specialized terminologies, 386–387
requirements analysis, 25
speech, role of, 30–32
UML (Unified Modeling Language), 35–37
Unification, 332. See also CONTINUOUS INTEGRATION.
Unified Modeling Language (UML), 35–37
Updating the design. See refactoring.
User interface layer
business logic, 77
definition, 70
separating from application and domain, 76–79
V
Validating objects, 227, 228–229
VALUE OBJECTS. See also ENTITIES; SERVICES.
associations, 102–103
bidirectional associations, 102–103
change management, 101
clustering. See AGGREGATES.
designing, 99–102
element, 167–168
immutability, 100–101
object assemblages, 98–99
overview, 97–99
passing as parameters, 99
referencing ENTITIES, 98–99
sharing, 100–101
tuning a database, example, 102
Vision statement. See DOMAIN VISION STATEMENT.
Vocabulary. See PUBLISHED LANGUAGE; UBQUITOUS LANGUAGE.
W
Waterfall design method, 14
Web site bookmark anecdote, 57–59