

---

# INDEX

---

## A

ABSTRACT CORE, 435–437  
 ADAPTERS, 367  
 AGGREGATES  
     definition, 126–127  
     examples, 130–135, 170–171,  
         177–179  
     invariants, 128–129  
     local *vs.* global identity, 127  
     overview, 125–129  
     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  
     example, 295–306  
     overview, 294  
     UBIQUITOUS LANGUAGE, 306–307  
 ANTICORRUPTION LAYER  
     ADAPTERS, 367  
     considerations, 368–369  
     example, 369–370  
     FACADES, 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  
     example, 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  
 BOUNDED CONTEXT. *See also*  
     CONTEXT MAP.  
     code reuse, 344  
     CONTINUOUS INTEGRATION,  
         341–343  
     defining, 382  
     duplicate concepts, 339–340  
     example, 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* ANTI-  
         CORRUPTION 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
  - FLYWEIGHT, 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, 415–416
  - 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* UBIQUITOUS 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
- E**
- Elephant and the blind men, 377–381
- Encapsulation. *See also* FACTORIES. COHESIVE MECHANISMS, 422–427 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

- associations, 169–170
  - automatic routing, 346–351
  - booking
    - BOUNDED CONTEXT, 337–340
    - extracting hidden concepts, 17–20
    - legacy application, 369–370
    - overbooking, 18–19, 222
    - vs.* yield analysis, 358–360
  - cargo routing, 27–30
  - cargo tracking, 41–43
  - COMPOSITE pattern, 316–320
  - composite routes, 316–320
  - concept analysis, 222
  - conclusion, 502–504
  - constructors, 174–175
  - CONTEXT MAP, 346–351
  - ENTITIES, 167–168
  - extracting hidden concepts, 17–20
  - FACTORIES, 174–175
  - identifying missing concepts, 207–210
  - isolating the domain, 166–167
  - large-scale structure, 452–460
  - MODULES, 179–181
  - multiple development teams, 358–360
  - performance tuning, 185–186
  - refactoring, 177–179
  - REPOSITORIES, 172–173
  - RESPONSIBILITY LAYERS, 452–460
  - route-finding, 312–314
  - scenarios, 173–177
  - SEGREGATED CORE, 430–434
  - shipping operations and routes, 41–43
  - STRATEGY, 312–314
  - system overview, 163–166
  - UBIQUITOUS LANGUAGE, 27–30
  - VALUE OBJECTS, 167–168
  - Examples, concept analysis
    - extracting hidden concepts, 17–20
    - identifying missing concepts, 207–210
    - implicit concepts, 286–288
    - researching existing resources, 217–219
    - resolving awkwardness, 211–215
  - Examples, distillation
    - COHESIVE MECHANISMS, 423–424, 425–427
    - GENERIC SUBDOMAINS, 410–412
    - organization chart, 423–424, 425–427
    - SEGREGATED CORE, 428–434
    - time zones, 410–412
  - Examples, integration
    - ANTICORRUPTION LAYER, 369–370
    - translator, 346–351
    - unifying an elephant, 378–381
  - Examples, large-scale structure
    - KNOWLEDGE LEVEL, 466–474
    - PLUGGABLE COMPONENT FRAMEWORK, 475–479
    - RESPONSIBILITY LAYERS, 452–460
  - Examples, LAYERED ARCHITECTURE
    - partitioning applications, 71–72
    - RESPONSIBILITY LAYERS, 452–460
  - Examples, loan management
    - analysis patterns, 295–306
    - breakthroughs, 194–200
    - concept analysis, 211–215, 217–219
    - CONCEPTUAL CONTOURS, 262–264
    - conclusion, 501–502
    - interest calculator, 211–215, 217–219, 295–306
    - investment banking, 194–200
    - refactoring, 194–200, 284–292
  - Explanatory models, 41–43
  - Explicit constraints, concept analysis, 220–222
  - External systems, 383–385. *See also* integration.
  - Extracting hidden concepts, 17–20. *See also* implicit concepts.
- ## F
- FACADES, 366–367
  - Facilities, 194
  - FACTORIES
    - configuring SPECIFICATION, 226–227
    - creating, 139–141
    - creating objects, 137–139
    - designing the interface, 143
    - ENTITY *vs.* VALUE OBJECT, 144–145

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

example, 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

example, 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
    - naïve 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
    - conceptual 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* UBIQUITOUS 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  
 examples, 111–112, 179–181  
 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

**N**  
 Naive metaphor, 448–449

Naming  
 BOUNDED 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* construc-  
 tors; FACTORIES.  
 defining, 81–82  
 designing for relational databases,  
 159–161  
 made up of objects. *See* AGGRE-  
 GATES; COMPOSITE.  
 persistent, 150–151

OBSERVERS, 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 ARCHITEC-  
 TURE.  
 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



- PLUGGABLE COMPONENT FRAME-  
WORK, 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
    - examples, 177–179, 181–185,  
194–200, 247–249
    - 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 data-  
bases, 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; UBIQUITOUS 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

UBIQUITOUS 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  
 example, 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; UBIQUITOUS LANGUAGE.

## W

Waterfall design method, 14  
 Web site bookmark anecdote, 57–59