

Index

A

- Abstract flow, 59
- Abstract use-case slice, 125
- Access violations, detecting, 330
- AccessingData pointcut, 99
- Actors
 - defined, 391
 - notation, 391–392
- Adaptive programming (AP), 18
 - applying, 235–236
- Advanced separation of concerns, 17–18
- Advices, 18
 - defined, 23
- Analysis
 - conducting, 150–151
 - constructs, 150
 - defined, 148–149
 - level of detail, 151
 - stereotypes, 150
- Analysis class stereotypes, 189
- Analysis element structure, 149
- Analysis model, 148–151
 - architectural view of, 344–347
 - architecturally significant analysis elements, 344–345
 - architecturally significant use-case analysis slices, 346–347
 - language of, 149–150
 - boundary class, 149
 - control class, 149
 - entity class, 149–150
- Application-extension use cases
 - adaptive programming, applying, 235–236
 - allocating use-case behavior to classes, 218–220
 - alternate flows
 - keeping separate, 222–223
 - structuring, 221–222
 - analyzing, 214–220
 - architecture, 321–322
 - classes, identifying, 215–216
 - component interfaces, identifying from use-case extensions, 225–226
 - dealing with changes in the base, 230–236
 - design patterns, applying, 232–235
 - designing, 224–230
 - keeping separate, 220–223
 - multiple extensions to a use case, dealing with, 226–229
 - multiple use cases, extending, 229–230
 - operation extensions, designing, 224
 - pointcuts, identifying, 216–218
 - identifying the structural context, 216–217

408 INDEX

- operations in Reserve Room Handler and Reserve Room form, 218
 - Room.updateAvailability, 217
 - reflection, applying, 232
 - separating functional requirements with, 213–237
- Application layer, 175
 - how use cases help to structure, 176
- Application peer use cases
 - analyzing, 188–194
 - allocating use-case behavior to classes, 190–194
 - identifying classes, 189–190
 - architecture, 320–321
 - components/interfaces, identifying, 203–205
 - design elements
 - identifying, 200–203
 - refining, 205–211
 - designing, 199–205
 - element structure, 195–196
 - keeping separate, 195–199
 - separating functional requirements with, 187–212
 - use-case structure, 196–199
 - non-use-case-specific slices, 196–197
 - use-case slices, 197–199
- Application use cases, 81, 85
 - capturing, 86–93
 - extension use cases, dealing with, 91–93
 - for Hotel Management System, 86
 - use-case variability
 - handling, 88–89, 91
 - identifying, 87–88
- Architectural iterations, 172–173
- Architectural views, 339
 - analysis model, 344–347
 - architecturally significant analysis elements, 344–345
 - architecturally significant use-case analysis slices, 346–347
 - design model, 347–351
 - architecturally significant deployment elements, 348
 - architecturally significant design elements, 349–351
 - architecturally significant process elements, 348–349
 - architecturally significant use-case design slices, 351
 - use-case model, 342–344
- Architectural work, prioritization of, 170
- Architecture
 - bind parameters, 320
 - classes, identifying, 320
 - defined, 168
 - evaluating, 319–337
 - with the help of use cases, 336
 - good architecture, defined, 168–170
 - importance of, 167
 - parameters, identifying, 320
 - use-case behavior, allocating to classes, 320
 - use-case specifics, separating, 320
- Architecture baseline, 167, 340–341
 - defined, 170–171
 - establishing iteratively, 172–173
 - steps to establish, 170–173
 - and use cases, 171–172
- Architecture-centric development, 377
- Architecture description, 173, 334–353
 - and architectural views, 340–341
 - concurrent development of, 341
 - defined, 341–342
 - typical contents of, 352
- Architecture Tradeoff Analysis Method (ATAM), 336
- Aspect orientation, 56
 - adopting, 373–374
 - at construction, 374
 - early elaboration, 373
 - enhancements, 374
 - late elaboration, 374
 - at planning stage, 373
 - defined, 17–18
 - future of, 377–378
 - and iterative development, 365
 - key benefit of, 214
 - relationship to model-driven architecture (MDA), 183–184

- textual specification of, 56
 - Aspect orientation, impact on object orientation, 7
 - Aspect-oriented programming (AOP), xvii–xviii, 255
 - and code generation/merging, 26
 - join points, 43, 64–65
 - Aspect-oriented software development (AOSD), xviii–xix, xxii, 29
 - with use cases, xix–xxi
 - AspectJ, xxiii, 18, 24, 26, 40, 180, 374
 - class extensions, 380
 - dependency checks with, 331
 - logging aspect in, 134
 - operation extensions, 381
 - pointcut expressions and the + character, 251
 - pointcuts, modeling, 384
 - Web site, 19
 - Aspects, 35–36, 375–379
 - applying infrastructure use-case slices with, 251–252
 - defined, 18
 - early, 35–36
 - identifying, 91
 - keeping peers separate with, 19–21
 - AspectWerkz, 18
 - Audit Transaction use case, 96, 98–99, 101
 - Authorization extension, 24
 - Authorization use-case module, 46
 - `AuthorizationHandler` instance, 245
 - Automation, to guide project managers in composing best practices, 378
- B**
- Balance, achieving, 369–370
 - Best-practice weaver, 378
 - Best practices, balancing, 376–377
 - Boundary class, 149
 - «boundary» stereotype, 251
 - Business tier, 265
- C**
- Change cases, 332, 332–333
 - `Check Authorization` extension flow, participating classes in, 247
 - `Check Authorization` interaction, 244–246
 - `Check In Customer` use case, 40, 62, 71, 123, 172, 188, 343, 348
 - `Check In` use-case module, 46
 - `Check Out Customer` use case, 40, 62, 172, 188
 - `Check Room Cost` use case flow, 71–72
 - `Check Room Details` use case, 62, 71
 - `checkAuthorization()` operation, 244–245, 253–254
 - `CheckInCustomer` use-case slice, 41
 - `CheckInHandler` class, 24, 25, 110–118
 - `CheckOutCustomer` use-case slice, 41
 - Child use-case instance, execution of, 75
 - Class definitions, 41
 - Class extensions, 41, 108
 - defined, 380
 - modeling intertype declarations with, 380
 - Classes
 - defined, 389
 - design, 152
 - stereotyped, 153–154
 - notation, 390
 - relationships between, 391
 - usage, 390–391
 - Classifier, in Unified Modeling Language (UML), 58
 - Code generators, and reflection, 232
 - Collaboration, in use-case slices, 117–118
 - Communication diagrams, xxii
 - Complexity
 - and models, 146
 - of system, 329
 - Component-based development, 3
 - Components
 - benefits of, 6
 - building a system with, 4–5
 - defined, 391
 - encapsulation of its contents, 4
 - limitation of, 6–11
 - notation/usage, 392
 - use of term, 7
 - uses of, 3–6
 - Composition filters (CF), 18
 - Concern composition technique, 11–12
 - Concern separation technique, 11–12

410 INDEX

- Concerns
 - defined, 6
 - difference between requirements and, 11
 - Concrete Logging use case, 140
 - Concrete use cases, 91–92
 - ConcreteLogging.java, 142
 - Configurability, 333
 - Control, and architecture, 170
 - Control class, 149
 - Create, read, update, delete (CRUD)
 - functionality, 151
 - CreateReservation(),
 - makeReservation() operation, 23
 - Crosscutting concerns, xx, 3, 7
 - defined, xviii
 - extensions, inability to keep separate, 8–9
 - peers, inability to keep separate, 8–9
 - Crosscutting extensions, 224
 - Customer Types variables, 88
 - CustomerApplicationAuthorization
 - filter, 253, 255
 - CustomerMainForm class extension, 209
 - CustomerScreen class extension, 44
- D**
- Delos development environment, 16
 - Dependency, of system, 329
 - Deployment structure, 152
 - Design classes, 152
 - Design element structure, 152–153, 326
 - Design elements, evaluating, 325
 - Design models, 32, 152–154
 - architectural view of, 347–351
 - architecturally significant deployment elements, 348
 - architecturally significant design elements, 349–351
 - architecturally significant process elements, 348–349
 - architecturally significant use-case design slices, 351
 - conducting design and implementation, 154
 - defined, 152
 - language of, 152
 - minimum design part, 153
 - platform-specific part, 153
 - Design packages, evaluating, 325–327
 - Design patterns, applying, 232–235
 - Development effort estimation, 359–362
 - effort estimation technique, 359–360
 - estimation at the beginning of a project, 360–362
 - refining the estimates, 362
 - Discipline selection, 370–373
 - design and implementation, 372
 - testing, 373
 - use-case modeling and analysis, 371–372
 - displayMenuOptions() operation, 207–208
 - displayReserveRoomOptions, 208
 - Distribution
 - EJB distribution mechanism, 278–281
 - minimal use-case design without, 277–278
 - overlying, 276–290
 - Distribution mechanism
 - applying, 286–290
 - designing, 281–286
 - distribution slice, 283
 - minimal use-case design slice, 283
 - use-case distribution slice, 284–286
 - doFilter(), 253–254
 - Domain classes, 88
 - Domain layer, 175
 - how to organize, 177
 - Downstream models, 155–156
- E**
- Earn and Redeem Credits use case, 92, 93
 - Eclipse Integrated Development Environment (IDE), implementing
 - use-case slices with, 118
 - Effort estimation technique, 359–360
 - EJB container, 334–335
 - defined, 278
 - EJB distribution mechanism, 278–281
 - EJBs, xxiii
 - Elaboration iterations, 172
 - Element structure, 38–39, 147, 174–177
 - application layer, 175
 - domain layer, 175
 - layers, 174–175

- Element under test (EUT), 310
 - Entity class, 149–150
 - Existion, 12
 - extend* relationship, 31, 63–70
 - extension flow declaration, 67
 - extension pointcuts, 65–67
 - extension points, 64
 - extension use cases, 63
 - join points, 64–65
 - UpdatingRoomAvailability
 - extension pointcut, 67–68
 - use-case extensions, execution of, 69
 - extends* keyword, Java, 142
 - Extensibility, evaluating/achieving, 332–333
 - Extension flow declaration, 67
 - Extension pointcuts, 65–67
 - applying, guidelines on, 69–70
 - compared to extension points, 65
 - keeping separate, 127–144
 - Extension points, 12, 31, 42, 64
 - direct reference to, 70
 - indirect reference to, 70
 - occurring in a single step, 69
 - occurring over multiple steps, 69
 - in UML, 43
 - Extension use-case flow, 31, 43
 - Extension use-case realizations, 129–130
 - generalizing, 139–142
 - Extension use cases, 214. *See also*
 - Application-extension use cases
 - and coordination, 92–93
 - dealing with, 91–93
 - defined, 127
 - operation extensions, 130–132
 - outlining, 92
 - realizing, 128–129
 - Extension use-case slices, as reusable elements, 334
 - Extensions, 378
 - building a system with, 375–376
 - early support for, 12–14
 - inability to keep separate, 9–10
 - keeping separate with pointcuts, 127–144
- F**
- Filter chain, 253–254
 - Filter interface, 255
 - FilterChain interface, 253
 - Filters, applying infrastructure use-case slices with, 252–254
 - Flows of events, and use-case instances, 53–54
 - FrontController class, 274
 - Functional decomposition
 - avoiding, 59
 - and structuring of use cases, 78
- G**
- generalization* relationship, 31, 73–76
 - defined, 73
 - generalization and extensions, 76
 - generalizations and inclusions, 76
 - use cases, 35
 - Generalized use-case slice, 121–123
 - classes, 123
 - collaboration, 122–123
 - features, 123
 - Generalizing, 139–142
 - generalizing logging extension, 140
 - GenericLogging.java, 141
 - realizing generalizations, 140
 - Generic Logging use case, 139–140
 - GenericLogging aspect, pointcuts, 141
 - GetQuantityAvailable(),
 - makeReservation() operation, 23
 - Glue code, 10
 - Good architecture
 - control and implementation, separating tests from the units being tested, 170
 - defined, 168–170
 - establishing, 170
 - functional requirements, separating, 169
 - nonfunctional requirements, separating from functional requirements, 169
 - platform specifics, separating, 169–170
- H**
- Handle Authorization use case, 25, 96, 98–99, 101
 - Handle Authorization use-case slice, 249

412 INDEX

- Handle Distribution use case, 323
 - <Handle Distribution> use case
 - brief description of, 277
 - template, 268
 - <Handle Persistence> use case, 291
 - template, 268
 - <Handle Presentation> extension use case, 333
 - <Handle Presentation> use case
 - brief description of, 270
 - template, 268
 - Handle Waiting List use case, 15, 25, 42, 62, 66, 214–215
 - simplified source code to, 23–24
 - Handle Waiting List use-case module, 159–160
 - Handle Waiting List use-case realization, 44
 - Handle Waiting List use-case slice, 44, 321–322
 - Hotel Management System, 5–6, 8–9, 19, 30, 32, 46, 52–53, 73–74, 82–83, 88, 91, 93, 106, 119, 124, 175, 179–181, 188, 227, 230, 290, 332, 342–344, 351
 - application use cases for, 86
 - architecturally significant use cases, 342–343
 - design model, deployment structure for, 180
 - key features of system, 83–84
 - Reserve Restaurant use case, 73–74
 - Reserve Room component, 9–10
 - use cases and classes in, 30–32, 40–41
 - Hotel Reservation non-use-case-specific slices, 178, 180
 - Hyperslices, compared to use-case slices, 40
- I**
- Implementation models, 152–154
 - include* relationship, 31, 70–73
 - inclusions and extensions, 72–73
 - use-case inclusions, 71
 - execution of, 72
 - Included use-case slice, 119–121
 - classes, 121
 - collaborations, 120
 - extensions, 121
 - Infrastructure concerns, 7
 - Infrastructure layers, slices of, 249
 - Infrastructure service packages, 247
 - Infrastructure support packages, 247–248
 - Infrastructure use cases, 81, 85
 - analyzing, 240–247
 - architecture, 322
 - Audit Transaction use case, 96, 98–99, 101
 - capturing, 93–101
 - components in the infrastructure layer,
 - identifying, 255–256
 - describing, 98–100
 - designing, 250–256
 - element structure, refining, 247–248
 - Handle Authorization use case, 96, 98–99, 101
 - infrastructure use-case slice with aspects,
 - applying, 251–252
 - infrastructure use-case slice with filters,
 - applying, 252–255
 - keeping separate, 247–250
 - multiple, dealing with, 256–261
 - and nonfunctional requirements, 100
 - Perform Transaction use case, 94–96
 - relationship with application use cases, 97
 - structuring infrastructure use cases, 96–98
 - systemwide concerns, dealing with, 100–101
 - Track Preferences use case, 96, 98–99, 101
 - Instances
 - defined, 389
 - notation, 390
 - usage, 390–391
 - Instrumentation, and architecture, 170
 - Integration tier, 265
 - Interaction diagrams for, xxii
 - Interfaces
 - defined, 391
 - notation/usage, 392
 - Intertype declarations, 18
 - Intrusive change, 10
 - IRemoteReserveRoom interface, 290
 - IReserveRoom interface, 290
 - IReserveRoomHome interface, 290

IsAuthorized() operation, 25
 Iterative development, 357–358
 activities in an iteration, 359
 and aspect orientation, 365
 phases in a project, 358
 planning, 363
 projects, 377
 shifting emphasis in, 358

J

J2EE. *See* Java 2 Platform Enterprise Edition (J2EE)
 Jaczone WayPointer, xxv
 Java, 152
 Java 2 Platform Enterprise Edition (J2EE),
 xxiii, 265
 core patterns, 279
 relational persistency mechanism in,
 292–293
 Web presentation mechanism in,
 271–273. *See also* Presentation
 mechanism
 Java Database Connectivity (JDBC),
 179–180
 Java Swing, 180
 JBoss-AOP, 18
 Join points, 43, 64–65
 JSP, 152

K

Kiczales, Gregor, xxv, 18, 26

L

“Language Support for Changeable Large
 Real-Time Systems,” xxvi
 Layers, 174–175, 388
 Lieberherr, Karl, xxv–xxvii
 Logging aspect, 132–134
 Logging collaboration, 132
 Logging extension use case, 128–129
 roles in, 129–130
 Logging.java, 134
 LogStream class, 132, 143

M

Maintainability, evaluating/achieving, 332

makeReservation() operation, 21–24
 ReserveRoomHandler class, 130
 testing, 310
 MakingPayment extension pointcut
 UpdatingRoomAvailability, 92
 Methodological guidance, need for, 26–27
 Minimal design, 349–350
 Minimal use-case design, without
 presentation specifics, 270–271
 Model-driven architecture (MDA), 183–184
 Models, 388
 and complexity, 146
 constructs in, 146–147
 defined, 146
 design, 152–154
 downstream, 155–156
 drivers, 147
 implementation, 152–154
 and project deliverables, 147
 updating in development, 155
 upstream, 155–156
 M-to-N mapping, 5
 Multidimensional separation of concerns
 (MDSOC), 18, 40
 Multiple basic flows, 57
 Multiple infrastructure use cases
 dealing with, 256–261
 extension of extension, 259–260
 handling variations in application use
 cases, 261
 multiple extensions at a single execution
 point., 258–259
 separate extensions, 260–261

N

Nodes
 defined, 392
 notation/usage, 392–393
 Nonfunctional requirements, separating,
 239–262
 Non-use-case-specific slices, 123–125,
 177–178
 NoRoomException, 23–24
 Notation guide, 387–393
 actors and use cases, 388–389
 classes and instances, 389–391
 components and interfaces, 391–392

414 INDEX

- package, 387–388
 - processes and nodes, 392–393
- O**
- Object-Oriented Software Engineering: A Use-case Driven Approach* (Jacobson et al.), 15, 60
 - Operation extensions, 43, 130–132, 224
 - declaration, 131–132
 - modeling advices with, 381
 - semantics of *around*, 382–383
 - semantics of *before and after*, 382
 - structural and behavioral context, 130–131
 - Ossher, Harold, xxv, xxvii
- P**
- Packages
 - defined, 387
 - relationships between, 388
 - tier, 388
 - usage, 387–388
 - Parameterized element, 142
 - Parameterizing pointcuts, 135–139
 - in aspect-oriented programming (AOP), 138–139
 - defining parameters, 137–138
 - identifying parameters, 136–137
 - Parameters
 - defining, 137–138
 - parameterized logging use-case slice, 137–138
 - using sequence diagrams with, 136
 - Peer use-case realizations
 - class extensions, 108
 - classes, identifying, 108–110
 - classes participating in, 176
 - collaborations, 106–108
 - keeping separate with aspects, 105–126
 - overlap between, 110–111
 - Peer use cases, 321. *See also* Application peer use cases
 - composing, 12
 - keeping separate, 40–42
 - realizations, 44
 - Peers, 8
 - keeping separate with aspects, 19–21
 - <Perform Transaction> use case, 24–25, 94–96, 250, 268–269, 343–344
 - interaction diagram, 243
 - relationship with application use cases, 97
 - Persistence
 - minimal use-case design without, 291–292
 - persistency mechanism
 - applying, 297
 - designing, 293–296
 - entity-persistence slice, 295–296
 - minimal use-case design slice, 295
 - persistence slice, 296
 - use-case persistence slice, 296
 - relational persistency mechanism, in J2EE, 292–293
 - Persistency, overlaying, 290–297
 - Platform, choosing, 179–180
 - Platform-independent structure, 174–179
 - element structure, 174–177
 - and functional requirements, 174
 - use-case structure, 174, 177–179
 - Platform-specific extension use cases, architecture, 323
 - Platform-specific model (PSM), 183
 - Platform-specific slices, 182
 - Platform-specific use-case slices, separating platform specifics with, 263–300
 - Platform specifics, 349–350
 - automatically incorporating, 269
 - distribution, overlaying, 276–290
 - incorporating into use cases, 264
 - keeping separate, 181–183, 264–269
 - overlaying, 179–184
 - persistency, overlaying, 290–297
 - tiers, modeling with infrastructure use cases, 268–269
 - use-case structure, preserving, 298–299
 - user interfaces, overlaying, 269–276
 - Pointcuts, xxiv, 18, 43, 127, 133–135
 - defined, 23
 - modeling, 383–384
 - parameterizing, 135–139
 - POJOs (Plain Old Java Objects), 266–267, 272, 298
 - Portability, evaluating/achieving, 333
 - Presentation mechanism
 - applying, 275–276

- designing, 273–275
 - Minimal Use-Case Design slice, 274
 - presentation slice, 274
 - use-case presentation slice, 274–275
 - Presentation tier, 265
 - Process mapping, 349–351
 - Process structure, 152
 - Processes
 - defined, 392
 - notation/usage, 392–393
 - Productivity
 - gains, by keeping concerns separate, 365–367
 - improving, 364–365
 - Project phases, 358
 - Project planning/control, 363–365
 - estimating project delays, 363–364
 - keeping the project on track, 364–365
 - Provide Cached Access use case, 100–101
- R**
- Rational Unified Process (RUP), xxv
- Rectangular classifier notation, 392
- Reflection, applying, 232
- Relational persistency mechanism, in J2EE, 292–293
- Relationships, use-case modules, 158–159
- Remote Method Invocation (RMI), 180
- Requirements, 11
- Reservation Channels variables, 88
- Reservation entity class, 297
- Reservation-persistence slice, 297
- Reservation state chart, 210
- Reserve Facility use case, 62, 74–75
- Reserve Facility use-case module, 158
- Reserve Restaurant use case, 73–75
- Reserve Room distribution slice, 287
- Reserve Room use case, 15, 40, 42, 62, 71, 74, 123, 188, 214–215, 343, 345, 348
 - using ellipse notation, 58
- Reserve Room use case instance, execution of, 71
- Reserve Room use-case module, 46, 158–159
- Reserve Room use-case slice, 112–113, 198, 216, 321
- Reserve Room use-case specification, 55–56, 189
- ReserveRoom class extension, 44
- ReserveRoom use-case slice, 41
- ReserveRoomDistributionClient,
 - source code for, 288
- ReserveRoomForm class, 198
- ReserveRoomHandler class, 22–25, 129–130, 134, 142, 198, 288–290
 - makeReservation() operation, 130
- ReserveRoomHandler, simplified source code to, 22–23
- Resilient architecture, 167–185, 378. *See also* Architecture
- retrieve() operation, 112, 131–133
- Reusability, evaluating/achieving, 334
- Reusable Asset Specification (RAS), 15
- Room class, 123–124
- roomAccessOperation parameter, 135–137, 139–140
- RoomAccessor parameter, 135–136
- RoomAccessor pointcut, 141
- roomCall pointcut, 135–137, 140, 143
- S**
- Scattering, 8–9
- Selection, 208
- selectMenuOption(), 209
- Separation of concerns, xviii, 7
 - advanced, 17–18
 - automating the evaluation, 329–330
 - defined, 332
 - design elements, 325
 - design packages, 325–327
 - enforcing, 330–331
 - evaluating, 324–332
 - use-case structures, 327–328
- Sequence variator, xxvii, 13–14
- Service packages, 216
- Servlets, xxiii
- Size, of system, 329
- Skinny system. *See* Architecture baseline
- Software development
 - difficulty of, xvii–xviii
 - and model building, 145
- Software Reuse* (Jacobson), 15
- SQL, 152

416 INDEX

Stakeholder concerns
 functional and nonfunctional
 requirements, dealing with, 84–85
 problem domain, 82–83
 system features, eliciting, 83–84
 understanding, 81–85

Stereotypes, analysis, 150

Structural context, identifying, 216–217

Subflow, referencing, 57

Subject-oriented programming, and
 hyperslices, 40

Summation, 207–208

Supporting classes, 313

System(s)
 development effort estimation,
 359–362
 effort estimation technique,
 359–360
 estimation at the beginning of a
 project, 360–362
 refining the estimates, 362

iterative development, 357–358
 activities in an iteration, 359
 and aspect orientation, 365
 phases in a project, 358
 planning, 363
 shifting emphasis in, 358

and models, 146–147

planning/controlling the project,
 363–365
 estimating project delays, 363–364
 keeping the project on track,
 364–365

platform specifics, 179–184

Systemwide concerns, evaluating/achieving,
 332–336
 extensibility, 332–333
 maintainability, 332
 performance and reliability, 334–336
 portability, 333
 reusability, 334

T

Tangling, 8–9

Template parameter, 135–136

Templating use-case slices, 142–144

Test-case slices, 323–324

Three-tier systems, 264–266

Tier packages, 266–267, 388
 use cases and tiers, 267

«trace» dependencies, 157

Track Preferences use case, 96, 98–99, 101

U

UML aspect, 132

UML package «merge» dependency, 160

Unified Modeling Language (UML), xxii
 classifier in, 58
 extension points in, 43
 modeling aspects and use-case slices in,
 379–385
 parameterized element, 142
 support for extensions in UML, 15–16

updateAvailability(), 107, 109, 112,
 116–117, 122, 122–123, 205, 217–220

Upstream models, 155–156

Use-case analysis structure, 149

Use-case behavior, allocating to classes,
 218–220

Use-case design structure, 152

Use-case diagram, 52

Use-case-driven development, 32–33,
 376–377
 models, 147

Use-case extensions, 15

Use-case flows
 describing, 54
 visualizing, 57–59

Use-case instances
 execution of, 53
 and flows of events, 53–54

Use-case model, 147–148
 architectural view of, 342–344
 partitioning into use-case packages, 52
 preserving the structure of, 155–156
 and traceability, 158
 and user's/stakeholders' concerns, 148

Use-case modeling, 31, 35, 51–53
Use-Case Modeling (Bitner/Spence), 60

Use-case modules, xx, xxiv, 37–47
 architecture, 46–47
 building systems with, 145–161
 composing and configuring, 159–161
 composing iteratively, 46–47

- cutting across models, 154–159
 - defective, removing out, 160–161
 - deriving relationships from use cases, 158–159
 - developing, 45–47
 - incremental development with, 46
 - parallel development with, 45
 - relationships, 158–159
 - use-case slices within, 157–158
- Use-case realizations, 32–33, 175
- Use-case relationships, 31, 61–62
- extend* relationship, 31, 63–70
 - generalization* relationship, 31, 73–76
 - include* relationship, 31, 70–73
- Use-case scenario, 54
- Use-case slices, xx, xxiv, 151, 153, 379
- building systems in overlays with, 38–40
 - collaboration in, 117–118
 - composing peer use-case realizations with, 42
 - defined, 37
 - and element structure, 38–39
 - element structure space and, 39
 - extending use-case slices, 260
 - extension, keeping separate, 42–45
 - generalized use-case slice, 121–123
 - implementing with Eclipse, 118
 - included use-case slice, 119–121
 - keeping infrastructure separate in, 248–250
 - modeling, 384–385
 - non-use-case-specific slice, 123–125
 - relationship to hyperslices, 40
 - as reusable elements, 334
 - templating, 142–144
 - updating models in development, 155
 - using to improve reuse, 179
- Use-case specifications, 148
- template, 56
- Use-case-specific class extensions, composing, 115–117
- Use-case-specific classes, composing, 113–115
- Use-case specifics
- contents of, 111–112
 - keeping separate, 110–118
- Use-case structure, 39, 147, 150–151, 174, 177–179
- evaluating, 327–328
 - preserving, 298–299
 - slices in, 177
- Use-case technique, 60, 214
- defined, 51
- Use-case test slices
- control, 311–312
 - designing tests, 310–317
 - elements to be tested, identifying, 308–310
 - implementing tests, 310–317
 - instrumentation, 312
 - separating tests with, 301–317
 - test cases
 - design, 316–317
 - identifying from use cases, 303–308
 - test data and verification, 314
 - test infrastructure, designing, 312–316
 - test infrastructure slice, 315
 - test sequence, controlling/instrumenting, 313–314
 - test-case slice, 315–316
 - test-first approach, 301–303
 - use-case design slice, 315
- Use-case variability
- handling, 88–89, 91
 - identifying, 87–88
- Use-case variables, organizing alternate flows with, 89
- Use cases, 29–31
- advantage of, 34
 - and required usages of a system, 52
 - aspects, 35–36
 - early, 36
 - capturing concerns with, 81–102
 - and classes, 33
 - defined, 30, 391
 - describing, 54–57
 - flows, 59–60
 - functional decomposition, avoiding, 59
 - modeling concerns with, 51–60
 - multiple basic flows., 57
 - notation, 392
 - peer, keeping separate, 40–42
 - referencing subflows, 57
 - roles/benefits of, 34



418 INDEX



specification template, 55–56
structuring, 61–79
technique, gaps in, 34–35
and tiers, 267
usage, 392
use-case specifications compared to, 57
utility, 77
when to use rectangular notation for, 59
Use-Cases: Patterns and Modeling Problems
(Overgaard), 60
User interfaces, overlaying, 269–276
Utility extensions, 77
Utility inclusions, 77

Utility use cases, 77
types of, 77

W

Waiting List Handler, 216
Waiting List service package, 216
`WaitingList` class, 44
Web container, 334–335
`withincode`, 135

Z

Zooming, 180

