



Index

NUMBERS

3GLs, 116

4GLs, 595

A

abstract class, 290

abstract data type (ADT), 420

abstraction gap, 41–42, 60–61,
94–95

abstractions, 344–345

applying, 54

benefit of higher levels of, 116

black-box, 56–57

business applications, 147

composability of, 399

creating mappings between,
97–99

defined, 55–56, 456

description of, 42–44

developing, 54

encapsulation as (*see*
encapsulation)

and evolution of solution
domains, 65–66

that expose metadata, 398–399,
401

and finding mappings, 94–95

function of, 49

general, 205

granularity of, 58–59

gray-box, 58

implemented by transformation,
475

importance of raising level of,
139

language-based, 63–64

levels of, 347

with mappings to general
programming languages, 460

models as, 213–214

modes of delivery, 61–66

packaging, 61–66, 139

and pattern languages, 201–202

in pattern languages, 209–210

and patterns, 499

platform-based (*see*
platform-based abstractions)

progressive, 49–50

raising level of, 53–61, 67–68

refactoring software in, 107

relating with mappings, 461–462

relationship to power, 275

relative levels of, 64–65

reusing knowledge through,
56–60

in software development, 49–50

642 Index

- abstractions, (*cont.*)
 - specificity of, 59–60
 - systematic application of, 60
 - view's levels of, 267–272
 - white-box, 57–58
 - working bottom-up with, 50–51
 - working top-down with, 50–51
- abstract syntax, 280, 285–298
- abstract syntax graphs (ASGs), 282, 293, 307–309, 457
- abstract syntax representation, 457–458
- abstract syntax trees (ASTs), 282, 288, 294, 299, 456
- accessibility, 47
- accidental complexity, 36–37, 38–41
- active guidance, 153–154
- activities, building for product line, 553–554
- Activity Execution Language (AEL), 535
- activity graphs, 366
- adaptability
 - of interface specifications, 422–423
 - of pattern-based abstractions, 399
- adapters, 232
- adaptive design, 105, 343–344
- advice, 465
- affordance, 47
- agile development, 102–104, 580–584
- agile development movement, 123
- agile methods, 123–125
- agile modeling, 577–579
- agility
 - balancing with structure, 152–153
 - defined, 102
 - and encapsulation, 114
 - increasing, 139, 150
 - measurable definition of, 103–104
 - in software development, 102–104
- analysis, 69
- analysis-paralysis, 13
- anti-patterns, 108
- architectural description
 - applied to model-driven development, 254–255
 - defined, 135
 - standards for, 135–137
- Architectural Description Standard (ADS), 136
 - defining, 137
 - versus software factory schemas, 167–168
- architectural mismatch, 131, 132
- architectural patterns, 433–435, 436–437
- architectural styles, 135
- architecture-driven development, 134–137
- architectures
 - aligning in supply chains, 356
 - for building software tools, 328–333
 - conflicting assumptions about, 113
 - defining for software product family, 174–175
 - derivation of products', 385–386, 387
 - development of, 526–546
 - for product lines (*see* product architecture)
 - for software product families, 342–344, 432–437 (*see also* product architecture)
 - summary of, 541–542
 - using application patterns of, 528–529
- arguments, bound to patterns, 244
- artifacts
 - defining relationships between, 169
 - external to models, 243–244
 - modeling, 267
 - synchronizing, 172
- ASICs, 6
- aspect frameworks, 263

- AspectJ, 465–466
- aspect-oriented programming (AOP) 71
 - adaptation of, 133
 - aspects in, 464
 - code tangling, 259
 - dealing with languages that don't support, 469
 - delocalization and, 463–465
 - in domain-specific languages, 468–470
 - example using AspectJ, 465–466
 - example using .NET, 467–468
 - function of, 260
 - goals of, 262
 - implementation of, 466
 - problems with, 470
 - shortcomings of, 133
 - versus variable encapsulation, 133–134
- aspects
 - binding to functional components, 263, 264
 - code tangling, 259
 - combining with functional components, 261–263
 - coupling, 262–263
 - defined, 71, 463
 - defining implementation of, 261
 - describing, 466
 - fixed set of, 263
 - implementation of, 259
 - join point for, 464
 - localizing, 464
 - separating, 260–261
 - weaving, 261–262, 468
- aspectual composition, 261
- aspectual decomposition, 259–261
- aspect weaving, 493–495
- assembly
 - development by, 185
 - by orchestration, 137–139
- assembly references, 224
- assembly technologies, 130–139
- asset packaging, 557
- asset provisioning, 548
- associations, 290, 407
- attributes, 290
- Auction Manager, 247
- auto completion, 504
- automation
 - changes in pattern application with, 240–241
 - through code generation, 77–78
 - model-driven development (MDD) for, 141
 - of patterns, 238
 - through Rapid Application Development (RAD), 79–80
 - of refactoring software structure, 248–249
 - tasks for, 232
 - through Round Trip Engineering (RTE), 81–82
 - through visual assembly, 78–79
- automation plan, 379
- B**
- Backus-Naur form (BNF), 286–287
- backward traceability, 473–474
- batch applications, 9–11
- beachhead principle, 584
- behavior, specification of, 420–422, 441–442
- behavioral models, 266
- behavioral views, 272
- bidirectional association, 290
- binding, 295–296, 297
 - of variable features, 371, 372, 378
- BizTalk Server, 443
- black-box abstractions, 56–57, 62, 63
- black-box languages, 211
- black-box mechanisms, 399
 - adapting, 422–423
 - applying, 505–508
 - See also* customization
- black-box transformations, 462, 501, 507
- bootstrapping, 311, 322, 332–333
- bottom-up development, 178–179
- Bowstreet Factory, 507
- BPEL tools, 554–556

644 Index

- brittleness
 - problem of, 89–90
 - reducing, 104–105
 - business application
 - development, 584–588
 - business applications
 - domain-specific language (DSL)
 - for, 255
 - problems in building, 115–116
 - specification for, 458–459
 - business case analysis, 373–374, 524–526
 - business component architecture, 434
 - Business Delegate, 246–248
 - business entity models, 366
 - business entity services, 535–539, 549–553
 - Business Entity viewpoint, 181
 - businesses
 - application of Internet in, 23
 - money spent on software
 - development by, 3
 - stitching applications together
 - for, 21–23
 - business models, 275–276
 - business PC, and software
 - integration, 16–17
 - business problem, programming
 - languages for, 60
 - business processes
 - automating, 21
 - building applications with,
 - 27–30
 - and entity models, 523–524
 - implementation of, 30, 533–535
 - map of, 27
 - for software product lines,
 - 516–519
 - Business Process Execution Language (BPEL), 139, 451
 - Business Process Management Systems (BPMS), 25–27
 - business process models, 366–368, 520–521
 - business process services, 531, 533–535, 554–557
 - Business Process viewpoint, 181
 - business protocols, 439–441, 442–443
- C**
- C#
 - efficiency of, 209–210
 - language of, 207–208
 - partial classes in, 476–477
 - programming idioms and, 211
 - and references to models, 221
 - C++, 64
 - campaigns, 517
 - cash box delivery, 103
 - catalog management, 518
 - catalog manager, 550
 - catalogs, queryable, 399, 423
 - Catalysis, 417–418, 421–422, 427–431
 - CBDiForum, 431
 - certification, 132
 - change
 - and prevention of stagnation,
 - 101–104
 - in problem domain, 100
 - reduction of brittleness from,
 - 104–105
 - reduction of fatigue from,
 - 105–108
 - responding to, 87–88
 - in solution domain, 100–101
 - sources of, 100–101
 - using object orientation for
 - dealing with, 101–108
 - choreography, of Web services, 451
 - class diagrams, 254–255
 - classes, 290
 - class frameworks
 - adaptability of, 406–407
 - building, 407–411
 - definition of, 403–404
 - development using patterns,
 - 405–406, 407–408
 - extension mechanisms, 403–404
 - inversion of control in, 404–405
 - class libraries, 395–396

- class modeling, 236, 237
- client programs, 62
- client/server applications, 17–18
- CLIPS, 491
- closures, 221, 242
- coarse-grained abstraction, 58–59
- code
 - adding to models, 274–275
 - advice, 465
 - join point in, 464
 - pointcut in, 464–465
 - protecting, 475–476
 - tracing to model, 473–474
 - See also* source code
- code generation, 77–78, 236–237
 - with code model, 480
 - combined with code
 - visualization, 81–82
 - importance of customizing relationships for, 170–171
 - keys to effectiveness of, 143–144
 - refinement as, 473
 - through visual assembly, 78–79
 - weak, 118–119
- code models, 480–482
- code tangling, 259
- code templates, 491–493
- code visualization, 76–77, 81–82, 236, 237
- Collaborative Supply Planning process, 518
- collateral development, 385
- COM, 416
- commonalities, 339, 344–345, 348–350, 365–366, 371, 376–377, 410
- Communicating Sequential Processes (CSP), 443
- compatibility, 224–225
- compilation, 77–78
- compilers
 - CORBA IDL, 63–64
 - for general-purpose programming languages, 489–490
 - horizontal transformations in, 456
 - for programming languages, 148–150
- compile-time binding, 476–477
- complementary specification, 273–274
- complete derivation, 169
- complexity
 - from abstraction, 41–44, 49–50
 - accidental, 36–37
 - defined, 35
 - essential, 36
 - and feature delocalization, 38–42
 - and levels of abstraction, 64–65
 - and object orientation, 66–68
 - recurrence of, 55
 - reducing, 139
- component-based development (CBD), 70, 110
 - in Catalysis, 417–418, 427–431
 - issues addressed by, 426–427
 - layered application
 - architectures, 433–435
 - readings on, 431
 - in UML Components, 431
- component outsourcing industry, 257–258
- components
 - conflicting assumptions about, 111–112
 - conflicting assumptions about relationships between, 112–113
 - contracts between, 131
 - systematic reuse of, 125–129
- component specification model, 558–559
- composability, of platform-based abstractions, 399
- computational objects, 91, 92
- computation-independent view, 271–272
- Computer Aided Software Engineering (CASE), 118–119, 595

646 Index

- conceptual views, 269, 270, 271–272
- concrete syntax, 281–282, 299–306
- configuration management tool, 195
- consequences, 75
- containers, 426
- constraint-based scheduling, 152–153
- constraints, 178, 295, 356–357
- context, 74
- context dependencies, 396–397, 399
- context-free grammars (CFGs), 286–288, 297–298
- continuous synchronization, 150, 490
- contracts, 438–442
 - defined, 131
 - enforcing, 131
 - proposed languages for, 450–451
 - schemas of, 444–445
- CORBA (Common Object Request Broker Architecture), 111, 402, 416, 417, 425
- CORBA IDL compiler, 63–64
- Core J2EE Design Patterns, 148
- coupling, 262–263
- create, read, update, and delete (CRUD) actions, 537
- credit card authentication, 553–554
- critical innovations, 161–163
- cross-functional code, 21–22
- custom application developers, 15
- customer databases, 90
- customer management, 517
- Customer Relationship Management process, 517–518
- customer relationships, management of, 187, 517–518
- customer self-service, 517–518
- customer service, 517
- customization, 344–345, 373, 385, 399, 402
- D**
- Database Management Systems (DBMS), 25–26
- database management technology, 11
- databases, 12, 19, 90
- data communications, 11
- data entry, during terminal host era of platform technology, 12–13
- data exchange, 23
- data independence, 11
- data integration, 11, 13, 19, 120
- data processing, during terminal host era of platform technology, 12–13
- data storage, during Batch Era, 10–11
- data structures, 483
- datatypes, 290
- DCOM, 111, 402, 416, 417, 425
- deadlock, 442
- debugging tools, 256
- decision tree, 52–53, 338
- declarative languages, 227–228
- declarative specification, 228–229, 272–273, 274
- declarative views, 272
- decomposition, 346, 458
- Decorator pattern, 407
- deferred encapsulation, 132–134
- definition property, 398
- delocalizations, 38–41, 462–465, 501–502
- delocalizing transformations, 149
- dependencies, 262, 423–426
- deployment
 - automating, 249–251
 - descriptors for, 249
 - environments of, 251
 - manifests of, 426
- derivation, 458
- design
 - agile, 580–584
 - automating, 566–567

- of business application software, 584–588
 - of class frameworks, 408
 - component-based (*see* component-based development (CBD))
 - defined, 69, 233
 - degree of determinism in, 348–350, 352, 377–378
 - for distributed execution environments, 400
 - of protocols, 442–443
 - of software requirements, 69–73
 - Design By Contract (DBC), 69–70
 - design decisions, 44
 - binding mode of, 371–372
 - binding role of, 372
 - binding time of, 365–366, 371, 378
 - bound in advance, 342–345
 - left variable, 344–345
 - and solution structure, 52–53
 - design documentation, 73–75, 76–77
 - Design for Deployment, 183
 - design patterns. *See* patterns
 - design problems, pattern of, 72
 - design time binding, 476
 - determinism, in design
 - development, 348–350, 377–378
 - development process, 121–125
 - development tools, 120, 219, 220
 - digital signature, 226
 - discontinuous innovation, 32–34
 - disposable software, 391
 - distributed execution
 - environments, 400, 402
 - Distributed Object pattern, 529
 - distributed objects, 402, 416–417
 - distribution, of pattern-based
 - abstractions, 400
 - domain models, 128, 365
 - domains, 91–94, 255–259
 - domain-specific languages (DSLs), 142–143, 571
 - abstract syntax for, 457–458
 - for activities, 535
 - aspects in, 468–470
 - bootstrapping, 332–333
 - for business applications, 255
 - for business models, 275–276
 - composing, from sublanguages, 322
 - custom, 256–257
 - defined, 255
 - difficulties in building, 276–277
 - and framework design, 145
 - function of, 255
 - versus general programming languages, 481
 - graphical editor of, 500
 - high-level, 146
 - implementers of, 500
 - low-level, 146
 - mappings between, 461
 - mappings from, 460
 - and model-to-model transformations, 471–473
 - model transformations with, 146–150
 - and pattern languages, 148
 - recording traceability
 - information with, 488–489
 - standardized, 256
 - technical concerns for, 276–277
 - tools for, 255–256
 - for user processes, 532–533
 - views of models based on, 459
 - for Web service connectivity, 539–540
- downstream product lines, 355
 - Dublin Core, 450
 - Dynamic Systems Initiative, 500
- E**
- eager encapsulation, 113–114
 - Eclipse Modeling Framework, 334
 - eCommerce, 26–27, 36
 - eCommerce application, 512–515
 - economics of reuse, 156–157
 - economics of scale, 157–160

648 Index

- economies of scope, 158–160, 348–350
 - editors
 - for domain-specific language, 255
 - pattern support by, 241–242
 - efficiency, 48
 - Eiffel, 421
 - elaboration, 458
 - embedded query languages, 490–491
 - encapsulation, 67–68, 395–396
 - and agility, 114
 - and Component-Based Development (CBD), 70
 - defining boundaries of, 68–73
 - defining languages with, 208–210
 - description of, 206–208
 - formalizing pattern languages with, 210–212
 - in monolithic construction, 113–114
 - of pattern languages, 148
 - reducing software brittleness with, 104–105
 - static, 113–114
 - variable, 132–134
 - Enterprise Application Integration (EAI) software, 21
 - Enterprise Integration Patterns language, 211
 - Enterprise JavaBeans, 504–505
 - Entity Business Components, 436
 - entity models, 523–524
 - eSales process, 518
 - essential complexity, 36
 - evaluation, of patterns, 245–246
 - event handlers, 471–473
 - events, predicting order of, 472
 - executables
 - abstraction gap of, 61
 - deriving from requirements, 99
 - development of, 388–390
 - execution domain, 45–46
 - invalidation of, 100
 - mapping general programming languages to, 462
 - and refinement, 44–45
 - versus requirements, 99
 - in solution domain, 100–101
 - solution domain in, 92
 - execution domain, 62
 - execution traces, 313–318
 - expressions, translation of, 207
 - extensibility, 48
 - eXtensible Markup Language (XML). *See* XML (eXtensible Markup Language)
 - extension libraries, 508
- F**
- façade service, 499
 - Factory Method pattern, 406
 - family of problems, 337–339, 340–342
 - FAST method, 349, 372, 431
 - fatigue
 - problem of, 89
 - reducing, 105–108
 - fault tolerance, 47
 - feature models, 368–370, 519–520
 - mapping to XML schemas, 369, 373
 - notation for, 369
 - semantics for, 369
 - for solution domain, 521–523
 - specializing, 373
 - features
 - in Aspect-Oriented Programming (AOP), 71
 - defined, 95
 - delocalization of, 38–42
 - limiting interaction of, 68
 - types of, 96
 - file abstraction, 49, 55. *See also* abstractions
 - fine-grained abstraction, 58–59
 - forced feature dependencies, 546–547
 - formalism, 122–123
 - formal languages, 280–282
 - formal model. *See* models

- formal process, 83–84
- formal processes, 151
- Form designer, 145
- Forte (Sun Microsystems), 334
- framework completion, 143–145
- frameworks
 - for activities, 535
 - in Catalysis, 429–431
 - difficulties with, 162
 - function of, 125
 - generating by distributed applications, 205
 - versus pattern languages, 204–205
 - for user processes, 532–533
 - for Web service connectivity, 539–540
 - See also* class frameworks
- functional components, 261–264
- Fusion, 427
- G**
- general-purpose programming
 - language (GPL), 229
 - abstract syntax for, 457–458
 - versus domain-specific languages, 481
 - extensions of, 506
 - function of, 255
 - mappings to, 460
 - mapping to executables, 462
 - meta programming facilities in, 506
 - transformations in, 456
- general-purpose programming language compilers, 489–490
- generated code, 475–479
- generated delegate, 479
- generated subclass, 478
- generators, 501, 505, 507–508
- Generic Modeling Environment (GME), 335
- grammar, 207
- granularity, 58–59
- graphical concrete syntax, 299–306
- graphical editors, 500
- graphical notation, 227
- graphical visualization, 214–216
- gratuitous generality, 115–120
- gray-box abstraction, 58
- gray-box languages, 211
- grey-box transformation, 501
- grey-box transformation systems, 503–505
- grids, for software factory
 - schema, 164–165
- grouping mechanisms, 345
- guidelines, 75
- H**
- hand-coded artifacts, 61
- hardware
 - client/server era of, 16–18
 - personal computing era of, 14–16
 - terminal host era of, 11–14
 - See also* platform technology
- hidden regions, 475–476
- horizontal transformations, 149, 456
 - delocalization with, 462, 463
 - examples of, 465–470
 - optimization with, 462
 - refactoring rules for, 495–496
 - refactoring with, 462–463
 - specifying, 493–496
- I**
- identification, 295–296, 297
- IF parts, 485, 486, 488, 491, 492
- IF-THEN parts, 485–490
- immutable software, 88, 89
- imperative code, 274
- imperative languages, 228
- imperative specification, 228, 273–274
- imperative views, 272
- implementation, 75
 - defined, 194, 208, 233
 - execution of, 209
 - generating from intent, 235–236
 - relationship to intent, 194, 195
- implementation asset
 - provisioning, 379–381

650 Index

- implementation assets, 127, 157
 - implementation details, 208
 - Independent Software Vendors (ISVs), 126
 - industrialization, 5, 155–161, 188
 - information
 - captured by development tools, 120
 - capturing for software models, 264–267
 - information systems, 18–19
 - information technology, 15–16
 - inheritance, 170–171, 290, 293, 406–407
 - inlining, 463
 - input specification, 505–506
 - instances, 290
 - integration context, 265
 - intent
 - captured by patterns, 197–198
 - capturing, 195–197, 233–236
 - description of, 194–195
 - after development of implementation, 196
 - difficulty of expression of, 196
 - expression of, 234
 - formal expression of, 235
 - generating implementation from, 236
 - informal expression of, 235
 - and Rapid Application Development (RAD), 196–197
 - relation to implementation, 194, 195
 - interactive transformation, 502–503
 - interface, and Design By Contract (DBC), 69–70
 - Interface Description Language (IDL), 417
 - interface specification models, 536
 - interface specifications, 415–423
 - in Catalysis, 421
 - contracts between, 438–442
 - defining invariants and behaviors, 417–418, 419–422
 - languages, 421
 - provided and required
 - interfaces, 424–425
 - interleaving, 463
 - intermediate domain, 45–46
 - Internet technologies, 23, 24
 - interoperability, 49
 - interpretive weaving, 261–262
 - invariants, specification of, 419–420
 - inventory replenishment, 555–556, 559–560
 - inverse mapping, 169
 - inversion of control, 404–405
 - iterative development, 95, 98–99, 101–102
- J**
- Java, 60, 221
 - jitting, 481
 - join points, 262, 464, 467
 - J2EE, 59, 90, 130, 403, 426
 - J2SE, 59
- K**
- Kent Modeling Framework, 334–335
 - Korba, 431
- L**
- language-based abstractions, 63–65
 - language-based tools, 162–163
 - language families
 - defining, 321–327
 - tools for implementing, 328–336
 - Language Framework pattern, 529, 530
 - languages
 - defining and implementing
 - domain-specific, 322–327
 - formal, 279–282
 - mathematical definitions of, 310
 - metacircular, 311, 322, 332–333
 - meta-tools for designing, 333–336

- modeling, 284–285, 318–319
 - translation of, 310, 312
 - See also* programming languages
- Larch, 421
- latency, 48
- layered application architectures, 433–435
- Layers pattern, 433, 499, 528
- lazy loading, 473
- legacy connector services, 542
- lexical analysis, 299
- life cycle, of software, 351, 405
- little languages, 571
- livelock, 442
- Logical Data Center Designer, 545–546
- Logical Systems Architecture viewpoint, 182
- logical views, 269–270
- M**
- maintainability, 48
- malleability, 48
- malpractice, 89
- mapping parameters, 485, 502–503
- mapping rules
 - application of, 501
 - context of, 498
 - in declarative language, 491
 - function of, 485
 - executing, 485–486
 - IF-THEN parts of, 485–489
 - implementing, 489–493
 - patterns as sets of, 496–500
 - problem definitions of, 497
 - problem/solution descriptions of, 498
 - using traceability information in, 487–489
- mappings
 - of available technologies, 542–543
 - between domains, 92–94
 - between domain-specific languages, 461
 - between viewpoints, 183
 - choosing, 96–97
 - computable, 169–170
 - computations across, 170
 - defined, 93
 - from domain-specific languages, 460
 - for DSL-based model, 459
 - finding, 93–94
 - to general programming languages, 460
 - between general programming languages and executables, 462
 - and grey-box transformation systems, 503–504
 - inverse, 169
 - for product line requirements, 546–548
 - relating abstractions with, 461
 - in software factory schema, 166, 169
 - parameters of, 485, 502–503
 - solutions to lazy loading of, 473
 - source of, 456
 - synchronizing artifacts with, 172
- markers, 475
- market syndication, 518
- Materials Requirements Planning (MRP) application, 12
- mathematical semantics, 310
- mediators, 138–139
- Message Exchange Patterns (W3C), 450–451
- metadata
 - access to, 398–399, 401
 - defined, 216–217
 - extensive use of, 218
 - integration of, 119–120
 - models as, 216–218
 - standardization of, 120
- metamodels, 289–294, 297–298, 321–327, 483, 484
- meta programming facilities, 506
- meta-tools, 333
- Microsoft BizTalk™ Server, 137
- Microsoft Visual Basic, 476–477

652 Index

- Microsoft Visual Studio
 - intellisense feature in, 504
 - 2005 version, 545–546
- Microsoft Windows, 145
- model-driven architecture
 - (MDA), 567–573
- model-driven development (MDD)
 - applying architectural
 - description to, 254–255
 - automating builds for, 249
 - automating deployment with, 249–251
 - automating patterns with, 238–248
 - automating testing of software with, 251–252
 - and automation, 141
 - basis of, 231–232
 - class modeling with, 236
 - code generation with, 236–237
 - debugging software through, 252–254
 - defining scope of information for, 264–267
 - for domain-specific languages, 142–143 (*see also* modeling languages)
 - essential features of, 232
 - framework completion with, 143–145
 - function of, 140
 - refactoring software with, 248–249
 - software with, 233–237
 - and specialized languages, 139–140
 - transformation of models by, 146–150
- modeling, 226–229, 318–319
- modeling languages, 284–285
 - function of, 212
 - grammar of, 143
 - notation for, 143
 - pattern languages and, 211
 - versus programming languages, 227
 - purposes of, 142–143
 - specification style for, 227–229
 - weak, 116–118
- model mapping
 - defined, 484, 498
 - function of, 484
 - mapping rules in, 498
 - reverse-refinement rule in, 488
 - specifications of, 485
- model mapping problem, 327
- model mapping specification, 485
- model merging, 327
- models
 - as abstractions, 213–214
 - abstract views in, 267–272
 - adding code to, 274–275
 - of artifacts, 267
 - artifacts external to, 243–244
 - of behavior, 266
 - for business applications, 275–276
 - of business process, 520–521
 - capture of build information in, 249
 - capture of deployment information, 249–251
 - capturing intent with, 233–234
 - complementary specification in, 273–274
 - computation-independent views in, 271
 - controls governing interactions in, 265
 - debugging software with, 252–254
 - declarative specification of, 272–273
 - defined, 217–218, 229
 - defining scope of information for, 264–267
 - delegates for, 479
 - as development artifacts, 218–226
 - elements of, 223
 - folder hierarchy of, 224
 - importance of multiple views of, 254

- for integrating incompatible implementation technologies, 232
 - integration context in, 265
 - integrity of, 225–226
 - markers for, 475
 - and metamodels, 483
 - as metadata, 216–218
 - partitioning, 219–221
 - perspective of, 220–221
 - platform independence of views of, 270–271
 - for problem feature, 519–520
 - references to, 221–224
 - removing patterns from, 244
 - reverse-refinement rules for, 488
 - separating concerns with, 213–214
 - serializing into single file, 220
 - for solution domain, 521–523
 - of structure, 266–267
 - subclasses in, 478–479
 - synchronization of, 471–473
 - tracing code to, 473–474
 - used for computation, 142
 - using to automate development (*see* model-driven development (MDD))
 - versions of, 224–226
 - visualization of, 214
 - model synchronization, 471–473
 - model-to-code transformations, 473–476
 - model-to-model transformations, 471–473
 - model transformations
 - continuous synchronization for, 490
 - defined, 484
 - function of, 484
 - implementing, 150
 - pattern-based, 146–148
 - types of, 149–150
 - modularization, 395–396
 - MOF metalanguage, 334, 571–572
 - monolithic construction, 110–115
 - multiplicity, 292
 - mutable software, 88–90, 391
- N**
- name mangling, 479
 - name space, 479
 - namespaces, declaration of, 222–223
 - naming convention, 479
 - naming mechanisms, 345
 - natural languages, 279, 281
 - negotiation, 97–99
 - .NET, 403
 - assemblies in, 225, 226
 - compiling programs in, 60–61
 - example of aspect-oriented programming with, 465–466
 - file references in, 221–222
 - folder hierarchy of, 224
 - pattern languages for, 202
 - NetBeans Metadata Repository, 334
 - .NET Distributed Systems Initiative, 250–251
 - .NET Framework, 426
 - networked applications, 400
 - non-functional requirements, 47–49
 - non-repudiation service feature, 556–557
 - non-terminal symbol production, 286
 - notations, 214, 216, 217, 227–229.
See also concrete syntax
 - Not Invented Here (NIH) syndrome, 114–115, 124
- O**
- Object Constraint Language (OCL), 295–296
 - object identity, 290
 - object orientation, 33–34
 - attacking complexity in, 67–68
 - and automated development, 77–82
 - chronic problems with, 109
 - dealing with change in, 101–108

654 Index

- object orientation, (*cont.*)
 - documenting design in, 73–77
 - function of, 66–67
 - organizing development of, 82–85
 - partitioning responsibility in, 68–73
 - widely used forms of specification in, 131
- Object-Oriented Analysis and Design (OOA&D), 69, 73
- object relational (O/R) mapping framework, 536–537
- object request broker, 402
- objects, 290
- oblique transformations, 149–150, 456, 496
- one-off development, 120–121, 128–129
- ontologies, 449–450
- optimizations, 462, 463, 501–502
- oracle hypothesis, 343
- orchestration, of Web services, 451
- orchestration engine, 137–139
- order management, 518
- organization, of platform-based abstractions, 400
- organizational hypothesis, 343
- organizational structures, 354–355
- Our Simple Language (OSL), 283, 323
- outsourcing, 20–21, 257–258
- OWL (W3C), 450
- ownership associations, 291
- P**
- packaged applications, 18–19
- packages, in UML-based models, 219–220
- paradigm shifts, 32–33
- parameters, 171, 242–243
- parametric generators, 507
- parametric modeling, 507
- parsing, 294, 299, 506
- partial classes, 476–477
- partial derivation, 169
- partial specifications, 476
- partitions, 219–221
- Partner Interface Processes (PIP), 440
- pattern authoring, 73–75
- pattern authors, 239–240
- pattern automation
 - application of, 244–245, 246–248
 - changes in patterns due to, 240–241
 - concerns about, 238
 - engine for, 241–244
 - evaluation of, 245–246, 247
 - keeping track of, 239–240
 - precautions for, 239
 - support for, 238–239
 - tools for users of, 240
 - uses of, 241
- pattern engines, 241–244
- Pattern Language for Evolving Frameworks, 408
- pattern languages, 148
 - characteristics of, 203–204
 - composition rules of, 207
 - defined, 499
 - description of, 201–203
 - for developing software, 204
 - efficiency of, 209–210
 - encapsulating, 205–212
 - examples of, 202
 - formalized, 211
 - formalizing, 210–212
 - versus frameworks, 204–205
 - guidelines of, 202
 - and modeling languages, 211
 - trends in, 203
 - well-formedness of, 207
- pattern readers, 497
- patterns
 - as abstraction mechanisms, 344
 - and abstractions, 499
 - in adaptive design, 105
 - adding to pattern languages, 203
 - application of, 72–73, 244–245, 246–248
 - applied directly to models, 241

- applying, 199
- architectural, 433–437
- automating with models (*see* pattern automation)
- body of, 242
- capture of intent by, 197–198
- concerns for automation of, 238
- context for, 74
- coupled with other patterns, 202
- creating and using, 198–200
- customizing relationships with, 171
- defined, 55, 72, 496
- descriptions of contexts of, 497
- descriptions of problems by, 497
- and development artifacts, 198
- evaluating, 198, 200
- evaluation of, 239, 247
- idempotent, 245
- implementation guidelines of, 202
- importance of, 72
- as informal descriptions of transformations, 498
- literature on, 74
- for model transformation, 146–148
- names of, 205–206
- parameters of, 242–243
- presupplied, 500
- in product families, 204–205
- reapplying, 245
- reevaluating, 200–201, 245
- in refactoring, 108
- reliance on human knowledge, 499–500
- separating evaluation from application of, 245
- as sets of mapping rules, 496–500
- as starting point for defining rules, 499
- and transformations, 499
- used to develop class frameworks, 405–406, 407–408
- weaving into languages, 201–204
- writing, 73–74
 - See also specific patterns*
- pattern specifications, 242–244
- pattern users, 240
- PC revolution, 15
- performance, 48
- personal computers, early
 - application for, 15
- personal computing, platform technology of, 14–16
- perspective, 431
- physical views, 269, 270
- π -calculus, 443
- pilot project team, 183–185
- planned expansion, 422
- platform-based abstractions,
 - 62–63, 423
 - access to metadata of, 398–399, 401
 - based on Web service technology, 397
 - combined with language-based abstractions, 64–65
 - discovery of, 398–399
 - and evolution of solution domain, 65
 - in hardware and software design, 393
 - properties required of, 397–400
- platform-based design, 397–411
- platform-independent protocols, 130
- platform-independent views, 270–271
- platform-specific views, 270
- platform technology, 25–26
 - advancement of, 4–5
 - Batch Era of, 9–11
 - building applications with services, 24–30
 - and challenges for software development, 30–32
 - client server era of, 16–18
 - direct mapping of, 99
 - eCommerce application for, 26–27

656 Index

- platform technology, (*cont.*)
 - evolution of, 8–24
 - growth of packaged applications
 - for, 18–19
 - and Internet, 23
 - with interpreters, 60–61
 - middleware, 23
 - and outsourcing, 20–21
 - personal computing era of, 14–16
 - and stakeholder expectations, 4
 - stitching applications together
 - for, 21–23
 - terminal/host era of, 11–14
 - views independent from,
 - 270–271
- Pluggable Objects pattern, 405
- point cuts, 464–465, 467
- portability, 48
- ports, 424–425
- postconditions, 420
- preconditions, 420
- Predicatable Assembly from
 - Certifiable Components (PACC), 132
- prioritization of features, 339
- problem analysis, 176
- problem descriptions, 74, 95
- problem domain
 - changes in, 100
 - defined, 91
 - description of, 516
 - feature model for, 519–520, 522
 - models of, 365–366
 - scoping, 362–370, 518–521
- problem families, solving, 55–56
- problem feature identification,
 - 363–370
- problem features, 39–40, 366–370
- problems
 - critical innovations for, 125
 - in data integration, 120
 - decomposing, 52–53
 - during deployment of software,
 - 544–545
 - families of, 55–56
 - general types of, 96
 - gratuitous generality, 115–120
 - identifying features of, 363–370
 - immaturity in software
 - development process,
 - 121–125
 - in monolithic construction,
 - 110–115
 - with one-off development,
 - 120–121
 - pattern of, 72
 - patterns in, 204
 - recurrence of, 55
 - solution strategy for, 74
- problem statement, 95
- procedural programming
 - languages, 489–491
- process, conflicting assumptions
 - about, 113
- process assets, 127–128, 157
 - provisioning, 381
- Process Business Components,
 - 436
- process components, 436
- process frameworks
 - active guidance of, 153–154
 - for building data access tier,
 - 150–151
 - constructing, 168
 - constraint-based scheduling in,
 - 152–153
 - defined, 150
 - effective use of, 151
 - and microprocesses, 153
 - proper size of, 153
 - resemblance to software factory
 - schema, 168–169
- process models, 559–560
- processors, for domain-specific
 - languages, 255
- product architecture, 376–378
 - business entity layer of, 544
 - business process layer in,
 - 543–544
 - data layer of, 544
 - design for deployment of,
 - 544–545
 - development of, 526–546
 - framework for activities of, 535

- framework for Web service
 - connectivity in, 539–540
- legacy connector services for, 542
- portal layer in, 543
- summary of, 541–542
- technology mapping in, 542–543
- User Process pattern in, 532–533
- using application patterns of, 528–529
- using business process services for, 533–535
- using development patterns for, 529
- utility services for, 540–541
- product assembly, 389
- product categories, 551
- product deployment, 176–177
- product design, 176
- product development, 350–355
 - activities and artifacts of, 382–389
 - defined, 557–558
 - process definition of, 378
 - with product specification, 558–560
 - summary of, 353–354
- product implementation, 176
- production assets, 127, 157, 347–348, 350–354, 378–379
- product line requirements, 522–523
- product line requirements mapping, 546–548
- product lines. *See* software product lines
- product specification, 176, 558–560
- product specification tools, 184
- product testing, 177
- program families, 126, 128–129
- program fragment, 229
- programming, 318–319
 - evolution of solution domain, 65–66
 - versus modeling, 226–229
 - relative levels of abstraction in, 64–65
 - traceability problem, 40–41
- programming language
 - compilers, 148–150
- programming languages, 284–285
 - and abstraction specificity, 59–60
 - and aspectual decomposition, 259–261
 - composition rules of, 207
 - domain specific (*see* domain-specific languages (DSLs))
 - effects of Design By Contract (DBC) on, 70
 - with embedded query and pattern matching capabilities, 490–491
 - for expression of intent, 234
 - extensions of, 506
 - within framework, 125
 - framework completion for, 143–144
 - increasing abstractness of, 210
 - interface specifications in, 416
 - interpreters for, 59–60
 - meta programming facilities in, 506
 - and model-driven development, 139–140
 - versus modeling languages, 227
 - and multiple platforms, 118–119
 - notation of, 227
 - versus pattern languages, 205
 - procedural, 489–490
 - resolving type references in, 224
 - specification style for, 227–229
 - transformations in, 456
 - well-formedness of, 207
- programs, 229
- progressive abstraction, 49–50
- progressive refinement, 44–46
- progressive transformation, 146
- project references, 224
- Prolog, 489, 491
- protocol design, 442–443
- prototype, 242
- provided interfaces, 424–425

658 Index

Q

query methods, 296

R

Rapid Application Development (RAD), 79–81, 218, 564–567

Rapid Application Development (RAD) environments, 196–197

RDF (Resource Description Framework), 450

realization, 458

reconstruction problem, 41

redevelopment hypothesis, 343

redundancy problem, 90

refactoring, 106–108, 405

automating, 248–249

describing rules of, 495–496

performance of, 501–502

with white-box transformations, 462

refactoring rule, 502

refactoring transformations, 149

references

among models, 221–224

resolving effectiveness of, 225

refinements

as code generation, 473

defined, 44, 456

progressive, 44–46

successive, 44

types of, 458

refinement transformation, 469–470

reflection, 399, 401

relationships, customizing in software factory schema, 170–172

reliability, 47

remote procedure calls (RPC), 17

required interfaces, 424

requirements, 44, 95

analysis and design of, 69–73

capturing, 68–69

defined, 44, 90–91

deriving executables from, 99

versus executables, 99

negotiating for simulations, 97–99

and progressive refinements, 45

role of, 94–95

as specifications, 99

requirements domain, 45–46

requirements gap, 94, 102

reuse, 125–129, 159–161

abstractions for, 395–400

barriers to, 399, 422

and development of class frameworks, 405, 406, 408–411

economics of, 156–157

through process frameworks, 151–152

See also family of problems; software product families

reverse-refinement rules, 487–488

robustness, 47

role composition, 70–71

Round Trip Engineering (RTE), 81–82

rule-based systems, 491

Rule of Three, 408

runtime binding, 477–479

S

scalability, 48

schematic notation, 216

scope and scoping, 345

scope evaluation, 374–375

scope rules, 295–296

security, 47

security policies, 494

self-definition of languages, 311

self-description, 131–132

seller reporting, 518

semantics, 281

for state machines, 284

trace-based, 312–318, 320

translational, 310, 312

of Web service data interchange, 448–450

of Web service interactions, 450–452

- Semantic Web, 449
- semi-formal languages, 281
- sequencing expressions, 441–442
- serialization syntax, 282, 306
- service components, 438–439
 - based on Web service technology, 397
 - interfaces of, 438
- Service-Level Agreements (SLAs), 438, 443–444
- service-oriented architectures (SOA), 25, 397
 - eCommerce application for, 26–27
 - implemented via Web services, 446–448
 - as loosely coupled systems, 445–446
- service-oriented interaction, 539
- simulations
 - choosing mappings for, 96–97
 - domains in, 91–92
 - mapping between domains in, 92–94
 - negotiating requirements for, 97–99
 - requirements as specifications for, 99
 - role of requirements in, 94–95
- SOAP (Simple Object Access Protocol), 448
- software
 - abstract descriptions of, 213–214
 - adaptive design of, 105
 - automated testing of, 251–252
 - automatic refactoring of, 248–249
 - brittleness of, 89–90
 - building, 175–179
 - causes of aging in, 88–90
 - changes in, 88
 - compatibility of, 224–225
 - debugging, 252–254
 - development process for, 178–179
 - disposable, 88, 391
 - documenting design of, 73–77
 - economies of scale in, 159–160
 - economies of scope in, 160
 - enhancement of, 41
 - fatigue of, 89
 - generating with model-driven development, 233
 - immutable, 88, 89
 - increasing abstractness of
 - descriptions of, 210
 - invalidation of, 100
 - levels of specificity in, 257–258
 - maintenance of, 41
 - mass customization of, 188–189
 - modeling (*see* software models)
 - mutable, 88, 391
 - non-functional requirements of, 47–49
 - pattern authoring of, 73–75
 - performance of, 48
 - preventing stagnation of, 101–104
 - product families of, 126
 - product lines of, 126–128
 - product line versus one-off
 - development of, 128–129
 - and raised abstraction levels, 60–61
 - reducing brittleness of, 104–105
 - reducing fatigue of, 105–108
 - redundancy of, 90
 - refactoring, 106–108
 - reliability of, 47
 - requirement negotiations of, 97–98
 - as simulation, 90–99
 - stagnation of, 88–89
 - standardization of, 186
 - supply chains for, 186–187
 - supportability of, 48–49
 - systematic reuse of, 159–161
 - tradeoffs between performance and supportability for, 529
 - unmaintainable, 391
 - usability of, 47
 - using multiple views of models for, 254–264
 - weak packaging of, 111–113

660 Index

- software architectures, 134. *See also* architectures; product architecture
- software components, 396–397, 413–415
 - adaptability of, 422–423
 - building reusable, 6
 - composability of, 423–426
 - systematic reuse of, 125–129
 - See also* component-based development (CBD)
- software developers
 - automation of patterns by, 238–240
 - capturing intent, 196–197
 - challenges for, 7
 - domain-specific assets, 187–188
 - effect of software factories on, 188
 - imbalance in skills of, 5
 - improving skills of, 5
 - intent of, 194
 - leveraging skills of, 5–6, 276
 - during PC revolution, 15
 - rote versus creative work by, 195–196
 - and User Processes, 532–533
- software development
 - abstraction gap in, 41–42
 - abstraction in, 49–50
 - accountability for, 3–4
 - activities and artifacts of, 382–389
 - adapting practices to product lines, 350–355
 - agile, 580–584
 - agility in, 102–104
 - applying abstractions in, 54
 - architecture-driven, 134–137
 - by assembly, 129–139, 185
 - automating, 566–567
 - automating production processes in, 5–6
 - automating rote aspects of, 77–82
 - bottom-up, 178–179
 - of business application software, 584–588
 - challenges for, 30–32
 - choosing systematic approach to, 55
 - of class frameworks, 408
 - completion of, 178
 - component-based (*see* component-based development (CBD))
 - dealing with change in (*see* change)
 - degree of determinism in, 348–350, 352
 - developing abstractions in, 54–56
 - discontinuous innovation of, 32–34
 - discussions of critical innovations in, 7–32
 - for distributed execution environments, 400
 - economics of reuse for, 156–157
 - economies of scale and scope in, 157–159
 - effect of industrialization of, 185–190
 - finding mappings in, 93–94
 - formal process of, 83–84
 - foundations for, 32–33
 - and gratuitous generality, 115–120
 - hand-coded artifacts in, 61
 - immaturity in process of, 121–125
 - industrialization of (*see* software factories)
 - industrializing, 155–161
 - integrating critical innovations in, 161–163
 - levels of specificity in, 257–258
 - model-driven, 139–150
 - models for, 218–226
 - money spent on, 3
 - negotiating requirements for, 97–99

- object-oriented (*see* object orientation)
- organizing, 82–85
- outsourcing, 187
- paradigm shifts in, 32–33
- patterns in, 72–73
- pilot project team for, 183–185
- platform-based abstractions for, 62–65
- and platform technology, 4–5, 8–24
- problems in (*see* problems)
- process frameworks for, 150–154
- process of, 121–125, 178–179
- and product families, 126
- raising level of abstraction in, 53–61
- with Rapid Application Development (RAD), 79–81
- reconstruction problem of, 41
- reduced control in solving problems for, 59–60
- refinement in, 44–46
- requirements as specifications for, 99
- requirements in (*see* requirements)
- results of projects for, 3
- satisfying constraints of, 51–53
- synchronization issues in, 82
- techniques in, 53–61
- traceability problem of, 40–41
- Unified Process (UP) of, 84–85
- variability mechanisms of, 179
- workflow strategy of, 83
- working at wrong level of abstraction in, 41–46, 49–53
- software factories
 - adoption of, 584–588
 - and agility, 580–584
 - building, 174–175
 - building product in, 175–179
 - building software factory template in, 175
 - building software product in, 175–179
 - business case analysis of, 524–526
 - central elements of, 163
 - Collaborative Supply Planning process for, 518
 - collecting feedback for, 175
 - versus computer-aided software engineering, 594–596
 - custom domain-specific languages in, 256
 - and customer relationship management, 187
 - Customer Relationship Management Process for, 517–518
 - defined, 163–164
 - defining scope of information for, 264
 - deployment of, 591–594
 - for domain-specific languages, 328–336
 - effective use of domain-specific languages in, 471
 - eSales process for, 518
 - example of, 180–185
 - foundational ideas of, 190
 - versus 4GLs, 594–596
 - goal for building, 175–176
 - implementation of, 591–594
 - implications of, 185–189
 - integration of developmental innovations with, 161–163
 - and mass customization of software, 188–189
 - versus model-driven architecture (MDA), 567–573
 - versus one-off product development, 179
 - and organizational changes, 188
 - pattern for, 161–162
 - pilot project team for, 183–185
 - problem analysis in, 176
 - product deployment in, 176–177
 - product design in, 176
 - product implementation in, 176

662 Index

- software factories (*cont.*)
 - product line analysis for, 174, 515–526
 - product line design, 174–175
 - product specification in, 176
 - product testing in, 177
 - versus rapid application development, 564–567
 - realizing vision of, 189–190
 - review of building process for, 511–515
 - state of the art of, 588–591
 - and supply chains, 186–187
 - template of, 173–174
 - traceability problem for, 480
 - versus the unified software development process, 573–577
 - use of language-based tools by, 162–163
 - view of activities in, 514
- Software Factory. *See* software factories
- software factory schemas
 - adequate population of, 178
 - versus Architectural Description Standard (ADS), 167–168
 - configuring, 172–173, 177
 - customizing relationships in, 170–172
 - description of, 164–166
 - example of, 513–514
 - figure of, 180
 - formalizing, 498
 - as graphs, 166–169
 - grid for, 164–165
 - importance of, 164
 - relationships between viewpoints in, 169–170
 - resemblance to process framework, 168–169
 - synchronizing artifacts in, 172
 - viewpoints in, 165–166
 - viewpoints of, 168–169
- software factory template, 173–175, 177–178, 183
- software industry, and platform technology, 6–7
- software life cycle, 351, 405
- software models
 - abstract views in, 267–272
 - adding code to, 274–275
 - for business applications, 275–276
 - complementary specifications in, 273–274
 - computation-independent views in, 271–272
 - declarative specification of, 272–273
 - defining scope of information for, 264–267
 - platform independence of views in, 270–271
 - See also* models
- software product families
 - applying software factory schema to, 164–165
 - automation of, 498
 - building members of, 173
 - building online, 512–515
 - common intent of, 197
 - define architectures, 342–344
 - and economies of scope, 348–350
 - enable abstraction, 344–345
 - examples of, 345–347
 - non-functional requirements for, 524, 525
 - patterns in, 204–205
 - process frameworks for, 150–154
 - production assets combined with, 347–348, 350–355
 - in software factory schema, 167–169
 - tradeoffs between performance and supportability for, 529
 - using architecture for, 238–239
- software product lines, 126–128, 347–348, 410
 - analysis of, 174, 361–362, 515–526
 - architecture for, 376, 432–437 (*see also* product architecture)

- asset provisioning for, 548–557
- automating, 379
- benefits of, 145
- building activity for, 553–554
- building business entity service for, 549–553
- building business process service for, 554–557
- business entity layer of, 544
- business process layer in, 543–544
- data layer of, 544
- definition of, 362, 515–518
- deployment of, 391
- design for deployment of, 544–545
- design of, 174–175, 375–379, 526–548
- development of, 350–355, 359–361
- development processes for, 350–354
- establish context and scope, 345
- evolution of, 391
- feature/asset mapping chart for, 547–548
- implementation of, 379–381, 548–557
- versus one-off development, 128–129
- organizational requirements of, 354–355
- organizational structure of, 129
- outsourcing, 187
- platform-based abstractions in, 395–397
- portal layer in, 543
- requirements for, 522–523
- requirements mapping for, 546–548
- scope of, 374
- and supply chains, 186–187
- tradeoffs between performance and supportability for, 529
- using application architecture patterns for, 528–529
- software supply chains, 186–187, 355–357, 542
- solution domain
 - changes in, 100–101
 - effect of platform-based abstractions on, 62
 - evolution of, 65–66
 - feature model for, 521–523
 - and language-based abstractions, 63
 - mappings between, 92–93
 - objects in, 92
 - scoping, 521–524
- solution features, 39–41
- solutions
 - implementation strategy for, 75
 - negotiating requirements for, 97–99
 - prototyping, 98–99
- solutions domain, 94–95, 370–373
- solution strategy, 74–75
- solution variants, 74–75
- source code
 - capture of, 193–194
 - compile-time techniques for, 476–477
 - defined, 229
 - design-time techniques for, 476
 - expressing mapping rule in, 486
 - generating, 78
 - input specification for, 505–506
 - partitioning, 221
 - runtime techniques for, 477–479
 - synchronization of input and output from, 504–505
 - validation of output specifications of, 505
 - visualization of, 76–77
- specialization, 346, 458
- specification model, for catalog manager, 549–550
- specifications
 - of behavior, 420–422
 - benefits to passing around, 506
 - breaking down, 46
 - complementary styles of, 273–274

664 Index

- specifications (*cont.*)
 - declarative, 272–273
 - defined, 208, 233
 - efficiency of descriptions of, 209–210
 - and implementations not separate in classes, 401
 - of invariants, 419–420
 - of model mappings, 485
 - for pattern engines, 242–244
 - of platform-based abstractions, 398
 - for product lines, 558–560
 - strong, 398
 - styles of, 227–229
 - weak, 398
 - specificity, 59–60, 257–258
 - SQL, 229, 272–273
 - stagnation
 - preventing, 101–104
 - problem of, 88–89
 - stakeholder portals, 529–531
 - standardization, 344–345
 - start symbol, 286
 - statecharts, 319
 - state machines, 283–284
 - stock replenishment, 518
 - strategies, 75
 - strong specification, 398
 - structural models, 266–267
 - stylistic notation, 216
 - subclasses, 478
 - successive refinements, 44
 - suppliers, 186–187, 518
 - supply chains, 186–187, 355–357, 542
 - supportability, 48–49
 - synchronization, 172, 471–473, 504–505
 - systematic reuse, 125–129, 159–161
 - abstractions for, 395–400
 - barriers to, 399, 422
 - and development of class frameworks, 405, 406, 408–411
 - economics of, 156–157
 - through process frameworks, 151–152
 - System Definition Model (SDM), 500
 - Systems Integrators (SIs), 126
 - syntactic analysis, 299
 - syntax
 - abstract, 280, 285–298
 - Bakus-Naur form (BNF), 286–287
 - concrete, 281–282, 299–306
 - graphical, 300–306
 - serialization, 282, 306
 - syntax processing tools, 299–300
 - Syntropy, 427
- T**
- technology evolution, 100–101
 - technology mapping, 542–543
 - Template Method pattern, 406
 - testability, 49
 - test case development, 388
 - test harnesses, 232
 - testing, automated, 251–252
 - textual notation, 216, 217, 227
 - THEN parts
 - description of, 485
 - executing mapping rules with, 486
 - implementing with code templates, 491–493
 - throughput, 48
 - tiered architecture, 434–435
 - tool factories, software, 328–336
 - tools
 - for domain-specific language, 255–256
 - traceability, 40–41, 479–481
 - traceability information, 487–489
 - traces, 313–318
 - transformations, 170, 455
 - applied to abstract syntax representations, 457
 - automatic performance of, 501
 - black-box, 501
 - continuous synchronization for, 490

- defined, 456
 - describing, 483–493
 - in general-purpose programming language (GPL), 456
 - generators for, 505–506
 - grey-box, 501
 - horizontal, 456, 462–470, 493–496
 - informal descriptions of, 498
 - interactive performance of, 501
 - manual performance of, 501
 - of models (*see* model transformations)
 - model-to-code, 473–476
 - model-to-model, 471–473
 - oblique, 456, 496
 - problems with, 470–476
 - recalculating, 172
 - solving composition problems of, 476–479
 - solving traceability problem of, 479–481
 - types of, 456–458
 - vertical, 456, 458
 - white-box, 501
 - transformation systems
 - applying black-box transformations in, 505–508
 - code models in, 480
 - defined, 500
 - grey-box, 503–505
 - interactive, 502–503
 - and mapping parameters, 502–503
 - types of performance for, 501
 - translational semantics, 310
 - type checking, 296
 - type system, 416–418, 419–422
- U**
- UDDI, 423
 - UML-based models, 219–221
 - UML Components, 431
 - uncontrolled edits, preventing, 474–475
 - Unified Modeling Language (UML), 117–118, 215, 281, 289, 294
 - Unified Process (UP), 82–83, 573–577
 - description of, 84
 - excessive formalism of, 122–123
 - process framework of, 153–154
 - relationships defined by, 84–85
 - success of, 151
 - views of, 85
 - and visual modeling, 84
 - unmaintainable software, 391
 - upstream product lines, 355
 - usability, 47
 - Use Case Components, 436
 - use cases, 366
 - user delegate, 479
 - user interface, 47
 - User Process pattern, 530–533
 - user subclass, 478
 - utility services, 540–541
- V**
- value chain optimization, 188–189
 - variabilities, 339, 344–345, 365–366
 - binding parameters of, 371–372
 - in class frameworks, 403–404
 - exploited in product lines, 348–350
 - points of, 377–378, 381
 - range of variation, 348–350
 - representing with feature models, 368–370, 372–373
 - variability mechanisms, 179
 - VDM specification language, 421, 427
 - version identifiers, 225
 - vertical transformations, 149, 456, 458–462
 - viewpoints, 136, 137
 - function of, 165–166, 254–255
 - grouped into subsets, 166, 167
 - illustration of, 181–183
 - language of, 169–170
 - mappings between, 183

666 Index

- viewpoints (*cont.*)
 - relationships between, 169–170, 455
 - scope of, 180–181
 - views, 136–137
 - from architectural description, 254–255
 - behavioral, 272
 - computation-independent, 271–272
 - conceptual, 269, 270
 - declarative, 272
 - imperative, 272
 - importance of having multiple, 254
 - levels of abstraction in, 267–272
 - logical, 269–270
 - physical, 269, 270
 - platform independence of, 270–271
 - Visitor pattern, 407
 - visual assembly, 78–80
 - visual modeling, 75–76
 - visual modeling tools, 78
 - vocabularies, 207
- W**
- weak specification, 398
 - weaving, 261–262, 466, 468
 - Web Ontology Language (W3C), 450
 - Web service connectivity, 539–540
 - Web Service Description Language (WSDL), 131–132
 - Web services
 - assembling, 137–138
 - defining, 131–132
 - expressing intent of, 197
 - modeling assemblies of, 265
 - modeling connectivity between, 539–540
 - refactoring rules for, 495–496
 - Web service technology, 25, 446–452
 - data semantics, 448–450
 - examples of, 25
 - platform independent protocols, 130
 - process semantics, 450–452
 - and service-oriented architectures (SOAs), 24
 - used to implement service components, 397
 - Web service viewpoint, 181–182
 - WebSphere Studio (IBM), 334
 - well-formedness, syntactic, 294–295, 330
 - white-box abstraction, 57–58
 - white-box languages, 211
 - white-box mechanisms, 399
 - adapting, 422–423
 - for class frameworks, 406–407
 - used to customize classes, 402
 - See also* customization
 - white-box transformations, 462–463, 501
 - Windows, Forms and Controls for, 145
 - Windows Form Editor, 78
 - wrappers, 422–423
 - WSDL, 417, 448
 - WSDL files, 131–132
- X**
- XLANG, 443
 - XML (eXtensible Markup Language), 306–309
 - effectiveness of, 506
 - mapping abstract syntax graphs to, 307–309
 - protocols in, 130
 - as standard, 23
 - XML-based models, 222–223
 - XML-based technologies, 100–101
 - XML files, 226
 - XML Metadata Interchange (XMI), 572–573
 - XML schemas, 369, 373
 - XQuery, 489
 - XSD type specifications, 417
- Z**
- Z specification language, 421, 427