
INDEX

- Abstract data type, 9
- Abstractions, 85–86, 88
- Acceptance test, 259, 262
- Accessibility, 143
- ActiveRecord, 135
- Agent(s), 72–80
 - metaphor, 73–75
 - systems, 72–74
 - types, 73
 - agent-oriented methods, 74
 - Agent-Oriented Software Engineering (AOSE), 72–74
 - Agent-oriented software life cycle, 73–74
- Aggregation, 114
- Agile Manifesto, 252–253, 264, 268
- Agile methods, 17, 109, 135–136, 145, 249–269
 - Adaptive Software Development (ASD), 251
 - Agile modelling, 251
 - Dynamic Systems Development Method (DSDM), 251
 - eXtreme Programming (XP), 250–251, 254–257, 259–261, 263–269
 - Crystal family, 251, 259
 - Scrum, 251, 259
- Algebraic specifications, 88–89, 97, 100
- Alternative hypothesis, 215–216
- Analysis activities, 85
- Architecture(s), *see* Software architectures
- Architecture slices, 23–24, 28–33, 37–39
- Association(s), 96–97, 105, 112, 114–116, 119, 124
- Binding, 3–18
 - context-aware, 12
 - deployment, 5
 - design time, 4
 - dynamic, 5, 10, 18, 86, 88, 105
 - location-aware, 12
 - run time, 4
 - static, 4, 5, 8, 11, 18
 - time, 4
 - translation time, 4
- Blog systems, 137–138
- Browser, 132–134, 138–143
- Business change, 154
- Business Process Execution Language (BPEL), 153

- CASE tool(s), 106, 268–269
- Cause construct, 206, 220, 226, 230
- CGI (common gateway interface)
applications, 134
- Choreography. *See* Orchestrated-choreography languages
- Class attribute criterion. *See* Coverage criteria
- Class(es), 4, 9–10, 34, 45–66, 86, 88–89, 100–105, 112–115, 122, 155–156, 161–162, 169–173
abstract, 52–54, 63–64
hierarchy of, 10
parent, 4, 9–10
state-based behavior of, 88–89
subclass, 4, 9–10, 53, 60, 63
superclass, 4, 66
generic, 88
- Class Control Flow Graph (CCFG), 100–102, 105
- Client-server, 132, 152–154
- COCOMO model, 35–36
- Collaborative tagging systems, 138
- Commonality, 26–27, 33
- Communicating Sequential Processes (CSP), 93
- Compatibility, 143–144
- Component(s), 3–5, 10–17, 25–41, 45–46, 53, 55, 73–78, 153, 155–156, 159–160, 162, 169, 171
coordination, 13–14, 17
encapsulation of, 73
localization of, 73
off-the-shelf, 7, 10, 16, 23, 27, 32, 37, 42, 48, 73, 157
- Composition(s), 3–17, 28, 29, 32, 35, 89–90, 92–93. *See also* Service(s)
choreographed, 17
mechanisms, 3, 12–16, 89–90, 92
orchestrated, 17
- Compositional product family approach, 23–24, 28–33, 36–40
- Concurrency, 90, 92
- Conjecture, 205, 224
- Contract Net Protocol (CNP), 75
- Coordination relation, 80
- CORBA, 11, 73, 171
- COTS. *See* Software components
- Coupling, 235–236, 240. *See also* View(s)
- Coverage criteria, 92
association-end multiplicity coverage, 96
boundary interior loop coverage, 90–92
class attribute coverage, 96–97
collection coverage, 96
condition coverage, 92, 96
each message on link coverage, 96
full predicate coverage, 96
generalization coverage, 96
message paths coverage, 96
path coverage, 90–91
transition coverage, 90–92
- CVS, 178, 184, 193, 198
- Cyclomatic number, 221, 235, 238–240. *See also* McCabe Cyclomatic Complexity
- Design, 85–88, 100–101, 104–105, 109–112, 127
collaborative, 109
specifications, 85
- Design for change, 7
- Design pattern(s), 43–67, 73
Adapter pattern, 55–57
Model View Controller (MVC), 134–136, 144
Observer pattern, 49–56
Strategy pattern, 59–65
- Diagrammatic specifications, 88
- Dispatcher, 13–14
- Distributed Problem Solving (DPS), 75
- Distributed systems, 3, 14
- Divide-and-conquer approach, 3
- Django. *See* Web application framework(s)
- e-business, 72, 75
- Eclipse, 36
- Effect construct, 206, 220, 226
- Empirical investigation(s), 203–244
in vitro, 207, 215, 228
in vivo, 207–208, 121, 226, 228
replication, 204, 220, 223–227, 230, 234, 244, 246
- Empirical study. *See* Empirical investigation(s)
- Encapsulation, 86, 89
- Endogenous control, 264, 267
- Equivalent scenarios, 89, 100
- e-science, 72, 75
- Event(s), 13–14, 89–90, 92–93
- Evolution. *See* Software evolution
- Evolution of composition
mechanisms, 3–17
- Evolutionary aspects, 177, 198
- Exception handler(s), 88
- Exception(s), 86, 88, 105
- Exogenous control, 264, 267
- Experience Factory (EF), 227

- Experiment/controlled experiment, 207, 215, 217, 226, 232
- Experimental subjects, 207, 215–217, 221–225, 229–230, 236
- Explorative investigation, 213, 233
- Fault(s), 85, 90, 92, 97, 100–101, 104–105, 207, 214, 221
- Finite State Machine(s) (FSMs), 89–91, 97
- Firewalls, 105
- Formal specifications, 88, 97, 109–126
- Foundation for Intelligent Physical Agents (FIPA), 76
- Framework(s), 7, 12, 72–73, 111, 115. *See also* Web Service Resource Framework, Resource Description Framework, Web Service Modeling Framework, *and* Web application framework(s)
- Gaia, 74
- Generalization criterion. *See* Coverage criteria
- Genericity, 86, 88
- Global Coordination Space (GCS), 13–14
- Globus Toolkit 4 (GT4), 76
- Goal Question Metrics (GQM), 228
- Grid
 - Grid agents, 76–77
 - Grid services, 76–77. *See also* Service(s)
 - Grid Services Architecture (GSA), 77
 - Open Grid Service Architecture (OGSA), 72, 75, 77
- GUI. *See* Interface
- Halstead intelligent content, 179, 182–183
- Halstead mental effort, 179, 182–183
- Halstead program difficulty, 179, 182–183
- Heuristic, 205, 230
- Html, 132–135, 142, 144, 185–186
- Http, 131–133, 185–186
- Https, 185–186
- Hypertext, 71
- Hypothesis, 205–206, 215–217, 221, 233
- Identifier, 122
- IEEE standard, 33
- Implementation, 86, 88–89, 94–96, 103
- Information hiding, 8–9, 86–87, 89, 205, 224, 240
- Inheritance, 9, 86, 88, 104–105
- Inheritance dependency, 179, 186
- Initialization, 118, 120–121
- Integration-centric approach, 24–26. *See also* Integration-oriented approach
- Integration-oriented approach, 22, 24–32, 37, 41
- Interface, 22, 55, 98
 - graphical 3, 45, 134, 141
 - of module, 8
 - of service, 6, 8–11, 143
- Interoperability, 16, 39, 72, 76–78, 143–144
- Invariant, 117, 119–124
- IRS-III, 80
- Is-A relationship, 114
- J2EE. *See* Web application framework(s)
- Java RMI. *See* RMI
- JINI, 15–16, 73
- Kiviat diagrams, 180–182, 198
- Knowledge Sources (KSs), 75
- Law, 205–206
- Lean management, 257, 264, 269
- Legacy, 55, 72, 153, 160, 162, 169, 173
- Lifeline, 114
- Link(s), 96
- Maintainability, 143
- Maintenance, 204, 206, 213–216, 228–230, 233, 235, 238–245
- McCabe cyclomatic complexity, 179, 182
- Measure(s), 212, 218, 220–222, 231–237, 240–242
- Message(s), 114–127. *See also* Method(s)
- Metaphors, 261–262
- Method(s), 86, 88–89, 91–104
- Metric(s), 35–36, 177–183, 191, 197, 208, 212, 235. *See also* Measure(s)
 - program complexity, 178–179, 181–183
- Middleware, 11, 13–15, 72–73, 76–77, 152, 171
- Migration, 151–153, 173
- Model transformation, 60, 62
- Model(s), 109–118, 123
 - dynamic, 111–113, 118–120
 - static, 111–113, 116–118
- Model-driven architectures. *See* Software architectures
- Model View Controller (MVC). *See* Design pattern(s)

- Modularity, 73, 78
- Module(s), 8–9, 178–183, 192–198
- Multi-agent applications, 73
- Multi-agent architectures. *See* Software architectures

- Networked systems, 73–75
- Nonparametric tests, 217
- Null hypothesis, 215–217

- Object Constraint Language (OCL), 92
- Object(s), 86, 88–90, 92–101, 113–114, 119–120, 122, 124, 132, 135–136, 140
- Object-oriented, 152–153, 162
 - design, 9, 86–87, 105
 - development, 43, 67
 - languages, 4, 9–10, 88, 135–136
 - software, 77
- Object-relational mapping, 136
- Observation(s), 204–205, 207, 213, 218–219
- Off-the-shelf components. *See* Component(s)
- Ontology, 78–80. *See also* Semantic Web
- Open source software, 23, 27–28, 35–37, 41
- Open-world software, 5
- Operational relationship, 80
- Optimization of test case execution, 86
- Oracles, 86, 89, 97, 100
 - automatic generation of, 89, 97
- Orchestration-choreography languages, 78
- OSGI, 15
- Outsourcing, 71
- Overengineered Architecture, 27

- Package(s), 8, 29, 35–36, 57, 63, 65.
 - See also* Unified modeling language
- Pair programming, 257–258, 267
- Parametric tests, 217
- Parnas Law, 205–206
- Parnas Theory, 221
- PHP, 133–134, 141
- Platform(s), 21–31, 36–41, 48–49, 71–72, 76–78. *See also* Software product lines
- Polymorphism, 10, 86, 105
- Post-condition, 115, 117, 119–122, 124
- Postmortem Investigation, 207–208
- Pre-condition, 118–121, 124
- Principle, 205–206
- Process calculus(i), 111, 115, 127
- Process(es), 21–25, 34, 38, 40–41, 78, 80.
 - See also* Software process
- Program understanding, 197

- Project investigation, 207, 212–213, 228, 233
- Publish-subscribe systems, 13. *See also* Software architectures

- Quality
 - assessment and improvements, 38
 - management, 38
 - requirements, 38

- Reengineering, 234, 241–243
- Refactoring, 60, 62, 183, 194, 257–258, 267
- Reference architecture(s). *See* Software architectures
- Reflection, 125
- Reliability, 143
- Remote object coordination, 76
- Remote procedure call (RPC), 140
- Requirements, 5–8, 12–17, 23–24, 26–27, 29, 31, 36–38, 40–41
 - analysis, 88
 - management, 36–38, 260–263
 - specifications, 85
- Resource Description Framework (RDF), 144
- Restoration, 234, 239–243
- Retrospective Investigation. *See* Postmortem Investigation
- Reuse, 21, 23, 73, 86, 103
- Reverse engineering, 197–198, 234, 236–239
- RMI, 11
- Ruby, 135, 141. *See also* Web application framework(s)
- Ruby-on-Rails. *See* Web application framework(s)

- Scaffolding, 86, 97
- Scalability, 142–143
- Security, 139, 141, 143
- Semantic Web, 77–80, 144–146
 - Semantic Grid, 77
 - Semantic Web Service, 77, 80–81
 - Web Ontology Language (OWL), 77, 79
 - Web Service Modeling Ontology (WSMO), 79–80
- Separation of concerns, 8, 151
- Service(s), 12, 14–18, 71–72, 74–80.
 - See also* Software components broker, 14–15
 - composition, 17, 75

- consumer, 14–16
- coordination, 75
- discovery, 14–16, 79
- invocation, 17
- orchestration, 17, 78–79
- provider, 14–17
- Service-Oriented Architecture (SOA), 14–17, 71–81, 153–154, 157, 160
- Service-oriented computing, 71–72
- Service-oriented middleware, 76
- service-oriented software development methods, 74
- Simple Object Access Protocol (SOAP), 16, 155, 163
- Software architecture(s), 3–17, 73, 77. *See also* Web Service(s)
 - Model-driven architectures, 72, 76–77
 - multi-agent architectures, 73
 - multi-tier, 4
 - peer-to-peer, 4
 - product line architecture, 21. *See also* Software product lines
 - publish-subscribe, 13
 - reference architectures, 29, 72
 - Service-Oriented Architectures (SOA). *See* Service(s)
 - tuple-space, 13
- Software components. *See* Component(s)
- Software composition. *See* Composition(s)
- Software costs, 152
- Software development process. *See* Software process
- Software evolution, 24, 26, 28, 30, 33, 37, 40
 - analysis, 177–198
- Software process, 3–7, 10–11, 85, 110, 128, 207, 221, 224, 227. *See also* Agile method(s)
 - agile, 4, 17
 - evolutionary, 109–110
 - iterative, 249
 - sequential, 4–6
 - spiral, 249, 263
 - waterfall, 6–7, 249, 263, 268
- Software product families, 22, 30
- Software product lines, 21–23, 28, 39
- Software visualization, 177–180, 184–198
- Specializations, 96
- Specification, 116–120, 123, 126–127
- State(s), 86, 88–93, 97–103
 - compound, 90
 - final, 89, 91–92
 - initial, 89, 91
- Statecharts, 89–93, 97
- State-dependent behavior, 86, 100, 105
- Structured Query Language (SQL), 135
- Struts. *See* Web application framework(s)
- Stub(s), 95, 167–168, 170–172
- Subscription, 13–14
- Survey, 207, 210, 233, 236
- Tag, 138, 142
- Tagging, 138, 143
- Temporal logic specifications, 143
- Test activities, 85
- Test case(s), 86, 88–100, 103–105
 - automatic generation of, 89, 97
 - integration, 104
 - reuse, 86
- Test suites, 90–92, 97, 100, 103–105
- Test-driven development, 257, 267
- Testing
 - code-based, 100
 - Data flow (DF), 86, 100, 103
 - integration, 32–35, 106. *See also* Test case(s)
 - interclass, 88, 92–97, 102, 105
 - intraclass, 88–92
 - object-oriented, 85–106
 - regression, 104–105
 - specification-based techniques, 88–89, 100
 - structural, 90, 100–103
- Theory, 204–206, 208, 213, 218–231
- Transition(s), 89–93
- Tropos, 74
- Tuple-space systems, 14. *See also* Software architectures
- Unified Modeling Language (UML), 88–89, 92–93, 111–115, 118, 123, 126–128
 - activity diagrams, 93
 - class diagrams, 4, 45, 49–50, 55, 60, 62–63, 77, 88, 92, 96–97, 104, 113, 117
 - collaboration diagrams, 93, 95–96
 - object diagrams, 92, 96
 - package diagrams, 93
 - sequence diagrams, 49, 51, 55–56, 88, 92–97, 114–115, 118–120, 122, 124, 126–127
 - UML sec, 127
 - use case, 93

- Unified Process (UP), 109
- Uniform Resource Identifier (URI), 144
- Universal Description, Discovery and Integration (UDDI), 154
- URL, 132–135, 144
- Usability, 143
- User stories, 256, 261–262, 268–269
- Uses dependency, 179, 181

- Validity
 - conclusion, 218
 - construct, 219, 225
 - external, 220, 226
 - internal, 219
 - risks and threats, 204, 218–220
- Variable(s), 4, 10, 88–92, 95, 98, 100–103
 - dynamic type of, 10
 - static type of, 10
 - type, 88, 116
 - visibility, 100
- Verification, 18
 - and validation, 18, 25
- Versioning, 178, 184
- View(s), 88, 112–113, 177–179, 181–197
 - change coupling, 177, 192–198
 - developer contribution, 188–191, 198
 - feature evolution, 177, 183–185, 198
 - fractal, 188–192
 - multiple evolution metrics, 177–181, 198
 - polymetric, 180–181, 197
- Visibility, 114, 118. *See also* Variable(s)
 - private, 114, 118
 - public, 114, 118
- Visualization. *See* Software visualization

- Web, 71, 131–145
- Web application(s), 133–144
- Web application framework(s), 135–136, 144
 - Django, 136
 - J2EE, 136
 - Ruby-on-Rails, 135
 - Struts, 136
- Web engineering, 135, 138, 140, 143–145
- Web service(s), 15–16, 76, 139–140, 143, 151–174. *See also* Service(s)
 - mining, 160–161
 - recovering, 159–160
 - Web Service Architecture (WSA), 72, 75–76, 78, 172
 - Web Service Modeling Framework (WSFM), 79
 - Web Service Modeling Ontology (WSMO), 79–80
 - Web Service Resource Framework (WSRF), 77
 - Web Services Description Language (WSDL), 77–78
- Web Service Definition Language (WSDL), 155, 160, 163
- Wide-Mouthed Frog protocol, 125
- Wiki systems, 137
- Workflow Management System (WfMS), 78
- Wrapper, 164–168, 173
- Wrapping, 159, 163–173

- XML, 16, 78, 133, 137, 140, 155, 163–173
- XP development process, 263–264. *See also* Agile methods