

# Bibliography

- [1] F. B. Abreu and R. Carapuca. Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software*, 26(1):87–96, Jan. 1994.
- [2] V. R. Basili. *Tutorial on models and metrics for software management and engineering*. IEEE Press, 1980.
- [3] V. R. Basili, L. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, Oct. 1996.
- [4] V. R. Basili and A. Turner. Iterative enhancement, a practical technique for software development. *IEEE Transactions on Software Engineering*, SE-1(4), Dec. 1975.
- [5] V. R. Basili and D. M. Weiss. Evaluation of a software requirements document by analysis of change data. In *5th Int. Conf. on Software Engineering*, pages 314–323. IEEE, 1981.
- [6] L. Bass, P. Clements, and Rick Kazman. *Software Architecture in Practice, Second Edition*. Addison-Wesley Professional, 2003.
- [7] K. Beck. *Extreme Programming Explained*. Addison-Wesley, 2000.
- [8] K. Beck. *Test Driven Development: by Example*. Addison-Wesley Professional, 2002.
- [9] R.V. Binder. *Testing Object-Oriented Systems—Model, Patterns, and Tools*. Addison-Wesley, 1999.
- [10] B. Boehm. Software engineering. *IEEE Transactions on Computers*, 25(12), Dec. 1976.

- 
- [11] B. Boehm. *Tutorial: software risk management*. IEEE Computer Society, 1989.
  - [12] B. W. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.
  - [13] B. W. Boehm. Software engineering economics. *IEEE Transactions on Software Engineering*, 10(1):135–152, Jan. 1984.
  - [14] B. W. Boehm. Improving software productivity. *IEEE Computer*, pages 43–57, Sept. 1987.
  - [15] G. Booch. *Object-Oriented Analysis and Design*. The Benjamin/Cummings Publishing Company, 1994.
  - [16] F. Brooks. *The Mythical Man Month*. Addison-Wesley, 1975.
  - [17] N. Brown. Industrial-strength management strategies. *IEEE Software*, July 1996.
  - [18] R.N. Charette. *Software Engineering Risk Analysis and Management*. McGraw Hill, 1989.
  - [19] R.N. Charette. Large-scale project management is risk management. *IEEE Software*, July 1996.
  - [20] E. Chen. Program complexity and programmer productivity. *IEEE Transactions on Software Engineering*, SE-4:187–194, May 1978.
  - [21] S. R. Chidamber and C. F. Kemerer. A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.
  - [22] T.S. Chow. Testing software design modeled by finite state machines. *IEEE Transactions on Software Engineering*, SE-4(3):178–187, 1978.
  - [23] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2003.
  - [24] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2001.
  - [25] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–443, 1997.
  - [26] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. *Software Engineering Metrics and Models*. The Benjamin/Cummings Publishing Company, 1986.

- 
- [27] J. S. Davis. Identification of errors in software requirements through use of automated requirements tools. *Information and Software Technology*, 31(9):472–476, Nov. 1989.
- [28] T. DeMarco. *Structured Analysis and System Specification*. Yourdon Press, 1979.
- [29] L. Dobrica and E. Niemela. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.
- [30] J. Eder, G. Kappel, and M. Schrefl. Coupling and cohesion in object-oriented systems. Technical report, University of Klagenfurt, 1994.
- [31] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM System Journal*, (3):182–211, 1976.
- [32] M. E. Fagan. Advances in software inspections. *IEEE Transactions on Software Engineering*, 12(7):744–751, July 1986.
- [33] W. Farr. Software reliability modeling survey. In M. R. Lyu, editor, *Software Reliability Engineering*, pages 71–117. McGraw Hill and IEEE Computer Society, 1996.
- [34] S. I. Feldman. Make—a program for maintaining computer programs. *Software Practice and Experience*, 9(3):255–265, March 1979.
- [35] M. Fowler. *UML Distilled—A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, 2003.
- [36] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.
- [37] D. P. Freedman and G. M. Weinberg. *Handbook of Walkthroughs, Inspections, and Technical Reviews—Evaluating Programs, Projects, and Products*. Dorset House, 1990.
- [38] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns—Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.
- [39] T. Gilb and D. Graham. *Software Inspection*. Addison-Wesley, 1993.
- [40] H. Gomma and D. B. H. Scott. Prototyping as a tool in the specification of user requirements. In *Fifth Int. Conf. on Software Engineering*, pages 333–341, 1981.
- [41] J. Goodenough and S. L. Gerhart. Towards a theory of test data selection. *IEEE Transactions on Software Engineering*, SE-1:156–173, 1975.

- 
- [42] S. E. Goodman and S. T. Hedetniemi. *Introduction to the Design and Analysis of Algorithms*. McGraw-Hill, 1977.
  - [43] R. Grady and D. Caswell. *Software Metrics: Establishing a Company-wide Program*. Prentice Hall, 1987.
  - [44] R. B. Grady and T. V. Slack. Key lessons learned in achieving widespread inspection use. *IEEE Software*, pages 48–57, July 1994.
  - [45] E.M. Hall. *Managing Risk: Methods for Software Development and Enhancement*. Addison-Wesley, 1998.
  - [46] M. Halstead. *Elements of Software Science*. Elsevier North-Holland, 1977.
  - [47] W. Harrison, K. Magel, R. Kluczny, and A. DeKock. Applying software complexity metrics to program maintenance. *IEEE Computer*, pages 65–79, Sept. 1982.
  - [48] S. Henry and D. Kafura. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, 7(5):510–518, 1981.
  - [49] S. Henry and D. Kafura. The evaluation of software systems’ structures using quantitative software metrics. *Software Practice and Experience*, 14(6):561–573, June 1984.
  - [50] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(3):335–355, 1969.
  - [51] IBM-Rational. Rational unified process best practices for software development teams. Technical report, IBMwebsite, 1993.
  - [52] IEEE. IEEE standard glossary of software engineering terminology. Technical report, 1990.
  - [53] IEEE. IEEE recommended practice for software requirements specifications. Technical report, 1998.
  - [54] IEEE. IEEE recommended practice for architectural description of software-intensive systems. Technical Report 1471-2000, 2000.
  - [55] International Standards Organization. Software engineering—product quality. part 1: Quality model. Technical Report ISO9126-1, 2001.
  - [56] I. Jacobson. *Object-oriented Software Engineering—A Use Case Driven Approach*. Addison-Wesley, 1992.
  - [57] P. Jalote. *CMM in Practice—Processes for Executing Software Projects at Infosys*. Addison-Wesley, 1999.
  - [58] P. Jalote. *Software Project Management in Practice*. Addison-Wesley, 2002.

- 
- [59] P. Jalote, A. Palit, and P. Kurien. The timeboxing process model for iterative software development. In *Advances in Computers, Vol. 62*, pages 67–103. Academic Press, 2004.
- [60] P. Jalote, A. Palit, P. Kurien, and V. T. Peethamber. Timeboxing: A process model for iterative software development. *The Journal of Systems and Software*, 70:117–127.
- [61] S.H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley, 1995.
- [62] T. Korson and J. D. Gregor. Understanding object-oriented: A unifying paradigm. *Communications of the ACM*, 33(9):40–60, Sept. 1990.
- [63] P. Kruchten. *The Rational Unified Process*. Addison-Wesley, 1999.
- [64] W. Lie and S. Henry. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 23(2):111–122, 1993.
- [65] B. Liskov. Data abstraction and hierarchy. *SIGPLAN Notices*, 23(5), May 1988.
- [66] B. Meyer. *Object Oriented Software Construction*. Prentice Hall, 1988.
- [67] J. D. Musa, A. Iannino, and K. Okumoto. *Software Reliability—Measurement, Prediction, Application*. McGraw Hill, 1987.
- [68] G. Myers. *The Art of Software Testing*. Wiley-Interscience, New York, 1979.
- [69] J. Offutt, S. Liu, A. Abdurazik, and P. Ammann. Generating test data from state-based specifications. *The Journal of Software Testing, Verification, and Reliability*, 13(1):25–53, March 2003.
- [70] M.S. Phadke. Planning efficient software tests. *Crosstalk*, Oct 1997.
- [71] L. H. Putnam. A general empirical solution to the macro software sizing and estimation problem. *IEEE Transactions on Software Engineering*, SE-4:345–361, July 1978.
- [72] L. H. Putnam and W. Myers. *Industrial Strength Software: Effective Management Using Measurement*. IEEE Computer Society, 1997.
- [73] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, 11(4):367–375, Apr. 1985.
- [74] W. W. Royce. Managing the development of large software systems. In *Proc. 9th Int. Conf. on Software Engineering (ICSE-9); originally in IEEE Wescon, Aug 1970*, pages 328–338. IEEE, 1987.

- [75] SEI (Software Engineering Institute). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.
- [76] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
- [77] M. D. Smith and D. J. Robson. Object oriented programming: The problems of validation. *Proc. of 6th Int. IEEE Conference on Software Maintenance*, pages 272–282, Nov. 1990.
- [78] M. D. Smith and D. J. Robson. A framework for testing object-oriented programs. *Journal of Object-Oriented Programming*, pages 45–53, June 1992.
- [79] W. P. Stevens, G. J. Myers, and L. Constantine. Structured design. *IBM Systems Journal*, 13(2), 1974.
- [80] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications, Second Edition*. Wiley-Interscience, 2002.
- [81] C. Watson and C. Felix. A method of programming measurement and estimation. *IBM Systems Journal*, 16(1), Jan. 1977.
- [82] G. M. Weinberg and E. L. Schulman. Goals and performance in computer programming. *Human Factors*, 16(1):70–77, 1974.
- [83] E. F. Weller. Lessons learned from three years of inspection data. *IEEE Software*, pages 38–53, Sept. 1993.
- [84] N. Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, April 1971.
- [85] M. Woodward, M. Hennell, and D. Hedley. A measure of control flow complexity in program text. *IEEE Transactions on Software Engineering*, SE-5:45–50, Jan. 1979.
- [86] R. T. Yeh and P. Zave. Specifying software requirements. *Proceedings of the IEEE*, 68(9):1077–1088, Sept. 1980.
- [87] B. H. Yin and J. W. Winchester. The establishment and use of measures to evaluate the quality of designs. *Software Engineering Notes*, 3:45–52, 1978.
- [88] E. Yourdon and L. Constantine. *Structured Design*. Prentice Hall, 1979.
- [89] W. M. Zage and D. M. Zage. Evaluating design metrics on large-scale software. *IEEE Software*, pages 75–81, July 1993.

# Index

- Acceptance testing, 230
- Activity diagram, 155
- Adaptive maintenance, 5
- Aggregation, 145, 151
- Agile, 28
- Algorithm design, 169
- Analysis, 39, 58, 66
  - problem partitioning, 58
  - projection, 58
  - transition to specification, 41
- Architecture, *see* Software architecture
- Architecture description language, 117
  
- Base class, 145
- Black-box testing, 236
  - boundary value analysis, 239
  - equivalence class partitioning, 237
  - error guessing, 243
  - pairwise testing, 240
  - state-based testing, *see* State-based testing
- Boundary value analysis, 239
  - an example, 240
- Branch testing, 249
- Bug, *see* Fault, Error
- Build process, 200
  
- Central transforms, 137
- Chief programmer team, 89
- Class, 143, 144
  
- Class diagram, 148
- Client-server interaction of objects, 144
- Client-server style, 112
- COCOMO, 71
  - distribution with phases, 73
  - effort multipliers, 72
  - schedule distribution, 77
  - schedule estimation, 76
- Code inspection, *see* Inspection process
- Coding, 181
  - incremental coding, 194
  - pair programming, 197
  - process, 194
  - refactoring, *see* Refactoring
  - test-driven development, 195
  - top-down approach, 194
- Coding standards, 191
  - commenting and layout, 192
  - conventions for statements, 192
  - conventions on files, 192
  - naming conventions, 191
- Cohesion, 126, 128
- Collaboration diagram, 153
- Combinatorial testing, 241
- Commenting, 192
- Communicating processes style, 114
- Complexity metrics, 216
- Component and connector view, 101, 119
  - blackboard style, 110

- client-server style, 112
- communicating processes style, 114
- components, 101
- connectors, 103
- n-tier structure, 113
- object-oriented style, 114
- peer-to-peer style, 114
- pipe-and-filter style, 108
- publish-subscribe style, 113
- shared-data style, 110
- Configuration management, *see* Software configuration management
- Context diagram, 61
- Control flow graph, 216, 217, 248
- Control flow-based testing
  - criteria, 248
- Corrective maintenance, 5
- Correctness of design, 122
- Cost estimation, *see* Estimation
- Coupling, 123, 125
- Coverage analysis, 251
- Cross referencing of requirements, 66
- Cyclomatic complexity, 216
  
- Data dictionary, 61
- Data flow diagram, 59, 135, 161
  - conventions, 59
  - example, 59
  - leveled, 61
- Debugging, 204
- Defect, 5
  - life cycle, 235
  - logging and tracking, 235
- Defect injection and removal cycle, 78
- Defect removal efficiency, 255
- Defects, 86
- Depth of inheritance tree, 175
- Design, 121
- Design constraints, 44
- Design methodology, 122
- Design metrics, 172
  - for OOD, *see* OOD Metrics
  - graph impurity, 173
  - identifying error-prone modules, 174
  - information flow metrics, 173
  - network metrics, 173
- Design principles, 122
- Design review, 172
- Design verification, 171
  
- Detailed design, 168
  - algorithm design, 169
  - state modeling of classes, 170
  - stepwise refinement, 169
- Development process, 12, 13
  - iterative development, *see* Iterative development
  - iterative enhancement model, 19
  - prototyping, *see* Prototyping
  - timeboxing model, *see* Timeboxing model
  - waterfall model, *see* Waterfall model
- Dynamic binding, 147
- Dynamic modeling, 159
  
- Efficiency, 4
- Effort, 86
- Effort estimation, *see* Estimation
- Encapsulation, 143
- Equivalence class partitioning, 237
  - an example, 238
  - selecting test cases, 238
- ER diagrams, 61
- Error, 225, *see also* Defect
- Error guessing, 243
- Estimation, 70, 91
  - a bottom-up approach example, 74
  - bottom-up approach, 74
  - COCOMO, *see* COCOMO
  - single variable models, 71
  - size estimation, 73
  - top-down estimation model, 71
- External interface requirements, 45
- Extreme programming, 28, 197, 201
  
- Factoring, 138
- Failure, 227
- Fault, *see also* Error, Defect
  - multi-mode, 241
  - single-mode, 241
- Fault tolerance, 45
- Finite state automata, 170
- Function points, 71
- Functional modeling, 160
- Functional requirements, 43
- Functional testing, *see* Black-box testing
- Functionality, 4
  
- Graph impurity metric, 173



- Halstead's measures, 215, 219
- Incremental development, 194, 204
  - refactoring, *see* Refactoring
  - test-driven development, 195
- Industrial-strength software, 7
- Inflow of a module, 174
- Information flow metrics, 173
- Information hiding, 186, 187
- Inheritance, 145
  - base class, 145
  - class hierarchy, 145
  - multiple, 146
  - nonstrict, 146
  - strict, 146
  - subclass, 145
  - superclass, 145
- Inspection process, 210
  - defect log, 214
  - group review meeting, 212
  - moderator, 211
  - planning, 211
  - self-preparation log, 212
  - self-review, 212
  - summary report, 214
  - summary report example, 214
- Integration testing, 229
- Interaction diagram, 151
- Internal documentation of programs, 192
- Iterative development, 34
  - iterative enhancement model, 19
  - timeboxing model, *see* Timeboxing model
- Iterative enhancement model, 19
- Knot count, 220
- Leveled data flow diagram, 61
- Levels of testing, 229
- Life cycle of a defect, 235
- Liskov substitution principle, 131
- Live variables, 219
- Logic design of modules, 169
- Maintainability, 4
- Maintenance, 5
  - adaptive, 5
  - corrective, 5
  - costs, 5
- Manpower ramp-up in a project, 77
- Mean time to failure, 253
- Measurements
  - defects, 86
  - effort, 86
  - size, 87
- Metrics, 214
  - complexity measures, 216
  - cyclomatic complexity, 216
  - defect removal efficiency, 255
  - for design, *see* Design metrics
  - for object-oriented design, *see* Metrics for OOD
  - graph impurity, 173
  - Halstead's, 215
  - information flow metrics, 173
  - knot count, 220
  - live variables, 219
  - network metrics, 173
  - reliability, 253
  - size measures, *see* Size
  - span, 220
  - topological complexity, 221
- Metrics for OOD, 175, 178
  - coupling between classes, 176
  - depth of inheritance tree, 175
  - response for a class, 176
  - weighted methods per class, 175
- Modularity, 122, 177
- Module size, 188
- Monitoring and control, 92
- Most abstract inputs, 136
- Most abstract outputs, 136
- Multi-mode faults, 241
- Multiple inheritance, 146
- Nesting of constructs, 188
- Network metrics, 173
- Nonfunctional requirements, 43
- Object, 143
  - behavior, 143
  - interface, 143
  - relationships between, 144
  - state, 143
- Object modeling technique, 156
- Object-oriented analysis, 157
- Object-oriented design, 142
  - aggregation, 145

- association between objects, 144
- class hierarchy, 145
- cohesion, 128
- coupling, 125
- defining internal classes and operations, 161
- design patterns, 142
- dynamic modeling, 159
- functional modeling, 160
- identifying associations, 158
- identifying attributes, 157
- Liskov substitution principle, 131
- methodology, 156
- metrics, *see* Metrics for OOD
- open-closed principle, 129
- rate of returns example, 165
- state modeling of classes, 170
- UML, *see* UML
- word counting example, 162
- Open-closed principle, 129
- Outflow of a module, 174
  
- Pair programming, 197
- Pairwise testing, 240, 241
  - an example, 242
  - generating test cases, 242
  - objective, 242
- Path testing, 250
- Peak team size, 77
- Peer-to-peer style, 114
- Performance requirements, 44
- Pipe and filter style, 108
- Polymorphism, 145, 146
- Portability, 4
- Post-condition of programs, 184
- Postmortem analysis, 33
- Pre-condition of programs, 184
- Problem analysis, *see* Analysis
- Problem partitioning, 58
- Process management, 12
  - software engineering process group, 12
- Process model, 13
- Productivity, 1, 3
- Programming practices, 182, 187
  - checking read return value, 189
  - coding standards, *see* Coding standards
  - control constructs, 187
  - correlated parameters, 190
  - empty if, while, 189
  - importance of exceptions, 190
  - information hiding, 187
  - nesting of constructs, 188
  - return from finally block, 189
  - robustness, 188
  - side effects, 188
  - switch case with default, 189
  - trusted data sources, 190
  - use of gotos, 187
  - use of user-defined types, 187
- Programming principles, 182
- Project management, 6, 12, 32, 35
  - cost estimation, *see* Estimation
  - monitoring and control, *see* Project monitoring and control
  - planning, *see* Project planning, *see* Project planning
  - postmortem analysis, 33
  - process, 32
  - project monitoring, 33
  - project tracking, 33
  - quality, *see* Quality plan
  - relationship with development process, 33
  - risk management, *see* Risk management
  - scheduling, *see* Project scheduling
- Project monitoring and control, 33, 86
  - activity-level monitoring, 87
  - measurements, 86
  - milestone analysis, 87
  - status reports, 87
- Project planning, 32, 69
  - chief programmer team, 89
  - effort estimation, *see* Estimation
  - manpower ramp-up, 77
  - peak team size, 77
  - quality plan, 80
  - risk management, *see* Risk management
  - schedule, *see* Project scheduling
  - team structure, 89
- Project scheduling, 76, 91
  - detailed schedule, 88
  - distribution among different phases, 77
  - estimation, 76
  - flexibility, 76

- Microsoft project, 89
- milestones, 77
- overall schedule estimation, 76
- square root check, 77
- Project tracking, 33
- Projection, 58
- Prototyping, 17, 34
  - benefits, 19
  - cost-cutting methods, 18
  - process, 17
  - throwaway, 17
- Psychology of testing, 228
- Publish-subscribe style, 113
  
- Quality, 3, 39, 225
  - defect injection and removal cycle, 78
- Quality and productivity, Q&P, 9
- Quality attributes, 3, 4
- Quality plan, 80
  
- Rational Unified Process, 22
- Rayleigh curve, 77
- Refactoring, 200
  - bad smells, 202
  - impact on design, 202
  - objective, 201
  - risk mitigation, 201
  - test suite, 202
- Regression testing, 230
- Reliability, 4, 253
  - definition, 253
  - failure intensity, 254
  - mean time to failure, 253
- Requirement analysis, *see* Analysis
- Requirement change, 6
- Requirement specification, 37, 41
  - components, 43
  - design constraints, 44
  - desired characteristics, 41
  - document structure, 46
  - external interface requirements, 45
  - functional requirements, 43
  - IEEE standards, 46
  - performance requirements, 44
  - specification language, 46
  - use cases, *see* Use cases
- Requirement validation, 63
- Requirements, *see* Software requirements
- Requirements review, 65
  
- Reviews, *see* Inspection process
- Risk management, 80, 92
  - a practical approach, 84
  - an example, 85
  - checklists of frequently occurring risks, 82
  - risk, 80
  - risk analysis, 83
  - risk assessment, 81
  - risk control, 83
  - risk exposure, 83
  - risk identification, 81
  - risk monitoring, 84
  - risk prioritization, 83, 84
  - top 10 risks, 82
- Rule of composition, 184
- RUP, *see* Rational Unified Process
  
- Scenario, 50
- Scenarios for modeling, 159
- Schedule, 3, 86
- Security, 45
- Sequence diagram, 151
- Shared-data style, 110
- Side effects, 188
- Single-entry, single-exit constructs, 185
- Single-mode faults, 241
- Size, 3, 71, 87, 215
  - halstead’s measure, 215
  - lines of code (LOC), 1, 3, 215
- Size estimation, 73
- Software
  - costs, 3
  - industrial-strength software, 2
  - productivity, 3
  - size, *see* Size
  - student system, 1
- Software architecture, 95, 119
  - allocation view, 99
  - analysis, *see* Software architecture evaluation
  - architectural styles, 108
  - behavior description, 116
  - blackboard style, 110
  - client-server style, 112
  - combining views, 117
  - communicating processes style, 114
  - component and connector view, *see* Component and connector view

- definition, 96
- deployment view, 154
- description, 96
- designing, 119
- documenting, 114
- element catalog, 115
- example, 104
- layered style, 113
- module view, 99, 121
- object-oriented style, 114
- peer-to-peer style, 114
- pipe-and-filter style, 108
- primary view, 115
- publish-subscribe style, 113
- quality attributes, 118
- relationship among views, 99
- relationship between elements, 98
- shared-data style, 110
- stakeholders, 115
- system context, 115
- views, 98, 99, 119
- Software architecture evaluation, 118
  - approaches, 118
  - quality attributes, 118
- Software configuration management, 12, 198
- Software engineering
  - definition, 9
  - problem domain, 7
- Software engineering process group, 12
- Software inspections, *see* Inspection process
- Software process, 10–12
  - components, 11
  - development process, *see* Development process
  - inspection process, *see* Inspection process
  - management, 12
  - nonengineering processes, 11
  - product engineering processes, 12
  - project management process, *see* Project management
- Software quality, *see* Quality
- Software reliability, *see* Reliability
- Software requirement process, 39
- Software requirement specification, *see* Requirement specification
- Software requirements, 37, 41, 66
  - analysis, *see* Analysis
  - consistent, 42
  - data flow diagram, 59
  - definition, 37
  - ER diagrams, 61
  - error types, 64
  - errors, 64
  - impact on development cost, 39
  - impact on quality, 39
  - nonfunctional requirements, 43
  - process, 39
  - specification, *see* Requirement specification
  - use cases, *see* Use cases
  - validation, *see* Requirement validation
  - verifiable, 42
- Software size, *see* Size
- Source code control, 198
  - checkout, 199
  - command types, 199
  - conflicts, 199
  - tools, 198
  - version maintenance, 200
- Specification language, 46
- Standards compliance, 45
- State diagram, 155, 170
- State modeling of classes, 170
  - an example, 171
- State-based testing, 244
  - an example, 245
  - coverage criteria, 246
  - selecting test cases, 245
  - state model, 245
  - test case selection criteria, 246
- Statement coverage criterion, 248
- Stepwise refinement, 169
- Structural testing, *see* White-box testing
- Structure chart, 132, 137
  - decision representation, 133
  - iteration representation, 133
- Structured design methodology, 177
  - central transforms, 137
  - factoring, 138
  - first-level factoring, 137
  - most abstract input, 136
  - most abstract output, 136
  - steps, 135
- Structured programming, 183
- Superclass, 145

- System design
  - correctness, 122
  - modularity, 122
  - object-oriented design, *see* Object-oriented design
  - principles, 122
  - verification, *see* Design verification
- System testing, 230
  
- Team structure, 89
- Test case design, *see* Test case specification
- Test case generation, 251
- Test case review, 234
- Test case specification, 233
- Test plan, 231
- Test-driven development, 195, 201
- Testing, 225
  - acceptance testing, 230
  - black-box, *see* Black-box testing
  - defect logging, 235
  - deliverables, 233
  - functional, *see* Black-box testing
  - integration testing, 229
  - levels of, 229
  - of object-oriented programs, 207
  - process, 231
  - psychology of, 228
  - purpose of, 229
  - regression, 230
  - structural testing, *see* White-box testing
  - system testing, 230
  - test case design, *see* Test case specification
  - test case execution and analysis, 234
  - test case review, 234
  - test case specification, 233
  - test plan, 231
  - white-box testing, *see* White-box testing
- Testing process, 231
- Timeboxing model, 25, 35
  - a time box, 25
  - execution with three-stage time box, 26
  - iteration completion times, 26
  - pipelined execution of iterations, 25
  - stages in a time box, 25
  - teams, 26
  - teamwise activity, 26
- Tool support for testing, 251
- Top-down approach to coding, 194
- Topological complexity, 221
  
- UML, 177
  - activity diagram, 155
  - aggregation, 151
  - association between classes, 148
  - class diagram, 148
  - class hierarchy representation, 149
  - collaboration diagram, 153
  - components, subsystems, packages, 154
  - interaction diagrams, 151
  - part-whole relationship representation, 151
  - sequence diagram, 151
  - state diagram, 155
  - stereotype, 156
  - subtype, supertype relationship, 148
  - tagged values, 156
  - use case diagrams, 156
- Unified modeling language, *see* UML
- Unit testing, 204, 229
  - an example, 207
  - with Junit, 207
- Usability, 4
- Use case diagram, 51
- Use cases, 49, 66
  - actor, 49
  - development, 56
  - examples, 52, 55
  - extension scenario, 50
  - failure conditions, 56
  - failure handling, 56
  - level of detail, 57
  - levels, 56
  - main success scenario, 50, 56
  - precondition, 52
  - primary actor, 49
  - refinement, 54
  - scenario, 50, 159
  - scope, 54
  - summary-level, 55
  - use case diagram, 51
  - writing them, 57
- User-defined types, 187

## Verification

- inspection, *see* Inspection process
- unit testing, *see* Unit testing

## Waterfall model, 14, 34

- impact of linear ordering of phases, 14
- limitations, 16
- outputs, 16
- stages, 14

## Weighted methods per class, 175

## Weinberg experiment, 181

## White-box testing, 247

- branch testing, 249
- control flow-based, 248
- path testing, 250
- statement coverage criterion, 248
- test case generation, 251
- tool support, 251

## Work products, 15

XP, *see* Extreme programming