Index

Accessibility, 43
Accountability, 43
Accuracy, 43
Ada, 166
Agile methods, 535
architecture, 538
assessing risk exposure, 541
balancing agility and discipline, 539
customers, 538
developers, 537
primary objective, 539
refactoring, 539
requirements, 538
size, 539
Analyzing risk, uncertainty, and the value of information, 121
Anchoring the software process, 367
Augmentability, 43
Automated aids to the design of large-scale reliable software, 231
Being a software engineer in the software century, 797
dependability, 800
diversity, 801
interdependence, 802
rapid change, 797
uncertainty and emergence, 798
Boehm, Barry W.
Air Force CCIP-85 study, 221
back into the software business, 66
CCIP-85, 66
control structures, 56
core domain architecture, 52
DARPA, 65
data structures, 56
desired operational capabilities: user view, 51
domain architecting, 65
domain engineering, 65
domain model and interface specifications, 52
eyear experiences, 47
eyear experiences in software economics, 219
eyear software experiences at Harvard, 48
efforts to get out of the software business, 66
experiences at General Dynamics, 48
first day in the software business, 47
flight program subroutines, 54
Graphic ROCKET, 63
great ideas, 573
integrated architecting: control structures, data structures, and user interface, 55
learning and doing software economics analyses at Rand, 220
lessons learned, 61
megaprogramming, 65
Programmer-Oriented Graphic Operation (POGO), 65
published interface specifications, 53
Rand and its environment, 49
rejoining the software field at TRW, 222
rocket development and usage, 61
rocket trajectory domain architecting, 50
user interfaces, 57
Bottom-up software development, 7
advantages, 8
disadvantages, 8
Index

Certiﬁcation technology, 106
Checklists, 37
COCOMO 2.0, 269
  ACAP—Analyst Capability and PCAP—
      Programmer Capability, 293
  AEXP—Application Experience, PEXP—
      Platform Experience, 294
application composition: object points, 276
application maintenance, 286
applications development, 279
breakage, 286
cost factors
  eﬀort multiplier cost drivers, 289
  scaling, 286
  sizing, 276
CPLX—Product Complexity, 291
development schedule estimates, 296
DOCU—Documentation Match to Life-
      Cycle Needs, 291
  early design and post-architecture function
      point estimation, 295
  function point counting rules, 280
  lines of code counting rules, 279
LTEX—Language and Tool Experience,
      294
major differences between COCOMO and
  COCOMO 2.0, 276
model rationale and elaboration, 274
models for the software marketplace sectors,
      273
MODP—Use of Modern Programming
      Practices, 294
  motivation for, 269
  object point estimation procedure, 278
  objectives, 270
  output ranges, 296
PCON—Personnel Continuity, 294
  personnel factors, 293
  platform factors, 291
  project factors, 294
PVOL—Platform Volatility, 293
  reuse and reengineering, 282
RUSE—Required Reusability, 291
SCED—Required Development Schedule,
      295
SECU—Classiﬁed Security Application, 295
SITE—Multisite Development, 295
  strategy and rationale, 273
TIME—Execution Time Constraint and
      STOR—Main Storage Constraint, 293
TOOL—Use of Software Tools, 294
topics addressed, 271
TURN—Computer Turnaround Time, 293
COCOMO model, 133
  cost driver ratings, 138
  module complexity ratings versus type of
      module, 137
  nominal eﬀort and schedule equations, 135
  software cost driver ratings, 136
  software development eﬀort multipliers, 135
  software development modes, 134
Commercial oﬀ-the-shelf (COTS) products, 69,
      529, 530
  ensure smooth interconnections, 71
  evolutionary dead ends, 72
  foster realistic expectations, 73
  functionality and performance, 70
  interoperability, 71
  nurture beneﬁcial mutations, 72
  perishable promises, 73
  plusses and minuses, 69
  product evolution, 71
  roadblocks to interoperation, 71
  use risk-driven process models, 70
Communicativeness, 43
Completeness, 44
Conciseness, 44
Consistency, 44
Constructive Cost Model; see COCOMO
Cost models for future software life cycle
  processes, 269
COTS; see Commercial oﬀ-the-shelf
COTS-based systems top 10 list, 81
  average COTS software product undergoes a
      new release every eight to nine months,
      with active vendor support for only its
      latest three releases, 82
CBS assessment and tailoring eﬀorts vary
  signiﬁcantly by COTS product classes, 84
CBS development and postdeployment
eﬀorts can scale as high as the square of
  the number of independently developed
  COTS products targeted for integration,
  82
CBS is currently a high-risk activity, 85
CBS postdeployment costs exceed CBS
development costs, 83
glue-code development usually accounts for
  less than half the total CBS software
development eﬀort, 83
more than 99% of all executing computer
  instructions come from COTS products.
  Each instruction passed a market test
  for value, 81
more than half the features in large COTS software products go unused, 82
nondevelopment costs, such as licensing fees, are significant, and projects must plan for and optimize them, 84
personnel capability and experience remain the dominant factors influencing CBS-development productivity, 85
COTS integration, 69
Critical path, 94

DACC Design Element Report Format, 238
additional design features, 243
costs, 243
experience on a large design specification, 240
useful features, 242
Defining delivered source instructions (DSI), 154
Design and coding error types, 232
Design Assertion Consistency Checker (DACC), 237
Design data input, 239
Design information representation, 238
Design query capabilities, 240
Developing groupware for requirements negotiation, 301
Device independence, 44
Differences between large and small software design activities, 232
Early application generator, 47
Early experiences in software economics, 219
Economic value of information, 123
Economics and software engineering management, 117
Efficiency, 44
Emerging value-driven design and development approaches, 209
Error causes by phase and category, 234
Error correction effectiveness of metrics, 36
Expectations management, 607
coming away a winner, 610
solving problems, 608
planning for success, 609
speaking clearly, 608
what is the problem?, 608
Future software practices marketplace model, 271
Future trends and implications for systems and software engineering processes, 545
Gaining intellectual control of software development, 687
NSF Workshop findings, 688
potential roadblocks, 694
taking the long view, 687
USC Workshop findings, 689
Generalized Information Management (GIM) Support Structure, 237
Hardware–software trade-offs, 102
Human engineering, 44
IKIWISI (I’ll know it when I see it), 529, 530
Implementing risk management, 481
Improving software economics within an enterprise, 201
design for life cycle value, 206
education, 213
efficient nature of software estimation accuracy, 202
general benefit-modeling techniques, 203
investing in the anticipation of change, 208
modeling costs, benefits, and value, 201, 202
modeling software development cost, schedule, and quality, 201
modeling value: relating benefits to costs, 204
opportunities for improvement, 205
research challenges, 212
tracking and managing for value, 204
value-driven design, 207
Improving software management, 99
Improving software productivity, 151
importance of, 152
Increased productivity strategy, 179
Increasing software productivity, 94
factors, 95
prescriptions, 97
Indirect costs, 93
ISDOS, 637
ISO/IEC 12207: IT life cycle processes, 369
J-STD-016, 368
Legibility, 44
Maintainability, 44
Managing software productivity and reuse, 179
Measuring software productivity, 153
defining inputs, 153
defining outputs, 154
DSI, 154
Metrics, 157
Model-Based (System) Architecting and Software Engineering (MBASE), 209
Model-driven software design, 12
advantages, 16
disadvantages, 16
Modifiability, 45
Objectives and priorities, 37
Planning spectrum, 535
Portability, 45
Programming, 643
current frontier technology, 644
current practice, 643	
trends, 644
Project termination doesn’t equal project failure, 737
project termination sources, 737
successful project terminations, 739
Prototyping versus specifying: A multiproject experiment, 319
Quality assurance activity, 37
Quality characteristics, 26
Quality code, 24
evaluation of metrics, 30
evaluation of metrics versus project error experience, 30
metrics, 28
Quality-enhancing tools and techniques, 39
Quantitative evaluation of software quality, 21
Quantitative models of software life-cycle evolution, 143
Quantitative models of software project dynamics, 143
Rapid application development (RAD), 523
acquire better people, 525
avoid single-point task failures, 525
eliminate tasks, 524
implementing full-scale RAD, 524
increase the effective workweek, 525
making RAD work for your project, 523
RAD forms, 523
reduce backtracking, 525
reduce time per task, 524
streamline activity networks, 525
surviving dumb RAD, 526
transition to a learning organization, 526
Rapid change, 529, 530
Reliability, 45
Requirements quandary, 530
change-driven requirements, 532
risk-driven requirements, 533
shared-vision-driven requirements, 531
surmounting, 531
value-driven requirements, 531
Risk analysis, 440
automated risk-analysis aids, 448
cost risk analysis, 445
cost model and cost driver analysis, 445
schedule risk analysis, 447
decision analysis, 441
network analysis, 442
statistical decision theory, 442
Risk assessment, 427
Risk control, 452
Risk identification, 428
assumption analysis, 437
checklists, 429
decision-driver analysis, 436
decomposition, 438
performance shortfalls, 435
personnel shortfalls, 429
shortfalls in external components and tasks, 434
straining computer science capabilities, 435
unrealistic schedules and budgets, 430
Risk-item reassessment, 461
Risk-item tracking, 459
Risk management in the software life cycle, 481
life-cycle roles of the customer and the developer, 481
risk-oriented software-process models, 482
spiral model, 483
Risk-management plan, 361
Risk-management planning, 453
plan formulation and coordination, 454
planning process, 453
Risk-management practices, 427
Risk monitoring, 457
Risk prioritization, 449
assessing risk probabilities, 449
dealing with compound risks, 450
Risk resolution, 456
Risk-resolution techniques, 471
cost and schedule estimation, 473
design to cost, 473
incremental development, 473
information hiding, 475
mission analysis, 474
performance engineering, 475
preaward audits, 475
prototyping, 474
reference checking, 475
requirements scrubbing, 474
staffing and prescheduling key people, 471
team building, 472
technical analysis, 476
Robustness, 45

Scalable spiral process model for 21st century
systems and software, 559
acquisition as C2ISR versus purchasing, 566
agile rebaselining and the C2ISR metaphor, 562
human relations, 566
implications for 21st century enterprise
processes, 565
model experience to date, 565
overview, 560
synchronizing hardware, software, and
human systems processes, 563
21st century system and software
development and evolution modes, 559
Self-containedness, 45
Self-descriptiveness, 45
Software and systems management, 573
Software and its impact: a quantitative
assessment, 91
Software architecture, 1
Software cost driver attributes and their effects, 142
Software cost estimation, 123, 140
algorithmic models, 126
COnstructive COst MOdel (COCOMO), 133
Doty model, 131
fundamental limitations, 125
major software cost estimation techniques, 124
Putnam SLIM model, 129
RCA PRICE S model, 132
Software cost estimation models, 139
Bailey–Basili meta-model, 140
Grumman SOFCOST model, 140
Jensen model, 140
Tausworthe Deep Space Network (DSN)
model, 140
Software cost model analysis and refinement, 142
Software data collection, 144
Software defect reduction top 10 list, 75
40 to 50% of user programs contain
nontrivial defects, 78
50% more per source instruction to develop
high-dependability software products
than to develop low-dependability
software products, 78
80% of avoidable rework comes from 20%
of the defects, 76
80% of the defects come from 20% of the
modules, 76
90% of the downtime comes from, at most,
10% of the defects, 77
disciplined personal practices can reduce
defect introduction rates by up to 75%,
77
finding and fixing a software problem after
delivery, 75
peer reviews catch 60% of the defects, 77
perspective-based reviews catch 35% more
defects than nondirected reviews, 77
software projects spend about 40 to 50% of
their effort on avoidable rework, 76
Software design, 1, 5, 639
current frontier technology, 640
design representation, 642
modularization, 641
top-down design, 640
current practice, 640
requirements/design dilemma, 639
trends, 643
Software design specification and review, 5
Software development environment for
improving productivity, 245
Software economics, 87
better monitoring and control for dynamic
investment management, 200
future trends, 193
history and current status, 193
increased emphasis on, 188
links between software economics and
policy, 198
links between technical parameters and
value, 197
making decisions that are better for value
creation, 196
need for multistakeholder satisficing, 191
new measures of value, 190
new sources of value, 189
richer design spaces, 197
road map, 195
shortcomings that need to be addressed, 194
Software economics roadmap, 185
Software engineering, 633
definitions, 634
software requirements engineering, 635
critical nature of, 635
current frontier technology, 637
current practice, 636
trends, 638
Software engineering as a value-creation activity, 187
Software engineering database, 108
Software engineering decision making, 186
Software engineering economics, 117
analysis techniques, 119
benefits and challenges, 144
benefits of a software engineering economics perspective, 144
benefits of software cost estimation technology, 145
challenges, 146
Software engineering state of the art and practice, 627
Software engineering—as it is, 663
how soon will we learn these software engineering lessons?, 666
how well have the lessons been learned?, 664
integrate approaches, 678
recent developments, 673
some factors inhibiting general progress in software engineering practice, 667
some software engineering lessons learned, 663
Software error causes, 6
Software error data analysis, 235
Software error sources, 6
Software-first machine, 99
Software-intensive systems (SIS) trends and their influence on systems and software engineering processes, 546
complex systems of systems trends, 555
computational plenty trends, 558
COTS, reuse, and legacy integration challenges, 556
globalization and interoperability trends, 554
increasing integration of software engineering and systems engineering, 546
rapid change trends, 551
systems and software criticality and dependability trends, 550
user/value emphasis trends and process implications, 548
wild cards: autonomy and biocomputing, 558
Software is big business, 92
Software maintenance, 649
current frontier technology, 651
current practice, 650
scope of, 649
trends, 652
Software management and integrated approaches, 652
current frontier technology, 653
current practice, 652
trends, 655
Software-model-clash spiderweb, 743, 744
avoiding, 746
clash of models: MasterNet, 745
model clashes, 743
Software process, 315
emerging extensions, 499
Software process models, 345
code-and-fix model, 346
evolution of, 346
evolutionary development model, 348
stagewise models, 346
transform model, 349
waterfall models, 346
Software product value chain, 159
Software productivity, 157
analyzing, 157
ranges, 157
Software productivity improvement opportunity tree, 161
building simpler products, 168
eliminating rework, 165
eliminating steps, 164
facilities, 162
front-end aids, 165
getting the best from people, 161
improved process models, 166
information hiding and other modern programming practices, 165
knowledge-based software assistants, 165
making steps more efficient, 164
management, 163
modern programming practices and Ada, 166
rapid prototyping, 167
reusing components, 169
staffing, 162
Software productivity system (SPS) requirements analysis, 245
1980 Software Productivity Study, 246
architecture and components, 252
corporate motivating factors, 246
experience supporting other projects, 263
general utilities, 257
hardware, 253
master project database, 254
office automation and project support, 259
project experience, 262
software, 256
Index

software development tools, 260
support scenarios, 261
training, 263
usage measurement, 264
work environment, 253
Software productivity trends, 171
Software productivity–quality interactions, 156
Software product-line management, 369
feasibility rationale, 372
IOC, 374
LCO/LCA: distinguishing features, 373
life-cycle architecture, 373
life-cycle objectives, 370
life-cycle plan, 372
operational concept, 370
system and software architecture, 372
system requirements, 372
top-level system objectives, 370
transitions, 375
Software project risk assessment, 1
Software quality, 1, 37
code, 40
requirements and design phases, 39
Software quality decision points, 22
checking for compliance with quality specifications, 22
making proper design trade-offs between development costs and operational costs, 22
preparing the quality specifications for a software product, 22
software package selection, 22
Software quality evaluation, 24
Software quiz, 110
Software R&D investment policy research framework, 199
Software reliability and certification, 104
problem symptoms, 104
technical problems, 105
Software responsiveness, 103
Software reuse, 179
pitfalls, 180
resources, 181
reuse works, 180
success factors, 181
Software risk management, 383
avoiding disasters, 405
avoiding overkill, 406
avoiding rework, 405
basic concepts, 388
fundamental concepts, 408, 411
generic and project-specific risks, 415
implementing, 399
primary risk reduction options, 409
principles and practices, 387
risk analysis and prioritization, 394
risk exposure, 411
risk-identification checklists, 392
risk management, 390
risk-management planning, 396
risk reduction leverage (RRL), 413
risk resolution and monitoring, 397
stimulating win–win situations, 407
typical risk-management situation, 408
UniWord Project, 416
using risk exposure to prioritize risk items, 413
what is software risk management?, 403
why is software risk management important?, 405
Software size and complexity metrics, 141
Software size estimation, 140
Software structuring, 5
Software testing and reliability, 644
current frontier technology, 645
automated aids, 646
fault-tolerance, 648
program proving (program verification), 648
software error data, 645
software reliability models and phenomenology, 645
symbolic execution, 647
test sufficiency and program proving, 647
current practice, 644
trends, 649
Software tools, 227
Spiral acquisition of software-intensive systems of systems, 615
risks and challenges, 616
software benefits for transformational systems, 615
Top 10 SISOS risks and spiral mitigation strategies, 619
achievable software schedules, 620
acquisition management and staffing, 619
adaptation to rapid change, 623
external interoperability, 624
product integration and electronic upgrade, 623
requirements/architecture feasibility, 620
software COTS and reuse feasibility, 624
software quality factor achievability, 623
supplier integration, 622
technology readiness, 625
WinWin spiral model overview, 617
Spiral model, 315
Spiral model of software development and enhancement, 345
Spiral model of the software process, 349
advantages, 358
cost of operations, 354
detailed design go-back, 357
difficulties, 360
feasibility study, 353
features, 358
initiating and terminating the spiral, 352
matching to contract software, 360
need for further elaboration, 361
relying on risk-assessment expertise, 360
results, 358
risk management plan, 361
RTT preliminary design specification, 357
Top-level requirements specification, 356
typical cycle of the spiral, 350
using the spiral model, 352
SREP, 637
STARS Project, 377
commercializing STARS, 378
milestones, 380
results, 380
winning compromises, 379
Statistical decision theory, 19
Structured programming, 11
advantages, 11
disadvantages, 12
Structuredness, 45

Taxonomy of software error causes, 234
Technical–value mismatch, 187
Testability, 45

Theory of value-based software engineering, 777
4+1 theory, 779
motivation and context, 777
Theory W, 780
using and testing the 4+1 VBSE theory:
process framework and example, 785
VBSE theory context, 778
VBSE theory evaluation with respect to
goodness criteria, 792
Theory W, 581
case study, 593
comparison to Theories X, Y, and Z, 582
connections between Theory W and game theory, 588
deriving strategic project guidelines from
Theory W win–win steps, 586
make everyone a winner, 582

subsidiary principles, 589
flying the plan, 590
planning the flight, 589
risk management, 591
risk-management steps, 591
tactical management example, 588
Theory-W software project management, 579
software management theory problem, 580
software project manager’s problem, 580
Thorpe/DMR benefits realization approach, 211
Top-down problem statement, 10
advantages, 10
disadvantages, 11
Top-down structured programming, 18
Top-down stub approach, 8
advantages, 9
disadvantages, 9

Understandability, 45

Unifying software engineering and systems engineering, 611
integrated CMM improvements, 612
large sequential-engineering near-disaster, 613
rooting out the waterfall legacy, 612
unified project approaches, 613
Usability (as-is utility), 45
Using quality characteristics to improve the software life-cycle process, 36

Value-based processes for COTS-based applications, 763
Five principles for CBA development, 764
avoid premature commitments, but have and use a plan, 766
buy information early to reduce risk and rework, 766
do not start with “requirements;” prepare for COTS change, 767
process happens where the effort happens, 764

glue-code element, 773
tailoring element, 772
value-based CBA process decision framework, 767

Value-based software engineering: A case study, 749
accounting for value in software engineering, 750
benefits-realization results chain, 756
costs, benefits, and ROI, 757
order-processing example, 755
real earned-value feedback control, 753
using earned value, 751
value-based monitoring and control, 759
value-based software engineering agenda, 750
Value-based software engineering, 731
Value-of-information approach, 122
Verification and validation, 5
View of 20th and 21st century software engineering, 697
2010s and beyond, 717
2010s antitheses and partial syntheses:
globalization and systems of systems, 717
computational plenty trends, 720
software-intensive systems of systems, 718
wild cards: autonomy and biocomputing, 720
Hegelian view of software engineering’s past, 698
1950s thesis: software engineering is like hardware engineering, 698
1960s antithesis: software crafting, 700
1970s synthesis and antithesis: formality and waterfall processes, 701
1980s synthesis: productivity and scalability, 705
1990s antithesis: concurrent versus sequential processes, 709
2000s antithesis and partial synthesis:
agility and value, 711
agile methods, 711
COTS, open source, and legacy software, 713
interacting software and systems engineering, 716
model-driven development, 715
software criticality and dependability, 713
value-based software engineering, 712
software engineering education, 723
timeless principles and aging practices, 721
WinWin approach, 302
EasyWinWin, 305
four generations of tool support, 304
group support system, 305
group support system infrastructure, 304
groupware lessons, 310
how does the WinWin negotiation model work?, 303
initial prototype, 304
lessons learned, 309
methodology lessons, 309
muscle-bound architecture, 304
project lessons, 312
strong vision, not-so-strong architecture, 304
why does WinWin work, 302
win–lose doesn’t work, 302
WinWin builds trust and manages expectations, 303
WinWin helps build institutional memory, 303
WinWin helps stakeholders adapt to changes, 303
WinWin Spiral Lifecycle Model, 192
WinWin Spiral Model, 375, 503
application, 507
adoption of applications, 516
application family, 507
application life-cycle architectures, 513
client acceptance, 518, 519
client involvement and reaction, 515
communication and trust, 518
degree of flexibility, 517
documentation, 519
initial operational capability, 514
layered architectural description, 519
nature of products, 516
number of cycles, 517
risk management, 515
smooth transitions, 518
use of developer time, 518
case study, 503
elements of, 504
milestone criteria, 376
negotiation front end, 504
process anchor points, 505
spiral cycles, 377
stakeholder concerns, 376
up-front milestones, 377
Win–win, win–lose, and lose–lose situations, 583
creating win–win situations, 583
establish reasonable expectations, 585
match people’s tasks to their win conditions, 585
understand how people want to win, 583