

A

Abstraction level, 53, 55
 Accident, 235, 377
 Adaptive testing, 113, 377
 Advice, logging, 182–86
 Algorithmic and Automatic
 Debugging Home
 Page, 20
 Algorithmic debugging
 (declarative
 debugging),
 137–40, 141, 144,
 145, 377
 ALICE language, dynamic
 slicing, 213–16
 ANDROID, 54–55,
 72, 102
 Anomaly, defined, 377
 Anomaly detection
 capturing normal
 behavior,
 253–54
 code sequences
 detection, 258
 collecting field data,
 260–61
 comparing program
 code coverage,
 254–58
 dynamic invariants,
 262–65
 exercises, 269–70
 finding defects from
 anomalies, 266–67
 induction techniques,
 254
 issues for field data
 collection, 260
 nearest neighbor
 detection, 258
 overview, 253
 program run properties,
 253–54
 statistical debugging,
 259–60
 summary, 267

TARANTULA, anomaly
 detection tool,
 256–57
 tools for, 268
 APPLESCRIPT, 55–56,
 57–58, 72
*Art of Scientific
 Investigation*, 145
 ASKIGOR automated
 debugger
 description, 16
 isolating cause and
 effect chain
 example, 316–20
 reference, 326
 ASPECTJ logging utility,
 182–86, 203
 Aspects, logging with,
 182–86
 Assertion techniques
 asserting correctness,
 229–32
 asserting invariants,
 226–29
 assertions as
 specifications,
 232–33
 automating observation,
 223–24
 basic assertions, 224–26
 checking production
 code, 242–44
 exercises, 247–51
 GUARD relative
 debugger, 237
 overview, 223
 pre and post-conditions,
 230–32
 production code checks,
 242–44
 reference programs,
 235–38
 relative debugging,
 237–38
 summary, 244–45
 system assertions,
 238–42

 avoiding buffer
 overflows with
 ELECTRICFENCE,
 239
 detecting memory
 errors with
 VALGRIND, 240–41
 language extensions,
 241–42
 overview, 238–39
 validating heap with
 MALLOC_CHECK,
 239
 tools for, 245–46
 traditional specifications,
 232
 and verification, 233–35
 Z specification language,
 232
 Audience for this book,
 xviii
 Automated assertion
 advantages, 224
 data invariants, 226–29
 examples, 228
 systematic uses, 226
 Automated debugging
 techniques
 ANDROID scripting
 (Mozilla), 54–55
 anomalies, 15–16
 asserting an invariant, 15
 assessing results, 56–57
 benefits, 53
 circumstance difference
 analysis, 121
 fragility, 54
 high-level interaction,
 55–56
 introduction, 14–15
 low-level interaction, i/o
 capture and replay,
 53–55
 Mozilla Talkback dialog,
 30
 observing state, 15
 privacy issues, 31

- Automated debugging techniques
 - (*continued*)
 - program control, 50–53
 - program slice, 15
 - simplified input, 14–15
 - system-level interaction, 55
 - testing layers, 52
 - using syntax for simplification, 120–21
 - watching state, 15
 - ways to speed the automation, 120
- XML, 59
- Automatic simplification, 110–12
- Automating observation, 223–24
- Automation*, 73
- B**
- Backward slices, 158, 160, 214, 216, 218, 332, 377
- BCEL debugging tool, 203
- Binary level, logging at, 186–88
- Blame-o-meter, 296
- Bohr bug, 91
- Bug report, 26, 38, 377, *see also* Problem report (PR) information
- BugAThon, Gecko, simplifying problems, 106–8
- Bugs
 - cache for, 355–56
 - vs. defects, 18–19
 - defined, 377
 - vs. faults, 18–19
 - first bug, 2
- BUGZILLA problem database, 31–32
- C**
- Cache for bugs, 355–56
- Caching, 119
- Cause, defined, 377
- Cause transitions, 320, 321, 324, 326
- Cause-effect chains
 - defined, 377
 - isolating, 305–27, *see also* Causes and effects; Failure causes
 - capturing program states, 307–11
 - cause transitions, 320–24
 - comparing program states, 311–12
 - description, 305
 - exercises, 327–28
 - failure-inducing code, 320–24
 - GNU C compiler crash example, 306
 - isolation method steps, 307
 - issues and risks, 324–26
 - memory graph, 311–12
 - overview, 305–7
 - relevant program states, 312–16
 - summary, 326
 - tools for, 326
 - unfolding a program state, 309
 - useless causes, 305–7
- Causes and effects, xxi, 271–80, *see also* Cause-effect chains, isolating; Failure causes
- causality in practice, 273–75
- causes in debugging, 278
- common context, 278
- exercises, 280–81
- finding actual causes, 275–76
- overview, 271–72
- summary, 279
- verifying causes, 272–73
- CCACHE, 297, 301
- CCURED programming language, 246
- CHAINSAW, logging utility, 182, 183
- Change request (CR), 26, 377, *see also* Problem report (PR) information
- Checking production code, 242–44
- Chocolate milk problem example, 38
- Chop operation, program slicing, 158–59
- Circular dependence, 64, 69
- Circumstance, defined, 377
- Classifying problems, 32–34
- COCA debugger (GDB), 198, 204, 294, 295, 297
- Code Complete*, 21
- Code smells, 161–66
 - defined, 377
 - interface misuse, 164
 - memory leaks, 163–64
 - null pointers, 164
 - reading uninitialized variables, 161–62
 - summary, 170
 - unreachable code, 162–63
 - unused values, 162
- CODESURFER, 157, 160, 166, 170
- Configuration, defined, 377
- Control flow graph, deductive debugging, 148–52
- Control layer, reproducing problems, 101
- Core dump, 192
- Correction, defined, 377
- Correctness
 - asserting, 229–32
 - defined, 377
- Correspondence graph, 327
- Cosmic rays, 88
- Counterfactuals, 145, 279
- CR (Change request), 26, *see also* Problem report (PR) information
- Crash, defined, 377
- Critical slicing, 221
- CYCLONE C language extension, 242, 246

D

- DAIKON invariant
 - detection tool, 234, 262–65, 267, 268
- Data Display Debugger (DDD), 201–2, 295, 296, 300, 310
- Data race failure, 291
- DDCHANGE for ECLIPSE, 298, 301
- DDD (Data Display Debugger), 201–2, 295, 296, 300, 310
- Ddmin algorithm
 - general description, 124
 - Python implementation, 113–16
- Debuggee, defined, 377
- Debuggers
 - after program crashes, 192–93
 - altering code during execution, 194
 - caveats, 195
 - controlling execution, 192
 - debugging session, 189–92
 - defined, 377
 - embedded, 194–95
 - invoking functions, 194
 - logging data, 193
 - overview, 188–89
- Debugging
 - defined, 377
 - program in seven steps, 20
 - statistics, 19
- Declarative debugging
 - (algorithmic debugging), 137–40, 141, 144, 145, 377
- Dedicated logging
 - advantages, 177
- Deducing errors, 147–71
 - code smells, 161–66
 - control flow, 148–52
 - exercises, 171–73
 - isolating value origins, 147–48
 - limits of static analysis, 166–70
 - overview, 147
 - slicing programs
 - backward slices, 158
 - executable slices, 160–61
 - forward slices, 157–58
 - leveraging slices, 160
 - overview, 157
 - slice operations, 158–60
 - tools for, 170
 - tracking dependencies
 - affected statements, 153–54
 - effects of statements, 152–53
 - following
 - dependences, 156
 - leveraging
 - dependences, 156–57
 - overview, 152
 - statement
 - dependences, 154–56
- Deduction, defined, 378
- Deduction debugging
 - techniques
 - changes to the program state, 153–54
 - code smells, 161–66
 - control flow, Fibonacci example, 148–49
 - control flow graph, 149–51
 - introduction, 147
 - program slicing, 147, 171
 - statement dependences, 154–56
 - unstructured control flow, 152
- Defect pattern, defined, 378
- Defects
 - applying corrections to code, 335
 - vs. bugs, 18–19
 - checking for multiple
 - locations defects, 336–37
 - checking for new
 - problems, 336–37
 - correcting, 335–38
 - defined, 18, 378
 - ensuring fixes correct
 - defects, 336
 - exercises, 340–41, 360–61
 - vs. faults, 18–19
 - focusing on most likely
 - errors, 330–32
 - improving quality, 347
 - learning from, 343–61
 - lifecycle, 25–26
 - locating, 329–30
 - mining, 344–46
 - most likely errors, 332
 - non-programmer
 - caused, 2
 - origins, 330–32, 346–47
 - overview, 329–30, 343–44
 - patterns, 165–66, 171, 344, 359
 - predicting
 - from change
 - frequency, 355
 - from imports, 354
 - during programming, 349–51
 - during quality assurance, 351–53
 - relationship to program failure, 1–4
 - risky, 353
 - during specification, 347–51
 - steps from infection to
 - program failure, 2–4
 - summary, 359
 - validating, 332–35
 - when can't be changed, 338
 - workaround examples, 338–39
- DEJAVU record and replay
 - tool, 291–93
- Delta, defined, 378
- Delta debugging
 - defined, 378
 - description, 14–15
 - introduction, 105
 - isolating cause and effect
 - chains, 305–27

- Delta debugging
 - (*continued*)
 - issues and risks, 324–26
 - limitations, 299–300
 - questions for debuggers, 325
 - user interaction, 117–18
 - Zeller and Hildebrandt (ddmin algorithm), 112
 - Dependence inversion principle*, 65, 71, 73
 - Dependencies of infected variables, 8
 - Design, debugging
 - considerations, 66–69
 - Design Patterns*, 73
 - Developer tests vs. problem reports, 43
 - Diagnosis, defined, 132
 - Dice operation, program slicing, 160
 - DIDUCE anomaly
 - detection tool, 265, 266, 268
 - Dijkstra, Edsger, 4, 19, 21
 - Domino effect, 351
 - Dr. Watson file (Windows), 192
 - Duplicate problem
 - identification, 39
 - Dynamic analysis, 143
 - Dynamic dispatch, control flow, 152
 - Dynamic invariants, 262–65
 - Dynamic slicing
 - description, 213–16
 - drawbacks, 216
 - example, 214–16
 - Korel and Laski, 221
 - method formal
 - description, 216
 - predicate
 - pseudovariable, 214
 - summary, 220
 - tracking origins, 213–16
 - WHYLINE system, 216–19, 221
- E**
- ECLIPSE method, 356
 - EDOBS visualization tool
 - for debugging, 204
 - Effect, defined, 378
 - EIFFEL language
 - design by contract
 - concept, 231, 246
 - Meyer (1997), 246
 - ELECTRICFENCE, buffer
 - overflow errors, 239–40
 - Embedded debuggers, 194–95
 - Error, *see also* Defects
 - defined, 378
 - vs. infection, 18–19
 - ESC/Java, 234, 245, 247
 - Events, querying
 - overview, 196
 - uniform event queries, 197–99
 - watchpoints, 196–97
 - Exceptions
 - control flow, 152
 - defined, 378
 - Executable slices, 160–61
 - Execution, logging, 176–88
 - at binary level, 186–88
 - logging frameworks, 180–82
 - logging functions, 177–80
 - logging with aspects, 182–86
 - overview, 176–77
 - Experiment, defined, 378
 - Experimental analysis, 378
 - Experimentation
 - techniques, 330–31
 - Explicit debugging, 134–35
 - External observation tool
 - benefits, 188–92
- F**
- F-16 flight software bugs, 3
 - Facts about failures, 19–20
 - Failure
 - automation and
 - isolation, 50
 - defined, 18, 378
 - introduction to their
 - origin, xix, 1–2
 - Failure causes, *see also* Cause-effect chains, isolating; Causes and effects
 - description, 271
 - experiments to verify causes, 278
 - finding a common
 - context, 277–78
 - isolating, 283–302
 - algorithm for isolation, 286–88
 - automatic cause
 - isolation, 283–84
 - description, 283
 - exercises, 302–3
 - failure-inducing input, 290–91
 - failure-inducing
 - changes, 293–98
 - failure-inducing
 - schedules, 291–93
 - general delta
 - debugging
 - algorithm, 287–301
 - implementing
 - isolation, 288–90
 - isolation algorithm, 286–88
 - overview, 283–84
 - problems and
 - limitations, 299–300
 - Python examples of
 - isolation, 288–90
 - vs. simplifying, 284–86
 - summary, 301
 - tools for, 301
 - narrowing possible
 - causes, 276–77
 - Ockham's Razor, 276, 279
 - practical causality, 273–75
 - select failure causes
 - from alternatives, 275–76
 - in simplification, 107–8

- theory of causality, 271–72
- verifying causes, 272–73
- Fallacy, defined, 378
- Fat pointers, CYCLONE C language extension, 242
- Fault
 - vs. defect, 18–19
 - defined, 18, 378
- Feature, defined, 378
- Feature requests vs. problem reports, 43
- FINDBUGS for JAVA, 165–66, 170
- Fix, defined, 378
- Fixing, defined, 378
- Flaw, 18, 378
- Formal specifications, 348
- Forward slices, 157–58, 378
- Frameworks, logging, 180–82
- Functionality layer, 51–52, 57–59
- Functions, logging, 177–80
- Fuzz input, 118–19

G

- GAMMA project (remote sampling), 268
- GDB (GNU debugger), 188–94, 196–201, 203–4, 294–97, 308–9, 316–18
- Gecko, 50
- Gecko BugAthon
 - example, 106–8
 - reference, 124
- Gecko example
 - simplifying problems, 107–8
 - testing techniques, 49–50
- General delta debugging algorithm, 287, 301
- GFORGE, 45–46
- GNU debugger (GDB), 188–94, 196–201, 294–97, 308–9, 316–18
- GUARD relative debugger, 237–38, 246

H

- Hanging, 378
- HATARI tool, 356
- Heap memory errors, C and C++, 238
- Heisenbug, 89–91, 378
- High-level interaction, 55–56
- Hildebrandt, 112, 113, 117, 119, 124
- Hopper, Grace, 21
- Hypothesis
 - defined, 378
 - deriving, 140–42, 144

I

- IBUGS repositories, 360
- IEEE defined, fault vs. defect, 18
- IGOR, *see also* ASKIGOR
 - automated debugger
 - example, 316–18, 320, 326
 - overview, 316
 - reference, 326
- Incident, defined, 378
- Indirect jumps, control flow, 152
- Induction
 - defined, 378
 - techniques, 254, 330–31
- Inductive analysis, 378
- Infected variable value
 - origins, 7, 8
- Infection chain, 4, 5, 379
- Infection in programs, 3, 18, 378
- Information hiding, 69
- Inputs, controllable, 79, 81
- Instructor advice, xxii
- INSURE++, memory error detection, 246
- Interfaces, unit layer testing, 59
- Internet Explorer, 57–58
- Introduction to Scientific Research*, 145
- Invariants
 - asserting, 226–29
 - defined, 379

- Isolating cause-effect chain, *see* Cause-effect chains, isolating
- Isolating failure causes, *see* Failure causes, isolating
- Issue, defined, 379
- ISSUETRACKER, 45

J

- Java Modeling Language (JML), 233–34, 245, 247
- JAVA SPIDER
 - description, 204
 - references, 204
- Jeffery, Clint, 20
- JML (Java Modeling Language), 233–34, 245, 247
- Jumps and gotos, control flow, 152
- JUNIT testing framework, Java class testing, 60–63, 72, 234

L

- Language extensions, 241–42
- Lifecycle diagram, problems, 34
- Limits of automated debugging, 325
- LOG4J logging framework, 180–82, 203
- Logbook in scientific debugging, 135–36
- Logging frameworks
 - LOG4J, 180–82
 - summary, 203
- Logging functions, 177–80, 203
- Logging macros, 178–80
- Logging statement
 - drawbacks, 176

M

- Machine learner, 354
- Macros, logging, 178–80
- Malfunction, defined, 379

MALLOC_CHECK, heap validation, 239
 Mandelbug, 91
 Manual simplification, 109–10
 Mark II machine bug (Harvard), 2
 Mastermind game example, 134–35
 Memory dump, 192
 Mining software repositories (MSRs), 359
 Mishap, defined, 379
 Mistake, defined, 343
 Mock object, 95–96
 Model-view-controller architectural pattern, 66–68
 MOZILLA
 ANDROID scripting, 53–55
 automated debugging techniques, 50
 vulnerability distribution in, 343, 344
 MSRs (Mining software repositories), 359
 Mutation testing, 352

N

N-version programming, 247

O

Observation, defined, 379
 Observational analysis, 379
 Observing facts
 ASPECTJ logging utility, 183–85, 203
 BCEL debugging tool, 203
 binary level logging, 186
 COCA debugger (GDB), 198–99
 DDD, Data Display Debugger, 201–2, 204
 debugger tools, 188
 dedicated logging advantages, 176–77

dedicated logging techniques, 177
 eDOBS visualization tool for debugging, 204
 embedded debuggers, 194–95
 exercises, 204–7
 fix and continue, 194
 GDB (GNU debugger), 188–94, 196–97, 203
 invoking function while debugging, 194
 JAVA SPIDER, 204
 LOG4J logging framework, 180–82, 203
 logging aspects, 182–86
 logging configuration files, 182
 logging data, 193
 logging execution, 176–88
 logging at binary level, 186–88
 logging frameworks, 180–82
 logging functions, 177–80
 logging macros, 178–80
 logging statement drawbacks, 176
 logging statements, 176
 logging with aspects, 182–86
 overview, 176–77
 observing state, 175–76
 overview, 175
 PIN logging framework, 186–88, 203
 postmortem debugging, 192–93
 program state, 175
 querying events, 196
 overview, 196–99
 uniform event queries, 197–99
 watchpoints, 196–97
 rules for observation, 175–76

summary, 202
 testing a hypothesis, 189–92
 tools for, 203–4
 using debuggers
 after program crashes, 192–93
 altering code during execution, 194
 caveats, 195
 controlling execution, 192
 debugging session, 189–92
 embedded, 194–95
 invoking functions, 194
 logging data, 193
 overview, 188–89
 visualizing state, 200–202
 watchpoints, 196
 Ockham's Razor, causality, 276, 279
 ODB debugger for JAVA description, 212
 drawbacks, 213
 web reference, 221
 Omniscient debugging, 211, 220
 Operating environments, reproducing, 84–86
 Oracle, defined, 379
 Oracle programs, 236
 Organizing problem information, 37
 Origin of failures, 1, 267
 Origins, *see* Tracking origins

P

Pareto's law, 343
 Patch, defined, 379
Pattern-Oriented Software Architecture Series, 73
 “People, Projects, and Patterns” WIKI, 246
 Personal software process, 360
 PHPBUGTRACKER, 45

- Physical influences in
 - problems, 88–89
 - PIN logging framework, 186–88, 203
 - Point cut, logging, 183–85
 - Post hoc ergo propter hoc, 273, 333
 - Practice of Programming*, 22
 - Presentation layer, 53–57, 71
 - testing at, 53–57
 - assessing results, 56–57
 - higher-level interaction, 55–56
 - low-level interaction, 53–55
 - system-level interaction, 55
 - Printf debugging, 176
 - Problem database, 31–32
 - advantages, 31
 - BUGZILLA problem database, 31–34
 - processing, 34–36
 - Problem environment, reproducing problems
 - control layer, 79
 - environment inputs, 77
 - local (programmer) environment, 76
 - user environment difficulties, 76
 - Problem lifecycle states
 - assigned, 35
 - closed, 36
 - new, 35
 - reopened, 36
 - resolved, 35
 - unconfirmed, 34–35
 - verified, 36
 - Problem list, difficulties using, 31
 - Problem report (PR) information
 - defined, 26, 379
 - expected behavior, 28
 - experienced behavior, 28
 - one-line summary, 28
 - operating environment, 29
 - problem history, 27
 - product release, 28
 - system resources, 29
 - Problems, 69, *see also*
 - Reproducing problems
 - defined, 18, 379
 - tracking, 25–44
 - classifying problems, 32–34
 - exercises, 46–47
 - managing duplicates, 39–40
 - managing problems, 31–32
 - overview, 25–26
 - processing problems, 34–36
 - relating problems and fixes, 40–43
 - reporting and tracking summary, 43–45
 - reporting problems, 26–31
 - requirements as problems, 37–39
 - tools for, 45–46
 - Production code checks, 242–44
 - Production process, fixing, 357–58
 - Program code coverage, 232
 - Program dependence graph, 156
 - Program execution layers, 71
 - Program slicing, *see also*
 - Backward slices;
 - Deducing errors;
 - Dynamic slicing;
 - Forward slices
 - overview, 147
 - summary, 170
 - Weiser, 171
 - Program states
 - capturing, 307–11
 - comparing, 311–22
 - determination, 313, 316
 - relevant, 312–16
 - PROLOG, 139, 198
 - PURIFY, memory error detection, 241, 246
- ## Q
- Quality assurance, errors during, 351–53
 - Querying events
 - overview, 196–99
 - uniform event queries, 197–99
 - watchpoints, 196–97
 - “Quick-and-dirty” process of debugging, 136–37, 144
- ## R
- Random input simplified, 118–19
 - Randomness
 - reproducing, 83–84
 - simplifying user interaction, 117–18
 - Reasoning about programs, 142–43
 - Recommendation systems, 356
 - Reference runs, 235–38
 - Regression testing, 49, 93, 236, 269
 - Reproducing problems
 - alternative interfaces (units), 91
 - capture/replay tool, 80–81
 - checkpoints, 102
 - communications, 82–83
 - creating test cases, 76
 - data, 80
 - debugging tool effects, 89–90
 - deterministic vs. nondeterministic, 78–90
 - exercises, 102–3
 - focusing on units, 91–97
 - control example, 92–95
 - mock objects, 95–96
 - setting up a control layer, 92
 - introduction, xix

Reproducing problems

(continued)

- mock object, 95–96
 - operating environment, 84–86
 - overview, 75
 - physical influences, 88–89
 - problem environment, 76
 - problem history, 76
 - randomness, 83–84
 - reasons for importance, 75
 - reproducing problem environment, 76–78
 - reproducing problem execution, 78–90
 - data, 80
 - effects of debugging tools, 89–90
 - operating environments, 84–86
 - overview, 78–80
 - physical influences, 88–89
 - randomness, 83–84
 - time of day, 83
 - user interaction, 80–82
 - reproducing system interaction, 90–91
 - schedules (multithreading), 86–88
 - summary of steps, 101
 - system-level interaction, 90–91
 - time element, 83
 - tools for, 101–2
 - unit layer (C subclassing), 91–97
- Requirements as problems, 37–39
- Result assessment, 51
- REVIRT (UMLinux), 91, 102
- Risk of change, 356
- Risky defects, 353

S

- Safari browser, 57–58
 - SCCB (Software change control board), 37, 44, 336
 - Schedules (multithreading), reproducing problems, 86–88
 - Schroedinbug, 90, 91
 - Scientific debugging
 - algorithmic debugging, 137–40
 - anomalies, 141
 - application of, 132–34
 - declarative debugging, 137–40
 - deductive reasoning, 142–43
 - deriving a hypothesis, 140–42
 - diagnosis, defined, 132
 - example, 132–34
 - exercises, 145–46
 - explicit debugging, 134–35
 - explicit problem statement, 134
 - inductive reasoning, 143
 - introduction, xix
 - keeping logbook, 135–36
 - Mastermind game example, 134–35
 - observation, 143
 - overview, 129
 - vs. quick and dirty debugging, 136
 - “quick-and-dirty” process, 136–37
 - reasoning about programs, 142–43
 - scientific method and debugging, 130–34
 - summary, 144
 - testing a hypothesis, 132–33
 - use of logbook, 135–36
 - Scientific method, defined, 379
 - Shell_sort debug example, to be debugged, 10
 - Showstopper!, 21, 33
 - Simple failure example, 1–2
 - Simplification algorithm, 112–17
 - Simplifying problems, 105–27
 - automatic simplification, 110–12
 - benefits of simplification, 107–8
 - circumstance, defined, 106
 - exercises, 124–27
 - faster ways, 119–22
 - caching, 119
 - isolating differences, not circumstances, 121–22
 - stopping early, 120
 - syntactic simplification, 120–21
 - Gecko example, 106–8
 - general method description, 106
 - manual simplification method, 109–10
 - overview, 105–6
 - random input simplified, 118–19
 - simplification algorithm, 112–17
 - simplifying user interaction, 117–18
 - tools for, 123
- Simplifying user interaction, 117–18
- Slice, defined, 157, 379
- Slice operations, 158–60
- Slicing programs
 - backward slices, 158, 159
 - chop operation, 158–59
 - dice operation, 160
 - executable slices, 160–61
 - forward slices, 157–58
 - introduction, 157
 - leveraging slices, 160
 - overview, 157
 - slice operations, 158–60

- Software change control
 - board (SCCB), 37, 44, 336
 - Software configuration management, 40–41
 - Software Configuration Management FAQ, 46
 - Software problem
 - lifecycle, 25–26
 - reporting problems, 26–31
 - Soul of a New Machine*, 21
 - SOURCEFORGE, 45
 - Sources of failures, 1
 - Specifications
 - assertions as, 232–33
 - defined, 379
 - errors during, 347–51
 - formal, 348
 - SPYDER debugger, 221
 - State, visualizing, 200–202
 - Statements, data
 - dependencies vs. control dependencies, 154–56
 - Static analysis
 - defined, 379
 - vs. dynamic analysis, 143
 - limits of, 166–70
 - Statistical debugging, 259–60
 - STRACE (Linux), 84–86, 92, 103
 - Strategy for locating arbitrary defects, 219–20
 - Surprise, defined, 379
 - Syntactic simplification, 120–21
 - System assertions, 238–42
 - avoiding buffer overflows with ELECTRICFENCE, 239
 - detecting memory errors with VALGRIND, 240–41
 - language extensions, 241–42
 - overview, 238–39
 - validating heap with MALLOC_CHECK_, 239
 - System-level interaction, 55
- ## T
- TARANTULA, anomaly detection tool, 256–57, 268
 - Terminology of bugs, faults, and defects, 18–19
 - Test cases
 - defined, 379
 - simplified, 107–8
 - Testing, 72
 - Testing, defined, 49, 379
 - Testing techniques, 49–74
 - exercises, 73–74
 - Gecko example, 50–53
 - isolating units, 63–66
 - overview, 49–50
 - preventing unknown problems, 69–70
 - testing at functionality layer, 57–59
 - testing at presentation layer, 53–57
 - assessing results, 56–57
 - higher-level interaction, 55–56
 - low-level interaction, 53–55
 - system-level interaction, 55
 - testing at unit layer, 59–63
 - testing for debugging, 49–50
 - testing for validation, 49
 - tools for, 72
 - Theory, defined, 379
 - Time bombs, C and C++, 239
 - Time of day, reproducing, 83
 - TRAC problem and version control system, 40–43, 45
 - Trace log data volume, 86
 - Tracking dependencies
 - affected statements, 153
 - effects of statements, 152–53
 - following dependences, 156
 - leveraging dependences, 156–57
 - overview, 152
 - statement dependences, 154–56
 - Tracking failures, xix
 - Tracking origins
 - dynamic slicing, 213–16
 - exercises, 221–22
 - exploring execution history, 211–13
 - leveraging origins, 216–19
 - overview, 211
 - reasoning backwards, 211
 - tools for, 221
 - tracking down infections, 219–20
 - Tracking problems, *see* Problems, tracking
 - TRAFFIC seven step debugging mnemonic
 - automate and simplify, 5, 9
 - correct the defect, 5, 13
 - find infection origins, 5, 9–12
 - focus on likely origins, 5, 12
 - isolate the infection chain, 5, 12–13
 - reproduce the failure, 5, 9
 - summary, 5, 20
 - track the problem, 5, 8
 - Transition, noninfected (sane) to infected state, 6, 7, 12
 - Transport layer, 53
- ## U
- UMLinux, 91
 - Uniform event queries, 197–99
 - Unit layer, 51, 59–63, 92–95

Unknown problems,
 prevention, 69–71
Unstructured control flow,
 152
User interaction,
 reproducing, 80–82
User simulation testing, 51

V

VALGRIND, memory error
 detection, 240–41,
 246–47
Validation, defined, 379
Value origins, 8, 147–48
VBSCRIPT, 57–58, 72

Verification, defined, 380
Version control
 software configuration
 management, 40, 46
 tags and branches, 40–41
Virtual machine, 55, 72, 91,
 101, 200
VULTURE tool, 354

W

Watchpoints, 196–97
WHYLINE system
 ALICE language, 216–19
 dynamic slicing, 213–16
 Ko and Myers, 216–19

Wilkerson, Daniel S., 123
Winrunner, 101
Workaround
 defined, 380
 examples, 338–39

X

XML, 59

Z

Z specification language,
 232, 247
Zeller, 112, 113, 117, 119,
 124, 290–93, 301–2,
 324, 327