

Index

A

- Alexander, Christopher, 5
- ambiguous requirements, 8
- architecture. *See* system architecture.
- archive mechanisms, 48
- assertions, 104
- automated builds
 - best practices, 207–209
 - unit testing, 156
- automated test engineer, 69
- automated test tools. *See also* tools.
 - application data management, 179
 - application incompatibility, 168
 - best practices, 175–176
 - building *vs.* buying, 167–169
 - candidates for, 173–174
 - choosing, 175–176
 - code-coverage analyzers, 163
 - code instrumentors, 163
 - cost, 174, 180–181
 - coverage, 174, 180
 - and development life cycle, 173
 - distraction from goals, 175–176
 - effects on testing, 171–176
 - efficiency, 174
 - GUI-testing, 164–165
 - help-desk problem reports,
 - 179–180
 - intrusiveness, 175
 - load performance, 165
 - memory-leak detection, 163
 - misconceptions, 172–176
 - network-testing, 164
 - number required, 172, 179
 - operating system incompatibility,
 - 167–168
 - overview, 159–160
 - predictability, 175
 - reducing test effort, 172–173
 - schedules, 173, 180
 - scope of testing, 177–178
 - specialized, 165

- stability requirements, 173
- stress testing, 165
- summary table of, 162
- and system architecture, 178–179
- test-data generators, 164
- test-management, 164
- test-procedure generators, 162–163
- testability hooks, 167
- testing on prototypes, 183–184
- training requirements, 174–175
- types of, 161–165
- usability-measurement, 163–164
- user group discussions, 177
- automated testing
 - best practices
 - automated builds, 207–209
 - black-box testing, 187–189
 - capture/playback tools, 187–189
 - data-driven frameworks, 197
 - developing test scripts, 197–200
 - establishing a start point, 194–195
 - hard-coded data values, 187–188
 - managing test results, 195
 - modular script development, 198
 - modular user-interface navigation, 198–199
 - non-modular scripts, 188
 - pre-built libraries, 199–200
 - regression testing, 201–205
 - reusability, 188–189
 - reusable functions, 199–200
 - smoke tests, 207–209
 - test-case management, 193–195
 - test harnesses, 191–195
 - version control, 200
 - candidates for, 113
 - designing tests, 113

B

- back-end testing vs. GUI, 31–32
- baselining requirements, 16
- beta products, testing, 41
- black-box testing
 - best practices, 187–189
 - definition, 91
 - vs. gray-box, 111
 - path analysis, 130
- boundary-value (BV) analysis, 44, 131
- budgets. *See also* costs.
 - automated test tools, 180–181
 - planning, 28, 40
- buffer overflow attacks, 225
- builds
 - automated, 156
 - planning, 60
 - unit test, 155–157
- business processing layer, 147
- BV (boundary-value) analysis, 44, 131

C

- capture/playback tools, 112, 187–189
- CCB (Change Control Board), 15, 16
- Change Control Board (CCB), 15, 16
- change management
 - baselining, 16
 - CCB (Change Control Board), 16
 - change-request forms, 16
 - communicating changes, 15–17
 - Engineering Review Board, 15
 - outdated documentation, 15–17
 - requirement-change process, 16
 - requirements-managing tools, 17
 - software defects, 16
 - tracking changes, 17
 - undocumented changes, 15

- change-request forms, 16
- code-coverage analyzers, 163
- code instrumentors, 163
- code layers, 146–149
- commercial off-the-shelf (COTS)
 - tools, 112
- compatibility testing, 235–237
- Competitive Engineering*, 5
- competitive evaluations, 222
- compiling, unit testing in local versions,
 - 155–157
- completeness of requirements, 6–7
- component testing. *See* unit testing.
- concurrency protection, 230–233
- concurrency testing, 229–233
- configuring logging, 105–106
- consistency of requirements, 7
- constraints, in test procedures, 135–137
- corporate culture, effects on testing, 27
- correctness of requirements, 6
- costs. *See also* budgets.
 - automated test tools, 174
 - early defect correction, 4
 - over development life cycle, 4
- COTS (commercial off-the-shelf)
 - tools, 112
- coverage. *See also* scope of testing.
 - automated test tools, 174, 180
 - code-coverage analyzers, 163
 - data flow, 43
 - planning tests, 25, 34
 - source code coverage analysis, 180
 - test, 25, 34
- critical requirements
 - designing tests, 124
 - mission-critical applications, 60
- customers. *See also* users.
 - environment, 48
 - expectations, 27
 - needs, 39

D

- data dictionaries, 43
- data-driven frameworks, 197
- data element details in test procedures,
 - 135–137
- data flow coverage, 43
- data for tests. *See* test data.
- data values, hard-coded, 187–188
- database abstraction layer, 146–147
- debug build, 104
- debug-information-level logging,
 - 105–106
- debug mode, 103–106
- debugging
 - with logs, 103–106
 - removing, 104–105
- defect trend analysis, 257
- Defect Workflow, 88
- defects
 - aging, 256
 - analyzing
 - Defect Workflow, 88–89
 - false negatives, 94–95
 - false positives, 94–95
 - fix-time to retest, 256
 - as measure of tester effectiveness,
 - 84–87
 - recurrence ratio, 257
 - system architecture, 93–95
 - density, 258
 - managing test results, 195
 - prevention, requirements phase, 3
 - tracking life cycle
 - closure, 250–253
 - defect attributes, 248–249
 - defect categories, 251
 - defect workflow, 252–253
 - documenting defects, 248–249
 - granularity, 249
 - overview, 247–248

- prioritization, 249–250
 - reoccurrence, 250
 - retest status, 247
 - design documentation, 43
 - design standards, 115–119
 - designing tests. *See also* planning tests; test procedures.
 - automation, 113
 - black-box vs. gray-box, 111
 - critical requirements, 124
 - design standards, 115–119
 - divide-and-conquer strategy, 110–114
 - exploratory testing, 139–141
 - high-risk requirements, 124
 - methodology, 110
 - nonfunctional tests, 118–119
 - personnel, 110
 - procedure templates, 115–119
 - prototyping, 127–128
 - reviewing the design plan, 123
 - scope of testing, 109
 - techniques for, 129–133
 - test-case scenarios, 129–133
 - test cases, 121–124
 - test data, 114
 - test harnesses, 111–112
 - test sequence, 109–110, 123
 - testing techniques, 112, 129–133
 - tools, 112
 - without requirements, 111
 - developer-to-engineer ratio, 52–53
 - development
 - based on existing systems, 19–22
 - development process, 22
 - documenting existing application, 21
 - documenting updates, 21–22
 - estimating resources, 20
 - managing expectations, 21–22
 - “moving target” environment, 20
 - overview, 19–20
 - pros and cons, 20
 - using fixed versions, 21
 - unit testing, 144–149
 - Development Ratio Method, 52–53
 - divide-and-conquer strategy, 110–114
 - documentation. *See also* requirements.
 - change management, 15–17
 - defect tracking, 248–249
 - existing systems, 21–22
 - functional requirements, 6
 - living documents, 125–126
 - nonfunctional requirements, 7, 216
 - requirements
 - outdated documentation, 15–17
 - undocumented changes, 15
 - use cases, 6
 - test procedures
 - constraints, 135–137
 - designing, 11–14
 - documenting, 117–118, 125–126
 - estimating number of, 54–56
 - and team size, 54–56
 - domain object layer, 147
- E**
- early involvement
 - requirements phase, 3–4
 - test planning, 27
 - effort level, estimating
 - factors affecting, 58–60
 - importance of, 28
 - number of tasks, 56–58
 - number of test procedures, 54–56
 - Task Planning method, 56–58
 - Test Procedure method, 54–56
 - engineer-to-developer ratio, 52–53
 - Engineering Review Board, 15

equipment purchase list, 49
 equivalence partitioning, 130
 error cases, 149
 error-level logging, 105–106
 estimating
 based on existing systems, 20
 effort level
 factors affecting, 58–60
 importance of, 28
 number of tasks, 56–58
 number of test procedures, 54–56
 Task Planning method, 56–58
 Test Procedure method, 54–56
 personnel hours, 56–58
 team size
 developer to engineer ratio, 52–53
 Development Ratio Method, 52–53
 number of tasks, 56–58
 number of test procedures, 54–56
 Project Staff Ratio method, 53–54
 Task Planning method, 56–58
 Test Procedure method, 54–56
 total project requirements, 53–54
 time
 Task Planning method, 56–58
 WBS (work breakdown structure), 52
 exceptions, 149
 experience levels of testers, 76
 exploratory testing, 139–141
 extreme programming, 151

F

false negatives, 94–95
 false positives, 94–95
 feasibility of requirements, 7
 features, prioritizing, 39–40
 field studies, 222

focus groups, 221–222
 functional analysis, 130
 functional test tools, 112, 187–189
 functional vs. nonfunctional
 requirements, 26–27
 testing, 82–83
 functions, reusing, 199–200. *See also* methods.

G

Gilb, Tom, 5
 global nonfunctional constraints, 216
 gray-box testing
 vs. black-box, 111
 definition, 41, 91
 problem types, 94–95
 GUI testing. *See also* user interface.
 automated test tools, 164–165
 vs. back-end testing, 31–32

H

hard-coded data values, 187–188
 hardware requirements, 48
 help-desk problem reports, 179–180
 high-risk requirements, 124

I

in-bounds data, 131
 interface-based unit testing, 152
 iterations, 13
 iterative development process, 13

K

keystrokes, recording, 164–165

L

libraries, reusable functions, 199–200
 living documents, 125–126
 load performance tools, 165
 lock acquisition, 232
 lock enforcement, 232
 lock release, 232
 locking data, 230–233
 log contents, 99–101
 logging. *See also* tracking test execution.
 configuring, 105–106
 debug-information level, 105–106
 as debugging tool, 103–106
 error level, 105–106
 increasing testability, 99–101
 source code, inspecting, 103

M

maintainability, test procedures, 135–137
 managing test results, 195
 manual test engineer, 68
 memory-leak detection tools, 163
 methods, 99. *See also* functions.
 mission-critical applications, 60
 “moving target” environment, 20
 multi-user data access, 229–233

N

necessity of requirements, 7
 negative testing, 131
 network characteristics, 48
 network test engineer, 69
 network-testing tools, 164
 nonfunctional requirements
 documentation, 7, 216
 vs. functional, 26–27
 nonfunctional testing
 compatibility testing, 235–237

 concurrency testing, 229–233
 designing, 118–119
 examples, 214–215
 vs. functional, 82–83
 global nonfunctional constraints, 216
 identifying the target audience, 221–223
 locking data, 230–233
 multi-user access, 229–233
 performance testing, 217–219
 planning, 213–216
 risks, 215–216
 security, 225–227
 test data, 217–219
 usability testing, 221–223
Notes On the Synthesis of Form, 5

O

on-bounds data, 131
 online resources
 automated tools, 198
 user group discussions, 177
 optimistic locking, 230–233
 orthogonal array testing, 112
 orthogonal arrays, 132–133
 out-of-bounds data, 131

P

pass/fail unit testing criteria, 157
 patches, 42
 path analysis, 130
 performance testing, 217–219
 personnel. *See also* teams.
 designing tests, 110
 estimating hours, 56–58
 required, 33, 40
 pessimistic locking, 230–233
 phased solutions, 28
 phases of testing, 34

- Planguage, 5
- planning tests. *See also* designing tests; test procedures.
 - archive mechanisms, 48
 - assumptions, 31
 - beta products, 41
 - budget, 28, 40
 - complexity, 39
 - corporate culture, 27
 - coverage, 25, 34
 - customer environment, 48
 - customer expectations, 27
 - customer needs, 39
 - defects, 42
 - design techniques, 33
 - developing test harnesses or scripts, 33
 - early involvement, 27
 - environment, 47–49
 - equipment purchase list, 49
 - estimating effort level
 - factors affecting, 58–60
 - importance of, 28
 - number of tasks, 56–58
 - number of test procedures, 54–56
 - Task Planning method, 56–58
 - Test Procedure method, 54–56
 - estimating personnel hours, 56–58
 - estimating team size
 - developer to engineer ratio, 52–53
 - Development Ratio Method, 52–53
 - number of tasks, 56–58
 - number of test procedures, 54–56
 - Project Staff Ratio method, 53–54
 - Task Planning method, 56–58
 - Test Procedure method, 54–56
 - total project requirements, 53–54
 - expertise required, 33
 - features, prioritizing, 39–40
 - functional vs. nonfunctional requirements, 26–27
 - goals, 25–29
 - gray-box testing, 41
 - GUI vs. back-end testing, 31–32
 - hardware requirements, 48
 - mission-critical applications, 60
 - network characteristics, 48
 - nonfunctional, 213–216
 - number of builds, 60
 - patches, 42
 - personnel required, 33, 40
 - phased solutions, 28
 - phases, 34
 - pre-release products, 41
 - prerequisites, 31
 - process definition, 60
 - production environment, 47–49
 - release criteria, 25, 34
 - results expectations, 27
 - results of previous tests, 28
 - reviewing the plan, 123
 - risks
 - assessing, 36–37
 - factors affecting, 35–36
 - mitigating, 36–37
 - prioritizing, 35, 39
 - schedules, 28, 34, 59, 60
 - service packs, 42
 - software issues, 41–42
 - software requirements, 48
 - solution type, 28
 - staged implementation, 42
 - strategies, 26, 31–34
 - system architecture, 31
 - technology choices, 28, 41–42
 - test data
 - breadth, 45
 - BV (boundary-value) analysis, 44, 131
 - conditions tested, 46
 - data dictionaries, 43

- data flow coverage, 43
- depth, 44
- design documentation, 43
- in-bounds data, 131
- integrity, 45
- on-bounds data, 131
- out-of-bounds data, 131
- preparation, 46
- representative data samples, 133
- scope, 45
- time estimation
 - Task Planning method, 56–58
 - WBS (work breakdown structure), 52
- tools, 33, 59
- understanding the requirements, 26–29
- pre-release products, testing, 41
- prioritization of requirements, 8
- prioritizing
 - features, 39–40
 - requirements, 8
 - risks, 35, 39
- production environment, 47–49
- progress tracking, 255–258
- Project Staff Ratio method, 53–54
- prototyping
 - designing tests, 127–128
 - testing automated test tools, 183–184
- publications
 - Competitive Engineering*, 5
 - Notes On the Synthesis of Form*, 5
- pure unit testing, 144

Q

- quality checklist, 6–9
- quality measures, 5

R

- Rational Software Corporation, 35
- recording keystrokes, 164–165
- regression testing
 - automated, best practices, 201–205
- release criteria, 25, 34
- release mode, 103–106
- reporting defects. *See* defects, analyzing.
- representative data samples, 133
- requirement-change process, 16
- requirements. *See also* documentation.
 - ambiguity, 8
 - based on existing systems
 - development process, 22
 - documenting existing application, 21
 - documenting updates, 21–22
 - estimating resources, 20
 - managing expectations, 21–22
 - “moving target” environment, 20
 - overview, 19–20
 - pros and cons, 20
 - using fixed versions, 21
 - completeness, 6–7
 - consistency, 7
 - correctness, 6
 - definition, 6
 - documenting, 6
 - feasibility, 7
 - functional vs. nonfunctional, 26–27
 - iterative development process, 13
 - necessity, 7
 - prioritization, 8
 - quality checklist, 6–9
 - quality measures, 5
 - reward/penalty technique, 8
 - testability, 3, 7, 12–13
 - traceability, 8–9

- understanding, 26–29
- use cases, 6
- verifiability, 7
- verifying, 5–9
- waterfall development model, 13
- requirements, nonfunctional
 - documentation steps, 7
- requirements-managing tools, 17
- requirements phase
 - change management
 - baselining, 16
 - CCB (Change Control Board), 16
 - change-request forms, 16
 - communicating changes, 15–17
 - Engineering Review Board, 15
 - outdated documentation, 15–17
 - requirement-change process, 16
 - requirements-managing tools, 17
 - software defects, 16
 - tracking changes, 17
 - undocumented changes, 15
 - defect prevention, 3
 - designing test procedures, 11–14
 - tester involvement, 3–4
- Requirements-Services-Interfaces (RSI)
 - unit testing, 152
- results. *See* defects.
- reviewing plans, 123
- reward/penalty requirements
 - technique, 8
- risks
 - assessing, 36–37
 - factors affecting, 35–36
 - high-risk requirements, 124
 - mitigating, 36–37
 - nonfunctional tests, 215–216
 - prioritizing, 35, 39
- RSI (Requirements-Services-Interfaces)
 - unit testing, 152

S

- scenarios
 - designing tests, 129–133
 - spreadsheets, 136
- schedules
 - automated test tools, 173, 180
 - entrance criteria, 241–243
 - establishing a start point, 194–195
 - exit criteria, 241–243
 - planning, 28, 34, 59, 60
- scope of testing. *See also* coverage.
 - automated test tools, 177–178
 - designing tests, 109
 - test data, 45
- scripts
 - developing, 33
 - developing, best practices, 197–200
 - modular development, best practices, 198
 - non-modular, best practices, 188
 - recording keystrokes, 164–165
- security
 - buffer overflow attacks, 225
 - locking data, 230–233
 - nonfunctional tests, 225–227
- security test engineer, 70
- service packs, 42
- SMEs (subject-matter experts)
 - identifying the target audience, 221
 - role in test team, 75–76
- smoke tests, 204, 207–209
- software builds
 - automating, 156
 - best practices, 207–209
 - build process, 156
 - debug build, 104
 - planning, 60
 - sequence of events, 208–209
 - unit test, 155–157

- software issues, 41–42
- software requirements, 48
- source code, inspecting, 103
- source code coverage analysis, 180
- staged implementation, 42
- strategies, 26, 31–34
- strength, orthogonal arrays, 133
- stress-testing tools, 165
- stubbed components, 144, 152–154
- subject-matter experts (SMEs)
 - identifying the target audience, 221
 - role in test team, 75–76
- surveys, 222
- system architecture
 - automated test tools, 178–179
 - debug mode, 103–106
 - defect reporting, 93–95
 - overview, 91
 - release mode, 103–106
 - risk analysis, 31
 - testability, 97–98
 - underlying components, 93–95

T

- Task Planning method, 56–58
- teams. *See also* testers.
 - early involvement, 3–4, 27
 - effectiveness, evaluating
 - attention to detail, 84
 - defect analysis, 84–86, 87, 88–89
 - development phase, 84
 - experienced testers *vs.* novices, 82
 - feedback, 87
 - following instructions, 84
 - functional *vs.* nonfunctional testing, 82–83
 - overview, 64, 79–80
 - self evaluations, 86–90
 - setting goals, 80–81

- SMEs *vs.* technical experts, 81–82
- tester comments, 88–89
- testing phase, 83–84
- roles and responsibilities
 - automated test engineer, 69
 - experience levels, 76
 - manual test engineer, 68
 - network test engineer, 69
 - sample team structure, 71–73
 - security test engineer, 70
 - skills mix, 75–77
 - SMEs (subject matter experts), 75–76
 - technical expertise, 76
 - technical leader, 67
 - technical testers, 76
 - test environment specialist, 70
 - test library and configuration specialist, 70
 - test manager, 66
 - usability test engineer, 68
 - work packages, 65–66
- size, estimating
 - developer to engineer ratio, 52–53
 - Development Ratio Method, 52–53
 - number of tasks, 56–58
 - number of test procedures, 54–56
 - Project Staff Ratio method, 53–54
 - Task Planning method, 56–58
 - Test Procedure method, 54–56
 - total project requirements, 53–54
- technical expertise of testers, 76
- technical leader, 67
- technical testers, 76
- techniques
 - definition, 193–194
 - for designing tests, 112, 129–133
- techniques for designing tests, 129–133
- technology choices, 28, 41–42
- templates for test procedures, 135–137

- test cases
 - designing tests, 121–124
 - managing, 193–195
 - scenarios
 - designing tests, 129–133
 - spreadsheets, 136
- test data
 - breadth, 45
 - BV (boundary-value) analysis, 44, 131
 - conditions tested, 46
 - data dictionaries, 43
 - data flow coverage, 43
 - depth, 44
 - design documentation, 43
 - designing tests, 114
 - in-bounds data, 131
 - integrity, 45
 - nonfunctional tests, 217–219
 - on-bounds data, 131
 - out-of-bounds data, 131
 - preparation, 46
 - randomly generated, 218
 - representative data samples, 133
 - scope, 45
- test-data generators, 164
- test environment
 - capacity, 246
 - change management, 245
 - isolating from development
 - environment, 245–246
 - operating environment, 246
 - performance, 246
 - removable disks, 246
 - shared labs, 246
 - version management, 245–246
- test environment specialist, 70
- test-harness adapters, 192
- test harnesses
 - best practices, 191–195
 - designing tests, 111–112
 - developing, 33
- test library and configuration
 - specialist, 70
- test-management tools, 164
- test manager, 66
- test plan. *See* designing tests; planning tests.
- test-procedure generators, 162–163
- Test Procedure method, 54–56
- test procedures. *See also* designing tests; documentation; planning tests; requirements.
 - constraints, 135–137
 - data element details, 135–137
 - designing, 11–14
 - documenting, 117–118, 125–126
 - entrance criteria, 241–243
 - estimating number of, 54–56
 - exit criteria, 241–243
 - as living documents, 125–126
 - maintainability, 135–137
 - standard contents, 117–118
 - and team size, 54–56
 - templates, 115–119, 135–137
 - test-scenario spreadsheets, 136
- test results. *See* defects.
- test-scenario spreadsheets, 136
- test sequence
 - designing tests, 109–110, 123
- test shells, 204
- test teams. *See* teams.
- testability
 - increasing with logs, 99–101
 - verifying, 97–98
- testability hooks, 167
- testability of requirements, 3, 7, 12–13
- testers. *See also* teams.
 - early involvement, 3–4, 27
 - technical expertise, 76
- testing. *See also* designing tests; planning tests; test procedures.
 - automated test tools on prototypes, 183–184

- based on existing systems
 - development process, 22
 - documenting existing application, 21
 - documenting updates, 21–22
 - estimating resources, 20
 - managing expectations, 21–22
 - “moving target” environment, 20
 - overview, 19–20
 - pros and cons, 20
 - using fixed versions, 21
- coverage. *See also* scope of testing.
 - automated test tools, 174, 180
 - code-coverage analyzers, 163
 - data flow, 43
 - planning tests, 25, 34
 - source code coverage analysis, 180
 - test, 25, 34
- techniques
 - definition, 193–194
 - for designing tests, 112, 129–133
- testing quality into a product, 26
- time estimation
 - Task Planning method, 56–58
 - WBS (work breakdown structure), 52
- tools. *See also* automated test tools.
 - capture/playback, 112, 187–189
 - COTS (commercial off-the-shelf), 112
 - debugging, 103–106
 - designing tests, 112
 - enterprise level, 207
 - functional test, 112, 187–189
 - planning tests, 33, 59
 - process management, 207
 - requirements-managing, 17
 - in test design, 112
- traceability of requirements, 8–9
- tracking test execution, 255–258. *See also* logging.
- training requirements, automated test tools, 174–175

U

- unit-test frameworks, 157
- unit testing
 - automated builds, 156
 - in the build process, 155–157
 - business processing layer, 147
 - code layers, 146–149
 - compiling in local versions, 155–157
 - database abstraction layer, 146–147
 - in the development process,
 - 144–149
 - domain object layer, 147
 - error cases, 149
 - before implementation, 151–154
 - interface-based approach, 152
 - missing components. *See* stubbed components.
 - overview, 143–144
 - parallel to implementation, 151–154
 - pass/fail criteria, 157
 - path analysis, 130
 - pure, 144
 - RSI (Requirements-Services-Interfaces)
 - approach, 152
 - standardization, 156–157
 - stubbed components, 144, 152–154
 - unit-test frameworks, 157
 - user interface layer, 147
 - user interfaces *vs.* component or software interfaces, 152
- usability
 - definition, 68
 - identifying the target audience, 222
 - measurement tools, 163–164
 - testing, 221–223
- usability test engineer, 68
- use cases, 6
- user interface
 - vs.* component or software interfaces, 152

- GUI testing
 - automated test tools, 164–165
 - vs. back-end testing, 31–32
- modular navigation, best practices, 198–199
- user interface layer, 147
- users. *See also* customers.
 - identifying the target audience, 221–223
 - needs analysis
 - field studies, 222
 - focus groups, 221–222
 - online group discussions, 177
 - surveys, 222

V

- verifying requirements, 5–9
- version control, 200

W

- waterfall development model, 13
- white-box testing, 130, 187–189
- work packages, 65–66
- wrap up, 188
- wrappers, 204