

## INDEX

- Acceptance testing, 88
- Action clause, 54
- Adaptive maintenance, 48
- All-c-use/some p-use testing, 32–34
- All-c-use testing, 34–36
- All-definition testing, 34
- All-du-path testing, 29–31, 107
- All-p-use/some c-use testing, 32
- All-p-use testing, 34
- All-use testing, 31–32
- Antecedents, 197
- Anticomposition axiom, 87
- Antidecomposition axiom, 87
- Arithmetic operator replacement, 43
- Array elements, 172–175
- Array reference replacement:
  - array reference for, 43
  - for constant replacement, 42
  - for scalar replacement, 42
  - scalar, 42
- Assertion checking:
  - characteristics of, 166–167
  - defined, 213
  - global assertions, 168
  - local assertions, 167
  - monitors, 168
- Assignment statement:
  - static analysis, 138, 152, 159
  - symbolic trace analysis, 110–113, 129–130
- Atomic formula, 201
  
- Backward substitution, 102, 213
- Basis set, 25–26
- Boolean functions, 68
- Bottom-up integration, 83, 88
- Boundary-interior testing, 213
- Boundary-interior testing method, 23–24
- Boundary test, defined, 23
- Boundary-value analysis, 70–71, 73, 91, 213
  
- Branching constructs, specification-based
  - test-case selection methods, 55–56
- Branch testing:
  - components of, 21–22
  - defined, 213
  
- Calculus:
  - first-order predicate, 199–206
  - propositional, 194–199
- Candidate paths:
  - data-flow testing, 31
  - symbolic trace analysis, 105, 108
- Cause-effect graphing, 54 n.1, 68
- C/C++ programming language, 2, 4, 10, 17, 21, 23, 44, 56, 85, 113–114, 117–125, 133, 181, 183–184
- Class, object-oriented testing, 85, 88
- Closed border segment, domain-strategy testing, 37–38
- Code-based test-set selection method:
  - branch testing, 21–26
  - characterized, 15–16, 76
  - data-flow testing, 26–36, 51
  - domain-strategy testing, 36–39
  - defined, 12
  - fault seeding, 45–46
  - path testing, 16–17
  - program mutation, 39–45
  - statement testing, 17–21
- Code inspection:
  - defined, 146, 213
  - inspection program log, 149
  - objectives, 146, 151–152
  - procedure, 147
  - record of changes made as result of, 150
  - summary report, 147–148
  - termination of, 151
- Code Inspection Summary Report, 148
- Compact function ( $f$ ), 10–11

254 INDEX

- Compiler object-code optimization, 136
- Component:
  - defined, 213
  - in test-case selection, 9
  - test-case selection methods, 11–12
- Composite path, 159
- Compound statement, trace subprogram, 188, 192
- Computational coupling coefficient, 7–8
- Computationally coupled inputs, 214
- Computation fault, 11, 214
- Concatenation, 23, 109, 130
- Concepts, 4–8
- Concurrency, data-flow anomalies, 136
- Conditional probability, 6–7, 15, 214
- Conditional statements, 152, 157
- Condition clause, specification-based test-case selection methods, 54
- Condition statement, trace subprogram, 184–185, 189
- Consequences, 197
- Constant replacement:
  - array reference, 42
  - defined, 42
- Constraints, symbolic trace analysis:
  - concatenation of, 130
  - moving/simplifying rules, 99–110
  - state constraints, 96–99, 109–110
  - supporting software, 127–129
- Contingent wff, 196
- Contradictory wff, 196
- Contributing components, test-set selection principles, 8–9
- Control flow:
  - diagram, 12
  - graph, static analysis, 136
  - symbolic trace analysis, 100, 102
- Correctness proof, construction of:
  - characteristics of, 152–153
  - inductive assertion method: top-down approach, 156–161
  - predicate transformation method: bottom-up approach, 153–156
- Cost-effectiveness, 214
- Coupling coefficient, 7–8, 16, 214
- C-up**, 126–131
- c-use, 214
- Cyclomatic number, 214
  
- Data-flow analysis, 130
- Data-flow anomaly:
  - characteristics of, 133
  - defined, 214
  - detection, *see* Data-flow-anomaly detection
- Data-flow-anomaly detection:
  - array elements, 172–175
  - cost-benefit analysis, 181
  - components of, 134–137, 69–171, 179–180
  - execution paths, 175–177
  - input data selection, 175–179
  - present method, 179–180
  - by static analysis, 172, 174
- Data flow, defined, 214
- Data-flow testing:
  - all-c-use/some p-use testing, 32–34
  - all-c-use testing, 34–36
  - all-definition testing, 34
  - all-du-path testing, 29–31
  - all-p-use testing, 34
  - all-p-use/some c-use testing, 32
  - all-use testing, 31–32
  - characterized, 26–27, 51, 109
  - components of, 27–29
  - defined, 215
- DATA statement, 43
- Data statement alteration, 43
- dd-type anomaly, 135
- Debug testing:
  - cost-benefit analysis, 15–16, 89
  - defined, 215
  - effectiveness of, 81
  - essential components of, 16
  - fault discovery, 5–6, 15
  - goals/purpose of, 5–6, 16
  - large software systems, 85
  - operational testing compared with, 81–82
  - program costs, 13
  - reliability, 81–82
  - test set construction, 14
  - value of, 15
- Decision tables, 54 n.1, 62, 64, 67, 215
- Decision-to-decision path, 22
- Decomposition:
  - subfunction testing, 55, 58
  - symbolic trace analysis, 114
- Define-define (DD) state, 179
- Define-undefine (DU) state, 179
- Definition-clear path, data-flow testing, 29
- $\Delta R$ , 89
- Development team, functions of, 73
- Differentiation, program mutation, 41, 45
- Directed graphs, 208–212
- Documentation, code inspection, 148–150
- Domain fault, 10–11, 215
- Domain predicate, subfunction testing, 56, 59–61, 63–68. *See also* Predicate testing
- Domain-strategy test, 36–39, 215

- DO statement, trace subprogram, 186, 190
- Du path, defined, 29. *See also* All-du-path testing
- Dynamic analysis, 136
  
- Effective test set, 90
- Efficient test set, 90
- Equivalence partitioning, 215
- Equivalence transformation, 131
- Equivalent mutant, 215
- Error guessing, 6, 55, 71–73, 91, 215
- Error seeding, 215
- Event sequencing, errors in, 132–133, 136
- Exhaustive test, 1–2
- EXIT statement, trace subprogram, 188, 192
- Expression statement, trace subprogram, 184, 189
- Extremal clause, 23
  
- Failure set, 5
- False alarms, 174
- Fault detection, subfunction testing, 64
- Fault discovery:
  - branch testing, 26
  - capability, 215
  - domain-strategy testing, 37
  - operational testing, 80
  - predicate testing, 69
  - significance of, 15, 91
  - statement testing, 21
- Faults:
  - classification of, 10–11
  - defined, 2
  - detection/discovery of, 5–6, 9
  - latent, *see* Latent faults
- Fault seeding, 45–46
- First-order predicate calculus, 199–206
- Flowcharts, inductive assertion, 159–160
- Flow graph, 86
- Formal proof, 1
- “For” statement, 132
- FOR statement, trace subprogram, 186–187, 190
- FORTRAN programs, 43, 133, 139, 180
- Function plot, 18
  
- Geometrical analyses, domain-strategy testing, 37
- “Goto” statement:
  - characterized, 132, 152
  - replacement, 43
- GOTO statement, trace subprogram, 188, 192
- Graphic user interface, code-based test-selection methods, 49
- Graph theory, branch testing, 26
  
- Hardware, input domain, 4
- “Hit,” 5
- Howden branch testing method. *See* Boundary-interior branch testing method
  
- Ideal set of test cases, 77
- Ideal test sets, 77–79
- If-then-else statement, 154
- “If” statement:
  - implications of, 189
  - subfunction testing, 58
  - symbolic trace analysis, 96, 115, 121
- Induction hypothesis, 207
- Induction proposition, 207
- Inductive assertion, 152, 156–161
- Inductive clause, 23–24
- Industrial applications, 47
- Inference rules, 197
- Inheritance, object-oriented tests, 87
- Initialization clause, 23–24
- Input (data):
  - conditions, MEP, 69
  - defined, 4
- Input domain:
  - defined, 4, 215
  - error guessing, 71
  - partitioning, 55, 57–64, 66
  - path testing, 16–17
  - program mutation, 40
- Inspection program log, 149
- Institute of Electrical and Electronics Engineers (IEEE), 2–3, 5
- Instrumentor, symbolic trace analysis, 126–131
- INST, trace subprogram 189–192
- INTDIV program, 153–156, 158–160
- Integration testing, 82–85, 88
- Interactive program testing, 4, 41
- Interface flaws, 133
- Interior test, defined, 23
- Interpretation of wffs, 202
- Iterative statements, 154
  
- Labeled statement, trace subprogram, 188, 192
- Latent faults, 6, 81
- Legacy software, 122
- Lemmas, inductive assertions, 159–160
- Linear combination path, 25
- Linearly independent path, 25–26, 216
- Linear programming, 48
- Logic:
  - design, 67–68
  - errors, 146
- Logical connector replacement, 43

256 INDEX

- Logical consequences, 197
- Logically equivalent wffs, 197, 204
- Logico-mathematical background:
  - directed graphs, 208–212
  - first-order predicate calculus, 199–206
  - path descriptions, 208–212
  - principle of mathematical induction, 206–209
  - propositional calculus, 194–199
- Loop constructs:
  - boundary-interior branch testing method, 23–24
  - missed-by-one error, 71
  - specification-based test-case selection methods, 55–56
  - static analysis, 136–137, 157
  - symbolic traces, 114, 124
- Loop-free path, data-flow testing, 29
- Loop invariant, 216
- Lower bound (LB), 70–71, 91
  
- Mathematical induction principle, 206–209
- Maximal probability, 26
- McCabe's cyclomatic number, 24, 26
- McCabe's test method:
  - characteristics of, 24–26, 214
  - defined, 216
- Memoryless program, 4, 216
- Message graph, 86
- Message, object-oriented testing, 86
- Method of equivalence partitioning (MEP), 69
- Military applications, 47
- "Miss," 5
- Missed-by-one error, 71
- Missing-path fault, 11
- Multiple test cases, 5–6
- Mutant, defined, 216
- Mutant tests, 45
  
- Nested loops, static analysis, 132, 159
- Notation, 4–8
- Null statement, trace subprogram, 188
  
- Object-oriented programs, 84–88
- Off test point, domain-strategy testing, 37–38
- On test point, domain strategy testing, 37–38
- Open border segment, domain-strategy testing, 37–38
- Operand, symbolic trace analysis, 127
- Operational profile, 80–81, 90, 216
- Operational testing, 12–13, 76, 80–82, 215–216
- Optimal test set, 9, 216
- Oracle, 5, 216
- Output domain, 215–216
  
- Padding process, symbolic trace analysis, 122
- Parsing, 130
- Partitions/partitioning:
  - defined, 11
  - subfunction testing, 55, 57–66
- Path, generally:
  - descriptions, 208–212
  - matrix, data-flow testing, 31
  - testing, 9, 16–17, 216
- pem, trace subprogram, 189
- Phased integration, 84
- Postcondition, 216
- Power set, 78
- Precondition, 152, 216
- Predicate:
  - defined, 216
  - testing, 68–70, 73, 91, 216–217
  - transformation, 152–156, 161
- Prenex normal form, wffs, 204–205
- Principle of test-case selection, defined, 217
- Probability distribution, 76, 80–81
- Procedural language, 84–85
- Process innovation, 91
- Process optimization, 91
- Program analysis, 217
- Program correctness, 217
- Program creator, code inspection, 146–147
- Program failures, 82
- Program graph:
  - applications, 17–18, 21–22
  - branch testing, 24–26
  - code-based test-selection methods, 49
  - data-flow testing, 27–28
  - defined, 217
  - symbolic trace analysis, 94–96, 102–103
- Program instrumentation:
  - applications of, 163–164, 179
  - assertion checking, 166–169
  - data-flow-anomaly detection, 169–181
  - defined, 217
  - statement testing, 20
  - test-case effectiveness assessment, 165–166
  - test-coverage measurement, 164–165
  - trace-subprogram generation, 181–192
- Programming errors, static analysis of, 132–133
- Programming fault, 10–11
- Programming language, 10, 48, 55–56, 71, 146.
  - See also* C/C++ programming language;
  - FORTRAN programs
- Programming style, 72
- Program mutation:
  - defined, 39, 217

- mutant construction, 42
- mutants, types of, 40–45
- statement testing, 20
- test, 217
- Program slicing, 137, 217
- Program state, defined, 97
- Program testing:
  - costs, 12–13
  - defined, 217
- Program-trouble report, 147, 151
- Program with memory, 217
- Propositional calculus, 194–199
- Propositional variable, 195, 197
- p-use, defined, 217
  
- Random testing, 12
- Real-time programs, 2, 13
- Record of Changes Made as Result of Code Inspection, 150
- Redundant statements, 111–112
- Reformatting, subfunction testing, 57–58
- Regression test, 88, 218
- Reliability-critical software system, 147
- Reliability of test-case selection criterion, 218
- Repetitive statements, 152
- Restrictive clause, 97
- “Return” statement, 43, 132
- RETURN statement, trace subprogram, 188, 191
  
- Safety-critical programs, 47, 82
- Sandwich integration, 84
- Satisfiable wff, 202
- Scalar replacement:
  - array reference for, 42
  - constant for, 42
  - defined, 42
- Schedule, data-flow anomalies, 136
- Scope, first-order predicate calculus, 201–202
- S-down**, 126–131
- Semantic modifiers, symbolic trace analysis, 99
- Sentinel forms, 200
- Sherer’s theory, 89
- Shortest execution path, symbolic trace analysis, 102–104
- Short integer, 4
- Simple path, data-flow testing, 29
- Simplify**, symbolic trace analysis, 126–131
- Single test case, 5–6
- Singularity index, 218
- skip statement, 153–154
- Slicing criterion, 142–143, 145
  
- Software:
  - audition, 152
  - counter, 218
  - reliability, 47
  - testing, *see* Software testing
- Software testing:
  - characterized, 76–77
  - cost-benefit analysis, 76–77, 89
  - ideal test sets, 77–79
  - importance of, 76
  - integration testing, 82–84
  - object-oriented programs, 84–88
  - operational testing, 76, 80–82
  - regression testing, 88
  - test-case selection criterion, choosing, 90–92
- Source code:
  - functions of, 6
  - statement test, 13
  - static analysis, 132
  - test-case selection and, 8
  - test-set selection, 11–12
  - transformation, 102
- Source constant replacement, 42
- Specialization, object-oriented tests, 87
- Specification-based test-set selection method:
  - boundary-value analysis, 70–71, 73
  - clauses, types of, 54
  - components of, 53–54, 72–73, 76
  - defined, 12–13, 53
  - error-guessing, 71–73
  - predicate testing, 68–70, 73
  - subfunction testing, 55–68, 73
  - theoretical significance of, 54–55
- State constraints:
  - defined, 96
  - properties of, 97–99, 109–110
  - scope of, 100
  - tautological, 110
- State-transition:
  - diagram, 169, 179
  - function, 170, 175
  - graph, static analysis, 135
- Statement analysis, program mutation, 43
- Statement-coverage test, 44
- Statement deletion, program mutation, 43
- Statements, symbolic trace analysis:
  - rules for moving/simplifying, 110–114
  - supporting software, 127–130
- Statement test/testing:
  - defined, 12, 218
  - components of, 13, 17–21, 91

## 258 INDEX

- Static analysis:
  - code inspection, 133, 146–152
  - data-flow anomaly detection, 133–137
  - defined, 132, 218
  - program slicing, 133, 137, 141–146
  - proving programs correct, 152–161
  - purpose of, 132–134
  - symbolic evaluation (execution), 133, 137–141
  - termination of, 156
- Stopping a test, criteria for, 88–89
- Strongly connected graphs, 24
- Structural flaws, static analysis of, 132–133
- Stubs:
  - defined, 54 n.1
  - top-down integration, 84
- Subcase fault, 11
- Subdomain-based test-set selection method, 12
- Subdomain, test-set selection, 10–11
- Subfunction testing, 12, 55–68, 73, 218
- Subprogram:
  - relation, 99
  - specification-based test-set selection method, 54–55
- Successful test:
  - characterized, 77–79
  - defined, 5, 218
- SWITCH statement, trace subprogram, 187–188, 191
- Symbolic evaluation (execution), 137–141
- Symbolic execution, 113–114, 133, 137–141, 218
- Symbolic traces:
  - analysis of, 94–131
  - code-based test-case selection, 49–51
  - data-flow testing, 29–30, 47–48
  - defined, 94, 218
  - execution paths, 95, 102–109, 114, 121–125
  - present analysis method, 114–115, 124–126
  - program graph and, 94–96
  - regression testing, 88
  - rules for moving/simplifying constraints, 99–110
  - rules for moving/simplifying statements, 110–114
  - simplified, 115, 121–122, 125–126
  - state restraint, 96–99
  - supporting software tool, 126–131
- Syntactic paths, data-flow testing, 29
- System testing, 88
- Tautology, 196–198
- Term, first-order predicate calculus, 201
- Terminology, 4–8
- Test case, defined, 77, 218
- Test-case designer, functions of, 20
- Test case selection:
  - automated, 48, 72
  - code-based methods, *see* Code-based test-case selection methods
  - cost effective, 9–10
  - criterion, choosing, *see* Test-case selection criterion
  - faults, classification of, 10–11
  - importance of, 3
  - methods, classification of, 11–12
  - principles of, 8–9
  - proactive approach, 20
- Test-case selection criterion:
  - choosing, 78, 90–92
  - defined, 4–5, 218
  - program testing costs, 12
- Test coverage:
  - branch testing, 22
  - defined, 219
  - monitoring capability, 48
  - statement tests, 20
  - verification, 91
- Test, defined, 218
- Test drivers, 84
- Tester, defined, 219
- Test execution, 6, 13
- Test harness, 83
- Test-result analysis, cost of, 13
- Test set:
  - construction of, *see* Test-set construction
  - coupling coefficient, 7–8
  - defined, 4, 219
  - effectiveness of, 6–7
  - faults, 5–6
  - operation sequences, 7, 55
  - selection, *see* Test case selection
- Test-set construction:
  - components of, 6, 15, 19
  - path testing, 15–16
  - statement testing, 19–20
- Text editor, symbolic trace analysis, 126–127
- Theorem(s):
  - defined, 219
  - proving, 1
  - test-case selection criterion, 79
- TN (trace number), 188–189
- Top-down integration, 83–84, 88
- Trace analyzer, 126
- Trace-subprogram generation through instrumentation:
  - overview of, 181–182

- program graph, 182
- symbolic trace, 182–183
- Trap statement, 43
- Truth table, 195, 200
  
- Unary operator, program mutation:
  - insertion, 43
  - removal, 43
- Undo**, symbolic trace analysis, 126–131
- Unit testing:
  - characterized, 83–86, 88
  - methods, 6
  - software development, 91
- Unreachable statements, 132
- Unreferenced labels, 132
- Unschedule, data-flow anomalies, 136
  
- Upper bound (UB), 70–71, 91
- Use-based test, 80
  
- Validate**, symbolic trace analysis, 126–131
- Validity of of test-case selection criterion, 219
- Validity testing, 78
- Valid messages, object-oriented testing, 86
- Valid wff, 202
- Value trace, 143–145
- Vectors, branch testing, 24
  
- Walkthrough, 219
- Weakest precondition, 99, 154, 219
- Well-formed formula (wff), 195–196, 201
- WHILE statement, trace subprogram, 185–186, 190
  
- Zero-two (ZT) subset, 178

