

Index

- A**
- Addition process, writing/
 - testing code
 - adding parameters, 67–70
 - declarations, causing other changes to ripple, 73–80
 - eliminating data duplication, 67–70
 - implementing/moving code, 61–65
 - metaphors, 56–59
 - AllTests suite testing, 163–164
 - Application test-driven development (ATDD), *versus* TDD, 199
 - Arrange/Act/Assert (3A) pattern, 97
 - Assert-First testing, 128–129
 - Assertions design pattern, 157–158
 - ATDD (application test-driven development), *versus* TDD, 199
- B**
- Bootstrapping problems, test method, 91–95
 - Broken Tests, 148–149
- C**
- Child Tests, 143
 - Classes
 - code metrics, 84
 - concrete, 46
 - defining, 5
 - eliminating explicit class checking with polymorphism, 61–65
 - equality testing, 15–18, 27–31
 - subclasses, 27–31, 36–37
 - subclasses, eliminating, 39–43
 - subclasses, replacing references with references to superclasses, 51–53
 - superclasses, 27–31
 - Clean code check-in, 149–150
 - Code duplication
 - and code dependency, 7, 8
 - removing, 24, 25
 - Code metrics, 84
 - Code refactoring, 40–43
 - Collecting Parameter design pattern, 114, 166, 178–179
 - Command design pattern, 165, 166, 167
 - Composite design pattern, 113–117, 166, 176–178
 - and Impostors, 176
 - Concrete classes, 46
 - Crash Test Dummy, 147–148
 - extracting interfaces, 187
 - Cyclomatic complexity, code metrics, 84
- D**
- Dependency on code, and duplication of code, 7, 8
 - Design patterns
 - Broken Tests, 148–149
 - Child Tests, 143
 - clean code check-in, 149–150
 - Collecting Parameters, 114, 166, 178–179
 - Command, 165, 166, 167
 - Composite, 113–117, 166, 176–178
 - Crash Test Dummy, 147–148, 187
 - Factory Methods, 166, 174–175
 - Impostors, 166, 175–176
 - Log Strings, 146–147
 - Mock Objects, 187
 - Null Objects, 166, 169–170
 - overview, 165–166
 - Pluggable Objects, 166, 171–173
 - Pluggable Selectors, 166, 173–174
 - Self Shunt Tests, 145–146
 - Singletons, 179
 - Template Method, 166, 170–171
 - Value Objects, 165, 166, 167–169
 - Design patterns, green bar patterns
 - Fake It, 151–153
 - Obvious Implementation, 154–155
 - One to Many, 155–156, 183–184
 - Triangulation, 153–154
 - Design patterns, red bar patterns
 - Explanation Tests, 135–136
 - Learning Tests, 136–137
 - One Step Tests, 133–134
 - Regression Tests, 137–138
 - Starter Tests, 134–135
 - Design patterns, xUnit framework
 - AllTests, 163–164
 - Assertions, 157–158
 - Exception Tests, 163

- Design patterns, xUnit framework, *continued*
 - Fixtures, 158–160
 - Fixtures, External, 160–161
 - Test Method, 161–163
 - Duplication of code and dependency on code, 7, 8
 - removing, 24, 25
 - E**
 - Equality testing, 15–18, 27–31, 33–34
 - Error messages, 47
 - Evident Data, 130–131
 - Exceptions
 - exception handling, 105–107
 - Exception Tests, 163
 - Explanation Tests, 135–136
 - Explicit class checking, eliminating with polymorphism, 61–65
 - Extract Interface and Self Shunt Tests, 146
 - Extract Method, 95
 - versus* Method Objects, 189
 - refactoring, 183, 184–185
 - Extract Objects, 183
 - Extreme Programming (XP), *versus* TDD, 204–205
 - F**
 - Factory Methods, 36–37
 - design patterns, 166, 174–175
 - Failures in testing
 - Mean Time Between Failures, 196–197
 - multiplication process, 4–6
 - xUnit framework, 109–111
 - Fake It implementations of TDD, 13, 106, 138
 - Crash Test Dummy, 147
 - green bar design patterns, 151–153
 - Feedback
 - amount needed in TDD, 196–197
 - influence diagrams, 208–209
 - Fibonacci, 211–213
 - Flags of called methods, 91–100
 - versus* logs, 101
 - Functions, code metrics, 84
 - G–H**
 - Green bar design patterns
 - Fake It, 151–153
 - Obvious Implementation, 154–155
 - One to Many, 155–156
 - One to Many data migration, 183–184
 - Triangulation, 153–154
- I**
 - Implementations of TDD
 - Fake It, 13, 103, 138
 - Fake It, and Crash Test Dummy, 147
 - Fake It , green bar design patterns, 151–153
 - Obvious, 13, 103, 138
 - Obvious Implementation, green bar patterns, 154–155
 - Triangulation, 13, 16–17, 138
 - Triangulation , green bar design patterns, 153–154
 - Impostor design pattern, 166, 175–176
 - Influence diagrams, 123–124, 127
 - elements, 207–208
 - feedback, 208–209
 - systems of development practices, 209–210
 - Inline Method, 185–186
 - Instance variables, private, 20–21
 - Isolated tests, design patterns, 125–126
 - Isolating changes
 - One to Many design patterns, 155–156
 - refactoring, 182–183
- J–K**
 - JProbe, 86
 - JUnit, using, 83
 - JXUnit, 158
- L**
 - Learning Tests, 136–137
 - Log Strings, 146–147
 - Logs of called methods, 101–107
 - versus* flags, 101
- M**
 - Mean Time Between Failures (MTBF), 196–197
 - Metaphors, 56–59
 - advantages, 82–83
 - Method Objects, *versus* Extract Method, 189
 - Methods
 - Extract Method, refactoring, 183, 184–185
 - Extract Method, *versus* Method Objects, 189
 - Factory Methods, 36–37, 166, 174–175
 - Inline Method, 185–186
 - Move Method, 187–189
 - parameters, adding to methods, 190
 - parameters, moving from methods, 190–191
 - reconciling, 45–49
 - setUp(), 97–100
 - tearDown(), 101–107
 - Template Method, 146, 166, 170–171
 - testMethod(), 91–95
 - Mock Objects, 144–145
 - and Crash Test Dummy, 148
 - and extracting objects, 187
 - Move Method, 187–189
 - MTBF (Mean Time Between Failures), 196–197
 - Multi-currency money object, creating, 3–10
 - Multiplication process, writing/testing code, 7, 11
 - equality testing, 15–18, 27–31, 33–34
 - factory methods, 36–37
 - implementations, Fake It, 13
 - implementations, Obvious, 13
 - implementations, redundant, 31
 - implementations, redundant, eliminating, 51–53
 - implementations, Triangulation, 13, 16–17
 - improving tests with functionality, 19–21
 - refactoring code, 40–43
 - testing cycle, 1, 7, 11, 84–85
 - testing cycle, decisions on initial steps, 4–5
 - testing cycle, failures, 4–6
 - testing cycle, scope, 5, 23–25
 - writing tests by copying/editing code, 23–25

- N**
 Negative feedback, influence diagrams, 208–209
 Null Objects
 design patterns, 166, 169–170
 and Impostors, 176
- O**
 Objects
 creating, 35–37
 creating, by force of tests, 61–65
 creating, constraints causing conflicts, 97–100
 creating, multi-currency money, 3–10
 Mock Objects, 144–145
 Mock Objects, and Crash Test Dummy, 148
 Mock Objects, and extracting interfaces, 187
 Null Object design patterns, 166, 169–170
 Null Object design patterns, and Impostors, 176
 Open/Closed Principle, 195–196
 Value Objects, 15–18
 Obvious Implementation of TDD, 13, 103, 138
 green bar patterns, 154–155
 One Step Tests, 133–134
 and Starter Tests, 135
 One to Many design patterns, 155–156
 data migration, 183–184
 Open/Closed Principle (objects), 195–196
- P–Q**
 Pair programming, physical setup, 140–141
 Performance testing, *versus* TDD, 86–87
 Pluggable Objects, design patterns, 166, 171–173
 Pluggable Selectors, 95
 design patterns, 166, 173–174
 Polymorphism, eliminating explicit class checking, 61–65
 Positive feedback, influence diagrams, 208–209
 Private instance variables, 20–21
- R**
 Realistic Data, 130
 Red bar design patterns
 Explanation Tests, 135–136
 Learning Tests, 136–137
 One Step Tests, 133–134
 Regression Tests, 137–138
 Starter Tests, 134–135
 Refactoring code, 40–43
 Extract Method, 183, 184–185
 Extract Objects, 183
 extracting interfaces, 187
 Inline Method, 185–186
 isolating changes, 182–183
 Method Objects, 183, 189–190
 migrating data, 183–184
 Move Method, 187–189
 parameters, adding to methods, 190
 parameters, moving from methods, 190–191
 reconciling differences, 181–182
 Regression Tests, 137–138
- S**
 Scripts with xUnit testing framework, 120
 Self Shunt Tests, 145–146
 Setup() method, 97–100
 Shower Methodology, 138
 Singleton design patterns, 179
 SmallLint for SmallTalk, 82
 Starter Tests, 134–135
 and One Step Tests, 135
 Stress testing
 influence diagrams, 123–124
 versus TDD, 86–87
 Subclasses, 27–31, 36–37
 eliminating, 39–43
 replacing references with references to superclasses, 51–53
 Suites for multiple tests
 AllTests, 163–164
 TestSuite, 113–117
 Superclasses, 27–31
 replacing references to subclasses, 51–53
- T**
 TDD
 analysis of term, 203–204
 versus application test-driven development, 199
 attributes of effective tests, 194–195
 beginning TDD practices in middle of projects, 199–200
 decisions on content for testing, 194
 deleting tests, 198
 extending reach of TDD, 205
 versus Extreme Programming, 204–205
 feedback amount needed, 196–197
 Fibonacci, 211–213
 influence of programming languages and environment, 198
 JProbe, 86
 Mean Time Between Failures, 196–197
 Open/Closed Principle (objects), 195–196
 versus other programming styles, 77–80
 versus other types of testing, 86–87
 potential users, 200–201
 reasons for effectiveness, 202–203
 reusable code, 195–196
 scaling to large systems, 198–199
 SmallLint for SmallTalk, 82
 test, definition, 123
 test quality, 86–87
 testing sequence, 201
 TDD design patterns, 201–202
 Assert-First, 128–129
 basics, 123–125
 Broken Tests, 148–149
 Child Tests, 143
 clean code check-in, 149–150
 Collecting Parameter, 114
 Collecting Parameters, 166, 178–179
 Command, 165, 166, 167
 Composite, 113–117, 166, 176–178
 Crash Test Dummy, 147–148
 Crash Test Dummy, extracting interfaces, 187
 Evident Data, 130–131
 Factory Methods, 166, 174–175

- TDD design patterns, *continued*
 - Impostors, 166, 175–176
 - isolated tests, 125–126
 - Log Strings, 146–147
 - Mock Objects, 144–145
 - Null Objects, 166, 169–170
 - overview, 165–166
 - Pluggable Objects, 166, 171–173
 - Pluggable Selectors, 166, 173–174
 - Realistic Data, 130
 - Self Shunt Tests, 145–146
 - Singletons, 179
 - Template Method, 166, 170–171
 - Test Data, 129–130
 - Test-First, 127–128
 - Test Lists, 126–127
 - Value Objects, 165, 166, 167–169
- TDD design patterns, green bar patterns
 - Fake It, 151–153
 - Obvious Implementation, 154–155
 - One to Many, 155–156
 - One to Many, data migration, 183–184
 - Triangulation, 153–154
- TDD design patterns, red bar patterns
 - Explanation Tests, 135–136
 - Learning Tests, 136–137
 - One Step Tests, 133–134
 - Regression Tests, 137–138
 - Starter Tests, 134–135
- TDD design patterns, xUnit framework
 - AllTests, 163–164
 - Assertions, 157–158
 - Exception Tests, 163
 - Fixtures, 158–160
 - Fixtures, External, 160–161
 - Test Method, 161–163
- TDD guidelines
 - physical setup, 140–141
 - starting over, 139–140
 - taking breaks, 138–139
- TDD refactoring
 - Extract Method, 183, 184–185
 - Extract Objects, 183
 - extracting interfaces, 187
 - Inline Method, 185–186
 - isolating changes, 182–183
 - Method Objects, 183, 189–190
 - migrating data, 183–184
 - Move Method, 187–189
 - parameters, adding to methods, 190
 - parameters, moving from methods, 190–191
 - reconciling differences, 181–182
- TDD testing cycle, 1, 7, 11, 84–85
 - failures, 4–6
 - scope, 5, 23–25
 - steps, initial steps, 4–5
 - steps, size of, 193–194
- TearDown() method, 101–107
- Template Method, 146
 - design patterns, 166, 170–171
- Test coupling, 98
- Test Data, 129–130
- Test-Driven Development. *See* TDD
- Test-First testing, 127–128
- Test Lists, 126–127
- Testing/writing code, addition process
 - adding parameters, 67–70
 - declarations, causing other changes to ripple, 73–80
 - eliminating data duplication, 67–70
 - implementing/moving code, 61–65
 - metaphors, 56–59
- Testing/writing code, multiplication process
 - equality testing, 15–18, 27–31, 33–34
 - factory methods, 36–37
 - implementations, Fake It, 13
 - implementations, Obvious, 13
 - implementations, redundant, 31
 - implementations, redundant, eliminating, 51–53
 - implementations, Triangulation, 13, 16–17
 - improving tests with functionality, 19–21
 - refactoring code, 40–43
 - testing cycle, 1, 7, 11, 84–85
 - testing cycle, decisions on initial steps, 4
 - testing cycle, failures, 4–6
 - testing cycle, scope, 5, 23–25
 - versus* work code, 8, 9
 - writing tests by copying/editing code, 23–25
- TestMethod() method, 91–95
- TestSuite, multiple tests, 113–117
- 3A (Arrange/Act/Assert) pattern, 97
- To-do lists, 4
- Triangulation implementation of TDD, 13, 16–17, 138
 - green bar design patterns, 153–154
- U–W
- Usability testing, *versus* TDD, 86–87
- Value Objects, 15–18
 - design patterns, 165, 166, 167–169
- Variables
 - private instance variables, 20–21
 - replacing constants, 45–49
- Working code, *versus* test code, 8, 9
- X–Z
- XP (Extreme Programming), *versus* TDD, 204–205
- XUnit testing framework
 - failures, 109–111
 - methods, setUp(), 97–100
 - methods, tearDown(), 101–107
 - methods, testMethod(), 91–95
 - reasons for implementing, 119
 - TestSuite, running multiple tests, 113–117
- XUnit testing framework, TDD design patterns
 - AllTests, 163–164
 - Assertions, 157–158
 - Exception Tests, 163
 - Fixtures, 158–160
 - Fixtures, External, 160–161
 - Test Method, 161–163