

# Index

## A

**abstractness, 136**  
 graph of instability and, 141

**actionable messages, 196–198**

**Active Directory Lightweight Directory Services (Microsoft), 9**

**Active Record pattern, 158**

**activities, correlation IDs and, 192**

**afferent coupling, 136**

**agile software methodologies, 11, 12, 21, 87–88, 93, 97. *See also* Scrum; XP**  
 criticism of, 87

**algorithm testing, 74, 80, 88, 103**

**Ankh, 115**

**Ant, 26, 27, 28, 100, 117**

**antipattern, in unit tests, 86–87**

**Apache**  
 Ant. *See* Ant  
 log4net. *See* log4net  
 Subversion and, 116. *See also* Subversion

**APIs. *See* application programming interfaces**

**application programming interfaces (APIs), 6.**  
*See also* base class libraries  
 database, 86, 87  
 tracing, 192

**ArgumentNullException, 18, 68, 72, 190**

**assertions, 14**  
 debug, 149–150

**atomic commits, 112–113**

**Authorization Manager (Microsoft), 8**

**authorization systems, 7, 8**

**automated build process**  
 scripts, 26–29  
 Java package (Ant, NAnt), 27–28  
 Rake, 28  
 simple class library (MSBuild), 26–27

static analysis integrated into, 140–143

**automated testing, 29–31, 96–97**

strategies, 96–97

## B

**base class libraries (BCLs), 6–7**

**baseline architecture, 8, 9, 10, 44, 87–88, 94, 95**

**BCLs. *See* base class libraries**

**Berkeley DB, 9**

**boundary conditions, 80–82**

**branching/branches, 109, 113, 122–133**  
 integration, 23, 127–129  
 cycle, 128  
 merging, 109, 114, 131–133, 220  
 personal, 129–130  
 task, 130–131  
 version  
 creating, 123–125  
 managing, 125–127

**broken builds, CI and, 23, 37–39, 105**

**build servers (CI), 25–26**  
 automating build process, 26–29  
 expanding build process, 29–31

**build tools**  
 Ant, 26, 27, 28, 100, 117  
 FinalBuilder, 26  
 MSBuild, 26, 28, 100, 117  
 NAnt, 26, 27, 28, 34, 100, 117, 118  
 Rake, 26, 28  
 VisualBuild, 26

**built-in methods, 6. *See also* base class libraries**

**business value, 3–4, 5, 10, 61**

**buy-versus-build perspective, 3–10. *See also* writing code**

**bytecode, 162**

## C (programming language)

---

### C

#### C (programming language)

- compilers, 26
- Simian on, 139

#### C#

- clear contracts, 148–149
- cyclomatic complexity, 136
- data contracts, 155–156
- dependency injection, 165–166
- documentation comments, 54, 55
- edge cases, 81
- error-handling policy, 203–207
- exceptions and, 18
- management systems, 4
- new keyword, 164–165
- sealed class, 163, 164
- Simian on, 139
- spec#, 151–152
- static initializer, 162
- turning off warnings, 52

#### C++

- code coverage tools, 68
- compilers, 26
- friend keyword, 74
- stdlib for, 6
- virtual methods, 163
- VTABLE, 165

#### calculator project (case study), 213–221

- build and test cycle for application, 219, 220
- build server (checking of), 213, 220
- buy-*versus*-build perspective and, 221
- dependency injection and, 221
- ‘done’ tasks, 218–220
- error-handling code, 218–219, 221
- mock view class, 216, 221
- MVP model and, 213, 221
- separate task branch in, 214, 215, 220
- static analysis and, 221
- Subtract functionality
  - added to MVP-style application, 215–218
  - new tests for, 214–215
  - update documentation, 219
- task 5: add subtraction to calculator, 213

- TDD process (starting with tests), 214, 215, 220

- tracing code, 218–219, 221

#### canned database, 86

#### case study. *See* calculator project

#### Castle Open Source project, 170

#### Castle Windsor, 170

#### CC. *See* CruiseControl; cyclomatic complexity

#### change sets, 107, 112, 113, 114, 115

- macro-level, 123
- mega-, 130
- shelveset and, 130

#### check-ins. *See* commits

#### CI. *See* Continuous Integration

#### class(es)

- internal, 163
- test fixture exercises one, 48–49
- test fixture for each, 45–48
- undocumented, 163

#### ClearCase, 110, 112, 113, 114, 130

#### ClearQuest, 113, 131

#### code construction

- contracts (software), 147–158. *See also* contracts
- dependencies (limiting of), 159–171
- error handling, 199–210
- MVP model, 173–187
- tracing, 189–198

#### code coverage, 64–74

- cyclomatic complexity and, 65–66
- high, 49–51, 68, 71
- importance of, 65
- NCover, 28, 30, 68, 69, 70
- NCoverExplorer, 70, 71, 72
- strategies for improvement, 72–74
- surplus code, 66–67
- tools, 12, 67–72

#### code-cleanliness tools, 142. *See also* FxCop

#### codelines, 122. *See also* branching/branches

#### COM (Component Object Model) architecture, 200

#### commits, 25

- atomic, 112–113

#### communication, importance of, 45

#### competitive advantage, writing code and, 5, 8, 9, 10

- compiler warnings, 51–52**
  - compile-time dependencies, 159–160**
  - complexity, of software, 3, 4, 7**
  - Component Object Model (COM) architecture, 200**
  - computer science, software engineering v., 5**
  - concurrent versioning system. *See* CVS**
  - Continuous Integration (CI), 21–39**
    - automated testing, 29–31
    - automating build process, 26–29
    - barriers to, 23–25
    - broken builds and, 23, 37–39, 105
    - build output, coordination of, 35–36
    - build servers, 25–26
    - CI server, 31–35
    - commits, 25
    - expanding build process, 29–31
    - goals of, 22, 39
    - integrate early and often, 22
    - multiple builds of same project, 35
    - new code changes
      - check-in process, 22–23, 24
      - developer’s role, 22–23, 24
    - notification of build results, 36–37, 105
    - pessimistic locking, 21, 24, 25
    - SCC systems and, 23, 25, 35, 36, 133. *See also* source code control systems
    - tester feedback, 22, 62
    - waterfall model and, 21, 22, 24
  - contracts (software), 13, 147–158**
    - C# and clear, 148–149
    - data, 155–158
    - naming conventions, 148
    - public interfaces, 152–155
    - TDD and, 13–16, 19, 63–64
    - unit tests and, 149
  - Controller layer (MVC), 174**
  - correlation IDs, activities and, 192**
  - cost v. benefit (software components), 4–5**
  - CR\_Documentor, 55**
  - CruiseControl (CC), 31–35, 143**
    - Java version, 31, 100
    - labeler, 36
    - .NET, 31, 34, 36, 70, 100, 143
    - notification settings, 37
    - ‘projects,’ 35
    - Ruby version, 31
  - Cunningham, Ward, 93**
  - CVS (concurrent versioning system), 25, 36, 112, 113, 116. *See also* Subversion**
    - branches v. labels, 124, 125, 126, 132
    - setting up, 116
  - cyclomatic complexity (CC), 30, 136**
    - code coverage and, 65–66
- D**
- data contracts, 155–158**
    - C#, 155–156
    - mapping to storage, 158
  - database(s)**
    - APIs, 86, 87
    - canned, 86
    - layouts
      - flat, 157
      - normal, 157
    - SQL, 111
      - data contracts and, 155
    - systems, 7, 9
  - debug assertions, 149–150**
  - defect-tracking system, 113, 114, 130**
  - dependencies, 159–171**
    - compile-time, 159–160
    - interfaces and, 162–165, 171
    - IoC and, 45, 49, 167–170, 171
    - limiting surface area, 162–165
    - reduction strategies, 171
  - dependency injection, 49, 165–167, 171**
    - C#, 165–166
  - ‘design by contract’ extensions, 152**
  - design decisions, discussion of, 44–45**
  - design goal, writing code and, 13**
  - developers. *See* software developers**
  - development process. *See* software-development process**
  - DirectX, 9**
  - ‘distance,’ virtual, 209**
  - documentation comments**
    - C#, 54, 55
    - XML, 54
      - thrown exceptions, 205, 206

## documentation in place (rule)

---

**documentation in place (rule), 53–56**

**Domain-Driven Design method, 177**

**'done' tasks (set of rules), 43–56**

- code coverage, high, 49–51, 68, 71
- design decisions, discussion of, 44–45
- documentation in place, 53–56
- each fixture exercises one class, 48–49
- every class has test fixture, 45–48
- no compiler warnings, 51–52
- static analysis tools generate no errors, 52–53
- update before committing, 24, 53

**dynamic analysis, 135. See also static analysis tools**

## E

**early testing, 12, 13–16, 60, 106**

**Eclipse IDE, 115**

**edge cases, 80–82**

- C#, 81
- testers and, 84

**efferent coupling, 136**

**Eiffel, 147, 148, 151**

**80% solution, 4, 5**

**encryption, 7, 9**

**Entity Beans (Java), 158**

**Entity Data Framework project (Microsoft), 158**

**error handling, 199–210**

- exception throwing, 201–203
- internal, 199
- policy, 203–207
  - C#, 204–207
  - defining of, 204–207
  - importance of, 203–204
- presented to user, 199
- result code reading, 200–201
- TDD and, 17–18
- unit testing, 82–83
- where to handle errors, 207–210

**exception throwing, 201–203**

**exception-handling code, 51, 65, 210**

**exceptions**

- C# and, 18
- expected, 18
- Java and, 18

**expanding build process, 29–31**

**expected exceptions, 18**

**[ExpectedException] attribute, 18, 63, 72, 83, 149**

**Extreme Programming. See XP**

## F

**factory classes, 161–162**

**FinalBuilder, 26**

**FIT (Framework for Integrated Test), 93, 103**

**[FixtureCategory] attribute, 99–100**

**flat database layout, 157**

**foreign key relationships, 158**

**Fowler, Martin, 11, 16, 25**

**Framework for Integrated Test. See FIT**

**friend keyword (C++), 74**

**functional tests, 89–93**

- FIT, 93, 103
- goal of, 93
- testers and, 93
- testing frameworks, 102

**FxCop, 30–31, 52, 137–138, 139, 140, 142**

- example report, 142–143

## G

**Gantt chart-like format, 192**

**Google.com search engine, automation of, 90–91, 92**

## H

**HelloWorldToday( ) example, 66, 67, 73**

**Hibernate, 158**

**high code coverage, 49–51, 68, 71**

**HTML**

- FIT and, 103
- public interface, 154, 155
- reports
  - MbUnit and, 100
  - NCoverExplorer and, 72
- Simian on, 139
- specialists, 3
- TestRunners and, 90
- WaitN and, 102
- XML reports formatted as, 29, 30

- I**
- IBM MQ series, 9**
  - IBM Rational Software Delivery Program, 112, 114, 131**
  - IDEs (integrated development environments), 26**
    - Eclipse, 115
    - Visual Studio. *See* Visual Studio
  - IFileReader, 50, 51, 54**
  - independent software vendors (ISVs), 9**
  - Indexed Sequential Access Method database files. *See* ISAM database files**
  - input parameters, validation of, 150–151**
  - instability, 136**
    - graph of abstractness and, 141
  - integrate early and often (CI tenet), 22**
  - integrated development environments. *See* IDEs**
  - integration branches, 23, 127–129**
  - integration tests, 84–89**
    - developers and, 89
    - remote triggering, 89
    - unit tests v., 84–85, 88
    - when to run, 88–89
    - when to write, 87–88
  - intent, of code, 16–18**
    - comments and, 16
    - TDD/unit tests and, 16–18, 149
  - interfaces (public), 152–155**
  - internal classes, 163**
  - InternalsVisibleTo attribute, 74**
  - interpreted script, 162**
  - inversion of control (IoC), 45, 49, 167–170, 171**
  - IoC. *See* inversion of control**
  - ISAM (Indexed Sequential Access Method) database files, 111**
  - ISVs (independent software vendors), 9**
- J**
- Java**
- Ant, 26, 27, 28, 100, 117
  - CruiseControl, 31, 100
  - ‘design by contract’ extensions, 152
  - Eclipse IDE and, 115
  - Entity Beans, 158
  - exceptions and, 18
  - JUnit for, 97
  - libraries, 6
  - log4j, 190
  - namespace organization, 120
  - packaging for deployment, 121
  - Simian on, 139
  - testing framework, 14
  - virtual methods, 163
  - VTABLE, 165
- JavaScript, ‘design by contract’ extensions in, 152**
  - JetBrain’s ReSharper, 14**
  - JUnit (testing framework), 90, 97, 117. *See also* NUnit**
- L**
- labels, 36, 132–133**
  - LDAP directories, 7, 9**
  - legacy code, 30, 51, 52, 71, 72**
  - limiting dependencies, 159–171**
  - locking**
    - optimistic, 110, 111, 121
    - pessimistic, 21, 24, 25, 108, 109, 110, 111, 112, 121
  - log sinks, 190–192**
  - log sources, 190–192**
  - log4j, 190**
  - log4net, 159, 160, 161, 190, 191, 192, 193, 195**
  - logging, 189, 190. *See also* tracing**
- M**
- ‘make file’ format, 26**
  - MbUnit, 28, 81–82, 83, 92, 150**
    - FixtureCategory attribute, 99–100
    - WatiN and, 102
    - XML report, 29–30
  - Mercury TestRunner, 102**
  - merging branches, 109, 114, 131–133, 220**
  - Message object, null, 18**
  - message-passing-style architectures, 16**
  - messaging systems, 9**
  - method calls, 9**

## Microsoft Active Directory Lightweight Directory Services

---

**Microsoft Active Directory Lightweight Directory Services, 9**

**Microsoft Authorization Manager, 8**

**Microsoft Entity Data Framework project, 158**

**Microsoft Research, 151. *See also* Spec#**

**Microsoft Team System, 114, 130, 131**

**Microsoft Visual Studio. *See* Visual Studio**

**mock object frameworks, 49, 51, 74, 79**

- IoC and, 168, 169, 170
- negative testing, 83
- reduce dependencies, 78
- Rhino.Mocks, 79, 168, 169
- to simulate Model, 186

**Model layer**

- MVC, 174
- MVP, 175

**Model-View-Controller (MVC) model, 174**

**Model-View-Presenter (MVP) model, 173–187**

- advantages of, 174, 176
- communication between View and Presenter, 176–177
- construction of, 177–184
- strategy of, 173
- testing applications, 184–187

**Moore's Law, 4**

**MQ series (IBM), 9**

**MS TeamServer, 100**

**MSBuild, 26, 28, 100, 117**

- simple class library, 26–27

**MVC model. *See* Model-View-Controller model**

**MVP model. *See* Model-View-Presenter model**

**MySQL, 5**

## N

**NAnt, 26, 27, 28, 34, 100, 117, 118**

**NCover, 28, 30, 68, 69, 70**

**NCoverExplorer, 70, 71, 72**

**NDepend, 30, 52, 136–137**

- abstractness/instability graph, 141
- user interface, 137

**NDoc, 55**

**negative testing, 82–83. *See also* error handling**

- mock object frameworks and, 83

**.NET Framework**

- 1.1 implementation, 6

BCL, 6

CruiseControl, 31, 34, 36, 70, 100, 143

Tracing subsystem, 190, 192

**network services, 9**

**new keyword (C#), 164–165**

**Neward, Ted, 158**

**notification of build results (CI), 36–37, 105**

**null Message object, 18**

**NullReferenceException, 18, 77, 203**

**NUnit (testing framework), 14, 34, 47, 76, 77,**

**81, 82, 83, 90, 92, 97, 98, 117, 173, 185**

expected exceptions and, 18

**nunit.org, 14**

## O

**Object Oriented Design. *See* OOD**

**object-oriented (OO) languages, 164**

**object-relational mapping systems, 158**

**OO languages. *See* object-oriented languages**

**OOD (Object Oriented Design), 177**

**Open Source software (OSS), 7**

Ankh, 115

Apache. *See* Apache

CVS. *See* CVS

Eclipse IDE and, 115

Subversion. *See* Subversion

**OpenGL, 9**

**operating system platforms, 7–9**

complexity, 7

services, 7–9

risk management, 8, 9

**optimistic locking, 110, 111, 121**

**OSS. *See* Open Source software**

## P

**pair programming, 12, 13, 64**

**performance testing, 93–95**

steps for, 94–95

when to do, 94

**personal branches, 129–130**

**pessimistic locking, 21, 24, 25, 108, 109, 110,**

**111, 112, 121**

**philosophies (pragmatic)**

buy-versus-build perspective, 3–10

CI, 21–39

TDD, 11–19

**POET, 158**

**policy**

error-handling, 204–207

tracing, 193–196

**polymorphism, 165**

**postconditions, 148, 152**

**#pragma warning disable directives, 52**

**preconditions, 85, 148, 151, 152**

**Presenter layer (MVP), 175**

**problem solving, 3, 5**

**'programming by contract,' 151, 152**

**public interfaces, 152–155**

**Python**

'design by contract' extensions, 152

packaging for deployment, 121

## Q

**QA. See quality assurance**

**quality assurance (QA), 57, 104. See also testing**

## R

**Rails, Ruby on, 158**

**Rake, 26, 28**

**Rational Robot, 102**

**Rational Software Delivery Program, 112, 114, 131**

**RCS (Revision Control System), 108–109**

**ReaderWriterLock implementation, 6**

**red light/green light user interface, 14**

**red/green/refactor cycle, 11, 66, 78, 139**

**refactoring, 11, 73–74. See also Test-Driven Development**

red/green/refactor cycle, 11, 66, 78, 139

**Refactoring: Improving the Design of Existing Code (Fowler), 11, 16**

**regression testing, 12, 19, 59, 83, 187**

**[Requires] attribute, 152**

**ReSharper, 14**

**result code reading, 200–201**

**Reverse Polish Notation. See RPN**

**Revision Control System. See RCS**

**Rhino.Mocks framework, 79, 168, 169. See also mock object frameworks**

**risk management**

OS services, 8, 9

third-party components, 8, 9, 10

**RPN (Reverse Polish Notation) calculator**

interface, 14, 15

test (before defining interface), 15

test (before implementing interface), 14

**Ruby**

class library, 6

CruiseControl, 31

'design by contract' extensions, 152

on Rails, 158

Rake build tool, 26, 28

Simian on, 139

Watir in, 90–92, 102

## S

**Sandcastle project, 55**

**Satisfactory Coverage percentage, 70**

**SCC systems. See source code control systems**

**Scrum, 21, 43, 130, 213**

**sealed class, 163, 164**

**SendEmailMessage method, 18**

**servers**

build. See build servers

virtualized, 25

**Service Configuration Editor, 195–196**

**Service Level Agreement, 6**

**Service Oriented Architecture (SOA), 16, 177**

**services (OS platforms), 7–9**

risk management, 8, 9

**shelveset, 130**

**Silk Test, 102**

**Simian, 34, 53, 139, 140, 143**

on various language files, 139

**singletons, 162, 179**

**SmallTalk, 14, 97**

**SOA. See Service Oriented Architecture software**

complexity, 3, 4, 7

## software (*continued*)

---

### **software (*continued*)**

contracts, 147–158. *See also* contracts

cost v. benefit, 4–5

third-party components. *See* third-party components

### **software developers, 5**

computer scientists v., 5

integration tests written by, 89

new code changes and role of, 22–23, 24

role of. *See* writing code

unit tests written by, 83

### **software-development process**

'done' tasks, 43–56

SCC systems, 107–133. *See also* source code control systems

static analysis tools, 135–143

testing, 57–106

importance of, 57–58, 106

### **solving problems, 3, 5**

### **source available software, 7**

### **source code control (SCC) systems, 45, 107–133**

adding folders/directories to, 121–122

atomic commits, 112–113

branching, 109, 113, 122–133

choosing, 110–117

CI and, 23, 25, 35, 36, 133

cost, 112

CruiseControl and. *See* CruiseControl

CVS, 25, 36, 112, 113, 116

branches v. labels, 124, 125, 126, 132  
setting up, 116

extensibility, 115

history, 108–109

integration with other tools, 114–115

labels in, 36, 132–133

locking models

optimistic, 110, 111, 121

pessimistic, 21, 24, 25, 108, 109, 110, 111, 112, 121

merging branches, 109, 114, 131–133

organizing source tree, 117–120

packaging for deployment, 121

performance/scalability, 111

reporting, 115

Subversion. *See* Subversion

TreeSurgeon, 117–120

### **Source Safe, 109**

### **SourceForge, 92, 109, 110**

### **Spec#, 151–152**

### **specialization, 3, 4, 5**

### **SQL**

databases, 111

data contracts and, 155

query engine, 5

### **static analysis tools, 135–143. *See also* FxCop; NDepend; Simian**

benefits, 139–140

drawbacks, 143

generate no errors, 52–53

integrating into automated build process, 140–143

metrics, 142, 143

### **static initializer**

C#, 162

### **stdlib for C++, 6**

### **String.Format, 194**

### **Subversion, 25, 36, 110, 111, 112, 113, 114, 118, 121, 122, 123, 125, 126, 127, 130, 132. *See also* source code control systems**

setting up, 116–117

Visual Studio and, 115

### **SUnit, 97**

### **surplus code, 66–67**

### **System.IO.File, factoring out dependency on, 50–51**

## **T**

### **task branches, 130–131**

#### **tasks**

defining of, 43

done, 43–56. *See also* 'done' tasks

### **TDD. *See* Test-Driven Development**

### **Team System (Microsoft), 114, 130, 131**

### **TeamServer, 100**

### **[Test] attribute, 14, 92**

### **test fixture, 98**

for each class, 45–48

exercises one class, 48–49

**Test-Driven Development (TDD), 11–19**

baseline architecture and, 8, 9, 10, 44, 87–88, 94, 95

benefits of, 19, 59–60

- make project sponsors happy, 61–64

contracts and, 13–16, 19, 63–64

intent (of code) and

- error handling, 17–18
- unit tests, 16–18, 149

less code to achieve goals, 59–60

process cycle, 11

red/green/refactor cycle, 11, 66, 78, 139

on software development lifecycle, 12

tests written first in, 12, 13–16, 60

UI testing, 173, 187. *See also* Model-View-Presenter model

**TestDriven.NET, 14****testers**

edge cases and, 84

feedback, CI and, 22, 62

functional tests written by, 93

testing and relationships with, 61

unit tests written by, 83–84, 89

**[TestFixture] attribute, 98****[TestFixtureSetUp] method, 98–99****testing, 57–106**

automated, 29–31, 96–97

code coverage, 64–74

- high, 49–51, 68, 71

early, before writing code, 12, 13–16, 60, 106

functional, 89–93

- FIT, 93, 103

importance of, 57–58, 106

integration, 84–89

mock object frameworks and. *See* mock object frameworks

MVP applications, 184–187

performance, 93–95

plans/strategies, 74–75

reasons for doing

- better designs, 59
- better relationships with testers, 61
- learning more about code, 60
- less code to achieve goals, 59–60

- make project sponsors happy, 61–64
- reasons for not doing, 58–59
- regression, 12, 19, 59, 83, 187
- types of, 74–75
- UI, 173, 187. *See also* Model-View-Presenter model
- unit, 75–84

  - of algorithms, 74, 80, 88, 103
  - contracts and, 149
  - testing frameworks, 97–101

**testing frameworks, 14, 97–102. *See also* JUnit; MbUnit; NUnit**

functional testing, 102

unit testing, 97–101

**TestRunners, 14, 77, 87, 90, 98, 100, 102, 187****text formatting, 9****third-party components, 9–10**

customized versions, 9, 10

evaluating, 10

risk management, 8, 9, 10

source code, 9

**ThoughtWorks, 31. *See also* CruiseControl****3D graphics, 9****tracing, 189–198**

actionable messages and, 196–198

activities/correlation IDs and, 192

definition of, 189

different kinds of messages, 189–190

log sources/log sinks, 190–192

policy, 193–196

- sample, 193–194

quadrants, 189, 190

**tracing API, 192****Tracing subsystem (.NET), 190, 192****TreeSurgeon, 117–120**

top-level structure, 119, 120

**TypeMock, 79. *See also* mock object frameworks****U****UI. *See* user interface****undocumented classes, 163****unit tests, 75–84**

of algorithms, 74, 80, 88, 103

## unit tests (*continued*)

---

### unit tests (*continued*)

- antipattern in, 86–87
- contracts and, 149
- edge cases, 80–82
- error handling, 82–83
- integration tests v., 84–85, 88
- against requirements, 80
- testing frameworks, 97–101
- writing
  - developers, 83
  - testers, 83–84, 89

**update before committing (rule), 24, 53**

### user interface (UI)

- components, 4, 10
- NDepend, 137
- testing, 173, 187. *See also* Model-View-Presenter model

## V

**validation, of input parameters, 150–151**

**Vault, 110, 112. *See also* ClearCase**

**VBUnit, 97**

**version branches, 123–127**

### View layer

- MVC, 174
- MVP, 175

**virtual ‘distance,’ 209**

**virtual methods, 163, 164**

**virtualized servers, 25**

### Visual Basic

- Simian on, 139
- VBUnit for, 97

**Visual Source Safe (VSS), 109, 111, 113, 115**

### Visual Studio

- .NET, 26, 32, 34, 55
  - TreeSurgeon and, 117–120
- Subversion and, 115
- Team System, 130
- VSS, 109, 115

**VisualBuild, 26**

**VSS. *See* Visual Source Safe**

**VTABLE, 165**

232

## W

**waterfall model, 21, 22, 24, 43, 93, 104**

- CI and, 21, 22, 24
- functional tests and, 93
- QA and, 104

**WatiN, 92, 102**

- MbUnit and, 102

**Watir (Web App Testing in Ruby), 90–92, 102**

**WCF. *See* Windows Communication Foundation Web App Testing in Ruby. *See* Watir**

**Windows Communication Foundation (WCF), 192, 195**

**Windows Forms, 4, 174**

**Windows Presentation Foundation, 4**

**Windows Vista SDK, 192, 195**

**Windsor, 170**

**wrapper, 161**

**write operation, 6**

### writing code

- BCLs and, 6–7
- business value, 3–4, 5, 10, 61
- buy-*versus*-build perspective, 3–10
- competitive advantage, 5, 8, 9, 10
- cost v. benefit, 4–5
- design goal and, 13
- 80% solution, 4, 5
- OSS and, 7
- Service Level Agreement and, 6
- services (OS platforms), 7–9
  - risk management, 8, 9
- software complexity and, 3, 4, 7
- solving problems, 3, 5
- specialization and, 3, 4, 5
- third-party components, 9–10
  - risk management, 8, 9, 10
- UI components and, 4, 10

## X

### XML

- appender, 192
- automated build scripts and, 26, 28
- comments (in C#), 54, 55

documentation comments, 54  
  thrown exceptions, 205, 206  
Log4net and, 195  
report (FxCop), 142–143  
report (unit testing), 29–30

**XP (Extreme Programming), 11, 21, 43, 93, 97, 123**

**XUnit frameworks, 97**

**xUnit.net, 97**

## Y

**YAGNI (You Ain't Gonna Need It) syndrome, 59–60**

## Z

**Zone of Pain, 141**

**Zone of Usefulness, 141**