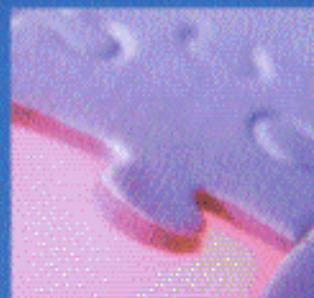


CEM KANER  
JAMES BACH  
BRET PETTICHORD

Lessons  
Learned



in ***SOFTWARE  
TESTING***

A Context-Driven Approach

## Chapter 1 The Role of the Tester

Lesson 1	You are the headlights of the project	1
Lesson 2	Your mission drives everything you do	2
Lesson 3	You serve many clients	3
Lesson 4	You discover things that will "bug" someone whose opinion matters	4
Lesson 5	Find important bugs fast	4
Lesson 6	Run with the programmers	5
Lesson 7	Question everything, but not necessarily out loud	5
Lesson 8	You focus on failure, so your clients can focus on success	6
Lesson 9	You will not find all the bugs	6
Lesson 10	Beware of testing "completely"	7
Lesson 11	You don't <i>assure</i> quality by testing	8
Lesson 12	Never be the gatekeeper!	8
Lesson 13	Beware of the not-my-job theory of testing	8
Lesson 14	Beware of becoming a process improvement group	9
Lesson 15	Don't expect anyone to understand testing, or what you need to do it well	10

## Chapter 2 Thinking Like a Tester

Lesson 16	Testing is applied epistemology	11
Lesson 17	Studying epistemology helps you test better	12
Lesson 18	Testing is grounded in cognitive psychology	13
Lesson 19	Testing is in your head	14
Lesson 20	Testing requires inference, not just comparison of output to expected results	14
Lesson 21	Good testers think technically, creatively, critically, and practically	15
Lesson 22	Black box testing is not ignorance-based testing	15
Lesson 23	A tester is more than a tourist	16
Lesson 24	All tests are an attempt to answer some question	16
Lesson 25	All testing is based on models	17
Lesson 26	Intuition is a fine beginning, but a lousy conclusion	17
Lesson 27	To test, you must explore	17
Lesson 28	Exploring involves a lot of thinking	18
Lesson 29	Use the logic of abductive inference to discover conjectures.	19
Lesson 30	Use the logic of conjecture and refutation to evaluate a product	20
Lesson 31	A <i>requirement</i> is a quality or condition that matters to someone who matters	20

Lesson 32	You discover requirements by conference, inference, and reference	21
Lesson 33	Use implicit as well as explicit specifications	22
Lesson 34	"It works" really means it appears to meet some requirement to some degree	23
Lesson 35	In the end, all you have is an impression of the product	23
Lesson 36	Don't confuse the test with the testing	23
Lesson 37	When testing a complex product: plunge in and quit	24
Lesson 38	Use heuristics to quickly generate ideas for tests	25
Lesson 39	You can't avoid bias, but you can manage it	25
Lesson 40	You're harder to fool if you know you're a fool	26
Lesson 41	When you miss a bug, check whether the miss is surprising or just the natural outcome of your strategy	27
Lesson 42	Confusion is a test tool	27
Lesson 43	Fresh eyes find failure	28
Lesson 44	Avoid following procedures unless they followed you first	28
Lesson 45	When you do create test procedures, avoid "1287"	29
Lesson 46	One important outcome of a test process is a better, smarter tester	29
Lesson 47	You can't master testing unless you reinvent it	30

### Chapter 3 Testing Techniques

Lesson 48	Testing combines techniques that focus on testers, coverage, potential problems, activities, and evaluation	32
Lesson 49	People-based techniques focus on who does the testing	34
Lesson 50	Coverage-based techniques focus on what gets tested	35
Lesson 51	Problems-based techniques focus on why you're testing (the risks you're testing for)	39
Lesson 52	Activity-based techniques focus on how you test	40
Lesson 53	Evaluation-based techniques focus on how to tell whether the test passed or failed	42
Lesson 54	The classification of a technique depends on how you think about it	43

### Chapter 4 Bug Advocacy

Lesson 55	You are what you write	65
Lesson 56	Your advocacy drives the repair of the bugs you report	66
Lesson 57	Make your bug report an effective sales tool	66
Lesson 58	Your bug report is your representative	67
Lesson 59	Take the time to make your bug reports valuable	68
Lesson 60	Any stakeholder should be able to report a bug	68
Lesson 61	Be careful about rewording other people's bug reports	69
Lesson 62	Report perceived quality gaps as bugs	69
Lesson 63	Some stakeholders cannot report bugs—you're their proxy	69
Lesson 64	Draw the affected stakeholder's attention to controversial bugs	70
Lesson 65	Never use the bug-tracking system to monitor programmers' performance	70
Lesson 66	Never use the bug-tracking system to monitor testers' performance	71
Lesson 67	Report defects promptly	71
Lesson 68	Never assume that an obvious bug has already been filed	71
Lesson 69	Report design errors	71

Lesson 70	Extreme-looking bugs are potential security flaws	73
Lesson 71	Uncorner your corner cases	73
Lesson 72	Minor bugs are worth reporting and fixing	74
Lesson 73	Keep clear the difference between severity and priority	75
Lesson 74	A failure is a symptom of an error, not the error itself	76
Lesson 75	Do follow-up testing on seemingly minor coding errors	76
Lesson 76	Always report nonreproducible errors; they may be time bombs	77
Lesson 77	Nonreproducible bugs are reproducible	78
Lesson 78	Be conscious of the processing cost of your bug reports	79
Lesson 79	Give special handling to bugs related to the tools or environment	80
Lesson 80	Ask before reporting bugs against prototypes or early private versions	81
Lesson 81	Duplicate bug reports are a self-correcting problem	82
Lesson 82	Every bug deserves its own report	82
Lesson 83	The summary line is the most important line in the bug report	83
Lesson 84	Never exaggerate your bugs	83
Lesson 85	Report the problem clearly, but don't try to solve it	84
Lesson 86	Be careful of your tone. Every person you criticize will see the report	85
Lesson 87	Make your reports readable, even to people who are exhausted and cranky	85
Lesson 88	Improve your reporting skills	86
Lesson 89	Use market or support data when appropriate	86
Lesson 90	Review each other's bug reports	87
Lesson 91	Meet the programmers who will read your reports	87
Lesson 92	The best approach may be to demonstrate your bugs to the programmers	88
Lesson 93	When the programmer says it's fixed, make sure it isn't still broken	88
Lesson 94	Verify bug fixes promptly	88
Lesson 95	When fixes fail, talk with the programmer	89
Lesson 96	Bug reports should be closed by testers	89
Lesson 97	Don't insist that every bug be fixed. Pick your battles	90
Lesson 98	Don't let deferred bugs disappear	90
Lesson 99	Testing inertia should never be the cause of bug deferral	91
Lesson 100	Appeal bug deferrals immediately	91
Lesson 101	When you decide to fight, decide to win!	91

### Chapter 5 Automating Testing

Lesson 102	Speed the development process instead of trying to save a few dollars on testing	94
Lesson 103	Expand your reach instead of trying to repeat the same tests over and over	95
Lesson 104	Select your automation strategy based on your context	96
Lesson 105	Don't mandate 100 percent automation	97
Lesson 106	A test tool is not a strategy	98
Lesson 107	Don't automate a mess	98
Lesson 108	Don't equate manual testing to automated testing	99
Lesson 109	Don't estimate the value of a test in terms of how often you run it	100
Lesson 110	Automated regression tests find a minority of the bugs	101
Lesson 111	Consider what bugs you <i>aren't</i> finding while you automate tests	101
Lesson 112	The problem with bad automation is that no one may notice	102
Lesson 113	Capture replay fails	103

Lesson 114	Test tools are buggy	104
Lesson 115	User interfaces change	106
Lesson 116	Select GUI test tools based on compatibility, familiarity, and service	107
Lesson 117	Automated regression tests die	108
Lesson 118	Test automation is a software development process	109
Lesson 119	Test automation is a significant investment	109
Lesson 120	Test automation projects require skills in programming, testing, and project management	110
Lesson 121	Use pilot projects to prove feasibility	111
Lesson 122	Have testers and programmers charter automation projects	111
Lesson 123	Design automated tests to facilitate review	112
Lesson 124	Don't skimp on automated test design	112
Lesson 125	Avoid complex logic in your test scripts	113
Lesson 126	Don't build test libraries simply to avoid repeating code	113
Lesson 127	Data-driven test automation makes it easy to run lots of test variants	114
Lesson 128	Keyword-driven test automation makes it easy for nonprogrammers to create tests	115
Lesson 129	Use automated techniques to generate test inputs	116
Lesson 130	Separate test generation from test execution	117
Lesson 131	Use standard scripting languages	117
Lesson 132	Automate tests using programming interfaces	119
Lesson 133	Encourage the development of unit test suites	120
Lesson 134	Beware of using automators who don't understand testing	121
Lesson 135	Avoid automators who don't respect testing	122
Lesson 136	Testability is often a better investment than automation	122
Lesson 137	Testability is visibility and control	123
Lesson 138	Start test automation early	124
Lesson 139	Give centralized automation teams clear charters	125
Lesson 140	Automate for immediate impact	126
Lesson 141	You may have more test tools than you realize	126

### Chapter 6 Documenting Testing

Lesson 142	To apply a solution effectively, you need to understand the problem clearly	131
Lesson 143	Don't use test documentation templates: A template won't help unless you don't need it	131
Lesson 144	Use test documentation templates: They foster consistent communication	132
Lesson 145	Use the IEEE Standard 829 for test documentation	132
Lesson 146	Don't use the IEEE Standard 829	133
Lesson 147	Analyze your requirements before deciding what products to build; this applies as much to your documentation as to your software	136
Lesson 148	To analyze your test documentation requirements, ask questions like the ones in this list	136
Lesson 149	Summarize your core documentation requirements in one sentence with no more than three components	141

## Chapter 7 Interacting with Programmers

Lesson 150	Understand how programmers think	144
Lesson 151	Develop programmers' trust	145
Lesson 152	Provide service	145
Lesson 153	Your integrity and competence will demand respect	146
Lesson 154	Focus on the work, not the person	147
Lesson 155	Programmers like to talk about their work. Ask them questions	148
Lesson 156	Programmers like to help with testability	149

## Chapter 8 Managing the Testing Project

Lesson 157	Create a service culture	151
Lesson 158	Don't try to create a control culture	152
Lesson 159	Develop the power of the king's ear	152
Lesson 160	You manage the subproject that provides testing services, not the development project	153
Lesson 161	All projects evolve. Well-run projects evolve well	154
Lesson 162	There are always late changes	154
Lesson 163	Projects involve a tradeoff among features, reliability, time, and money	155
Lesson 164	Let the project manager choose the project lifecycle	156
Lesson 165	Waterfall lifecycles pit reliability against time	156
Lesson 166	Evolutionary lifecycles pit features against time	158
Lesson 167	Be willing to allocate resources to the project early in development	159
Lesson 168	Contract-driven development is different from market-seeking development	160
Lesson 169	Ask for testability features	161
Lesson 170	Negotiate the schedules for builds	161
Lesson 171	Understand what programmers do (and don't do) before delivering builds	162
Lesson 172	Be prepared for the build	162
Lesson 173	Sometimes you should refuse to test a build	162
Lesson 174	Use smoke tests to qualify a build	163
Lesson 175	Sometimes, the right decision is to stop the test and fix cycle and redesign the software	163
Lesson 176	Adapt your processes to the development practices that are actually in use	164
Lesson 177	"Project documents are interesting fictions: Useful, but never sufficient"	165
Lesson 178	Don't ask for items unless you will use them	165
Lesson 179	Take advantage of other sources of information	166
Lesson 180	Flag configuration management problems to the project manager	167
Lesson 181	Programmers are like tornadoes	168
Lesson 182	Great test planning makes late changes easy	168
Lesson 183	Test opportunities arise whenever one person hands off an artifact to another	170
Lesson 184	There is no universal formula for knowing how much testing is enough	170
Lesson 185	"Enough testing" means "enough information for my clients to make good decisions"	170
Lesson 186	Never budget for just two testing cycles	171

Lesson 187	To create a schedule for a set of tasks, estimate the amount of time needed for each task	172
Lesson 188	The person who will do the work should tell you how long a task will take	173
Lesson 189	There is no right ratio of testers to other developers	174
Lesson 190	Trade tasks or transfer people from tasks that they are failing at	174
Lesson 191	Rotate testers across features	175
Lesson 192	Try testing in pairs	175
Lesson 193	Assign a bug hunter to the project	176
Lesson 194	Charter testing sessions, especially exploratory testing sessions	176
Lesson 195	Test in sessions	177
Lesson 196	Use activity logs to reveal the interruptions that plague testers' work	177
Lesson 197	Regular status reports are powerful tools	178
Lesson 198	There's nothing more dangerous than a vice president with statistics	179
Lesson 199	Be cautious about measuring the project's progress in terms of bug counts	180
Lesson 200	The more independent coverage measures you use, the more you know	181
Lesson 201	Use a balanced scorecard to report status on multiple dimensions	182
Lesson 202	Here's a suggested structure for a weekly status report	183
Lesson 203	A project dashboard is another useful way for showing status	184
Lesson 204	Milestone reports are useful when milestones are well defined	185
Lesson 205	Don't sign-off to approve the release of a product	186
Lesson 206	Do sign-off that you have tested a product to your satisfaction	186
Lesson 207	If you write a release report, describe your testing work and results, not your opinion of the product	187
Lesson 208	List unfixed bugs in the final release report	187
Lesson 209	A useful release report lists the 10 worst things critics might say	187

### Chapter 9 Managing the Testing Group

Lesson 210	Mediocrity is a self-fulfilling prophecy	189
Lesson 211	Treat your staff as executives	190
Lesson 212	Read your staff's bug reports	191
Lesson 213	Evaluate your staff as executives	191
Lesson 214	If you really want to know what's going on, test with your staff	193
Lesson 215	Don't expect people to handle multiple projects efficiently	193
Lesson 216	Build your testing staff's domain expertise	194
Lesson 217	Build your testing staff's expertise in the relevant technology	194
Lesson 218	Work actively on skills improvement	195
Lesson 219	Review technical support logs	195
Lesson 220	Help new testers succeed	195
Lesson 221	Have new testers check the documentation against the software	196
Lesson 222	Familiarize new testers with the product through positive testing	197
Lesson 223	Have novice testers edit old bug reports before writing new ones	197
Lesson 224	Have new testers retest old bugs before testing for new bugs	197
Lesson 225	Don't put novice testers on nearly finished projects	198
Lesson 226	The morale of your staff is an important asset	199
Lesson 227	Don't let yourself be abused	200
Lesson 228	Don't abuse your staff with overtime	200
Lesson 229	Don't let your staff be abused	202

Lesson 230	Create training opportunities	202
Lesson 231	Your hiring decisions are your most important decisions	203
Lesson 232	Hire contractors to give you breathing room during recruiting	203
Lesson 233	Rarely accept rejects from other groups into testing	203
Lesson 234	Plan in terms of the tasks you need to do in your group and the skills needed to do them	204
Lesson 235	Staff the testing team with diverse backgrounds	204
Lesson 236	Hire opportunity candidates	205
Lesson 237	Hire by consensus	206
Lesson 238	Hire people who love their work	206
Lesson 239	Hire integrity	206
Lesson 240	During the interview, have the tester demonstrate the skills you're hiring him for	206
Lesson 241	During the interview, have the tester demonstrate skills he'll actually use on the job over informal aptitude tests	207
Lesson 242	When recruiting, ask for work samples	207
Lesson 243	Hire quickly after you make up your mind	208
Lesson 244	Put your hiring promises in writing and keep them	208

### **Chapter 10 Your Career in Software Testing**

Lesson 245	Choose a career track and pursue it	209
Lesson 246	Testers' incomes can be higher than programmers' incomes	211
Lesson 247	Feel free to change your track and pursue something else	212
Lesson 248	Whatever path you take, pursue it actively	212
Lesson 249	Extend your career beyond software testing	213
Lesson 250	Extend your career beyond your company	213
Lesson 251	Conferences are for conferring	214
Lesson 252	Lots of other companies are as screwed up as yours	214
Lesson 253	If you don't like your company, look for a different job	215
Lesson 254	Be prepared in case you have to bet your job (and lose)	215
Lesson 255	Build and maintain a list of companies where you'd like to work	216
Lesson 256	Build a portfolio	216
Lesson 257	Use your resume as a sales tool	217
Lesson 258	Get an inside referral	218
Lesson 259	Research salary data	218
Lesson 260	If you're answering an advertisement, tailor your answer to the advertisement	218
Lesson 261	Take advantage of opportunities to interview	218
Lesson 262	Learn about companies when you apply for jobs with them	219
Lesson 263	Ask questions during job interviews	220
Lesson 264	Negotiate your position	221
Lesson 265	Be cautious about Human Resources	223
Lesson 266	Learn Perl	223
Lesson 267	Learn Java or C++	223
Lesson 268	Download demo copies of testing tools and try them out	224
Lesson 269	Improve your writing skills	224
Lesson 270	Improve your public speaking skills	224

Lesson 271	Think about getting certified	224
Lesson 272	If you can get a black belt in only two weeks, avoid fights	226
Lesson 273	A warning about the efforts to license software engineers	226

### **Chapter 11 Planning the Testing Strategy**

Lesson 274	Three basic questions to ask about test strategy are "why bother?", "who cares?", and "how much?"	231
Lesson 275	There are many possible test strategies	232
Lesson 276	The real test plan is the set of ideas that guides your test process	233
Lesson 277	Design your test plan to fit your context	233
Lesson 278	Use the test plan to express choices about strategy, logistics, and work products	234
Lesson 279	Don't let logistics and work products blind you to strategy	235
Lesson 280	How to lie with test cases	235
Lesson 281	Your test strategy is more than your tests	236
Lesson 282	Your test strategy explains your testing	236
Lesson 283	Apply diverse half-measures	237
Lesson 284	Cultivate the raw materials of powerful test strategies	238
Lesson 285	Your first strategy on a project is always wrong	238
Lesson 286	At every phase of the project, ask yourself "what can I test now and how can I test it?"	239
Lesson 287	Test to the maturity of the product	239
Lesson 288	Use test levels to simplify discussions of test complexity	241
Lesson 289	Test the gray box	242
Lesson 290	Beware of ancestor worship when reusing test materials	242
Lesson 291	Two testers testing the same thing are probably not duplicating efforts	243
Lesson 292	Design your test strategy in response to project factors as well as product risks	243
Lesson 293	Treat test cycles as the heartbeat of the test process	244