

THE POWER OF GO TESTS

“Engaging, friendly, informative, and snappy”

—Kevin Cunningham



Go 1.22

PROGRAMMING WITH CONFIDENCE IN GO

JOHN ARUNDEL

Contents

Praise for <i>The Power of Go: Tests</i>	7
Introduction	8
1. Programming with confidence	10
Self-testing code	11
The adventure begins	12
Verifying the test	14
Running tests with <code>go test</code>	15
Using <code>cmp.Diff</code> to compare results	16
New behaviour? New test.	18
Test cases	19
Adding cases one at a time	21
Quelling a panic	23
Refactoring	24
Well, that was easy	26
Sounds good, now what?	27
2. Tools for testing	29
Go's built-in testing facilities	30
Writing and running tests	31
Interpreting test output	32
The "magic package": testing functions that don't exist	34
Validating our mental models	36
Concurrent tests with <code>t.Parallel</code>	36
Failures: <code>t.Error</code> and <code>t.Errorf</code>	37
Abandoning the test with <code>t.Fatal</code>	38
Writing debug output with <code>t.Log</code>	39
Test flags: <code>-v</code> and <code>-run</code>	41
Assistants: <code>t.Helper</code>	42
<code>t.TempDir</code> and <code>t.Cleanup</code>	43
Tests are for failing	44
Detecting useless implementations	45
Feeble tests	46
Comparisons: <code>cmp.Equal</code> and <code>cmp.Diff</code>	48
3. Communicating with tests	50
Tests capture intent	50
Test names should be sentences	51

Failures are a message to the future	53
Are we testing all important behaviours?	54
The power of combining tests	55
Reading tests as docs, with <code>gotestdox</code>	57
Definitions: “does”, not “should”	59
A sentence is about the right size for a unit	60
Keeping behaviour simple and focused	61
Shallow abstractions: “Is this even worth it?”	62
Don’t worry about long test names	62
Crafting informative failure messages	63
Exercising failure messages	65
Executable examples	66
4. Errors expected	73
Ignoring errors is a mistake	73
Unexpected errors should stop the test	75
Error behaviour is part of your API	77
Simulating errors	79
Testing that an error is not <code>nil</code>	81
String matching on errors is fragile	83
Sentinel errors lose useful information	84
Detecting sentinel errors with <code>errors.Is</code>	86
Wrapping sentinel errors with dynamic information	89
Custom error types and <code>errors.As</code>	90
Conclusions	92
5. Users shouldn’t do that	94
Constructing effective test inputs	95
User testing	97
Crafting bespoke bug detectors	99
Table tests and subtests	100
Table tests group together similar cases	102
Using dummy names to mark irrelevant test inputs	103
Outsourcing test data to variables or functions	105
Loading test data from files	108
Readers and writers versus files	109
The filesystem abstraction	109
Using <code>t.TempDir</code> for test output with cleanup	110
Managing golden files	110
Dealing with cross-platform line endings	112
What about the inputs you didn’t think of?	113
6. Fuzzy thinking	114
Generating random test inputs	115
Randomly permuting a set of known inputs	116
Property-based testing	116

Fuzz testing	118
The fuzz target	119
Running tests in fuzzing mode	120
Failing inputs become static test cases	121
Fuzzing a risky function	122
Adding training data with <code>f.Add</code>	124
A more sophisticated fuzz target	124
Using the fuzzer to detect a panic	127
Detecting more subtle bugs	130
What to do with fuzz-generated test cases	131
Fixing the implementation	132
7. Wandering mutants	135
What is test coverage?	135
Coverage profiling with the go tool	136
Coverage is a signal, not a target	138
Using “bebugging” to discover feeble tests	140
Detecting unnecessary or unreachable code	142
Automated mutation testing	143
Finding a bug with mutation testing	144
Running <code>go-mutesting</code>	145
Introducing a deliberate bug	146
Interpreting <code>go-mutesting</code> results	148
Revealing a subtle test febleness	149
Fixing up the function	151
Mutation testing is worth your while	152
8. Testing the untestable	153
Building a “walking skeleton”	153
The first test	155
Solving big problems	156
Designing with tests	156
Unexported functions	158
Concurrency	159
Concurrency safety	167
Long-running tasks	171
User interaction	172
Command-line interfaces	178
9. Flipping the script	182
Introducing <code>testscript</code>	183
Running programs with <code>exec</code>	184
Interpreting <code>testscript</code> output	185
The <code>testscript</code> language	187
Negating assertions with the <code>!</code> prefix	188
Passing arguments to programs	190

Testing CLI tools with <code>testscript</code>	191
Checking the test coverage of scripts	195
Comparing output with files using <code>cmp</code>	196
More matching: <code>exists</code> , <code>grep</code> , and <code>-count</code>	198
The <code>txtar</code> format: constructing test data files	199
Supplying input to programs using <code>stdin</code>	201
File operations	203
Differences from shell scripts	204
Comments and phases	205
Conditions	206
Setting environment variables with <code>env</code>	207
Passing values to scripts via environment variables	208
Running programs in background with <code>&</code>	209
The standalone <code>testscript</code> runner	211
Test scripts as issue repros	212
Test scripts as... tests	213
Conclusion	215
10. Dependence day	216
Just don't write untestable functions	217
Reduce the scope of the dependency	218
Be suspicious of dependency injection	219
Avoid test-induced damage	220
"Chunk" behaviour into subcomponents	221
Reassemble the tested chunks	223
Extract and isolate the key logic	224
Isolate by using an adapter	225
Example: a database adapter	226
Fakes, mocks, stubs, doubles, and spies	232
Don't be tempted to write mocks	233
Turn time into data	234
11. Suite smells	240
No tests	240
Legacy code	242
Insufficient tests	243
Ineffective code review	245
Optimistic tests	246
Persnickety tests	251
Over-precise comparisons	254
Too many tests	255
Test frameworks	256
Flaky tests	258
Shared worlds	260
Failing tests	261
Slow tests	262

A fragrant future	264
About this book	265
Who wrote this?	265
Cover photo	265
Feedback	266
Join my Go Club	266
For the Love of Go	266
The Power of Go: Tools	266
Know Go: Generics	267
Further reading	267
Credits	267
Acknowledgements	268

Praise for *The Power of Go: Tests*

Brilliant. I read it with genuine pleasure. Well written, clear, concise, and effective.

—Giuseppe Maxia

I'd happily pay three times the price for John's books.

—Jakub Jarosz

The writing style is engaging, friendly, informative, and snappy.

—Kevin Cunningham

John's books are so packed with information, I learn something new every time I re-read them.

—Miloš Žižić

A great read—it's a treasure trove of knowledge on not just testing, but software design in general. Best of all, it's completely language-agnostic.

—Joanna Liana

I really enjoyed this book. The humour makes learning Go a lot more fun.

—Sean Burgoyne

This could get someone fired!

—David Bailey

The best introduction to mutation testing. John's writing style made me smirk, smile and vigorously nod my head as I was reading.

—Manoj Kumar