How To Test It: A Short Dictionary of Modern Software Testing Heuristics

Alex Groce

November 14, 2018

Preface

Acknowledgements

This book takes its title, much of its structure, and some of its approach from George Polya's famous *How to Solve It* [Polya(2004)]; it takes its existence from a largely abandoned project, started by John Regehr and the author, to write a book on testing, begun in 2013. Thanks are also due to my wife, Josie Holmes, who pushed me to return to this effort.

Part I

Introduction

Chapter 1

Introduction: What is Testing?

Software testing is a special instance of a more general concept: the idea of *testing* a specific claim, a physical object, a scientific theory, or a more abstract and general idea. At heart, testing involves *probing for weakness.* We say that a soldier is tested by combat, a scientific theory is tested by an experiment [Popper(1959)], or (most commonly) a student is tested by, well, a test. Testing is inherently adversarial: a soldier is tested by combat precisely because it is a matter of "trial by fire" where there is a possibility of *failure.* A soldier may be unable to fire on command, unable to advance on a fortified position, or may even turn and run. A scientific theory may make a prediction that is falsified. A student may fail to make a passing grade. To start this book on a grim note, testing is an inherently *skeptical*, *adversarial*, *harsh*, and even (in a certain sense) *cruel* practice.

It is also, of course, essential. One way to look at testing is to consider the case of the soldier tested by combat, though shifted to the implements of war to better match the problem of software testing; as John Paul Jones wrote, "I wish to have no connection with any ship that does not sail fast; for I intend to go in harm's way." That is to say, the purpose of a vessel in a navy is to engage in dangerous behavior. Yet, even John Paul Jones would certainly prefer to discover if a ship does in fact sail fast in a non-harm's-way trial, and obtain a different ship, or alter the arrangement of sails at least, if it does not. The art and science of testing, in software or otherwise, is generally a response to the fact that reality will impose a most vicious kind of testing indeed on all of our creations; we would prefer to find out if the subjects of this unavoidable testing are up to the task in a less consequential way, and have a chance to fix any leaks before sailing into deep waters. Great Nature, or at least some hapless software user or malicious hacker, will put every important piece of code into "combat" — we need to train our "soldiers" to meet enemy fire with honor and skill.

Testing is, also, of course, a great deal of fun, if you have the right kind of mindset. Creation is deeply engaging, but destruction also has its charms. Many people (not all of them unpleasant people in comment boxes on the Internet) enjoy posing a question that undermines an incorrect argument; arguably, at least since Socrates, the philosophical mind has concerned itself with finding "tests" that defeat weak ideas.

1.1 Testing vs. Proof

In the world of mathematics, which has fundamental connections to software (since computer programs are mathematical entities, at heart), testing is intimately connected to the notion of proof. Consider a mathematical claim, such as Fermat's Last Theorem, which states that no three positive integers a, b, and c exist that satisfy the equation $a^n + b^n = c^n$ for any integer $n > 2$. There are two obvious things to "do" with such a claim. One of these is to test it: to attempt to produce an a, b, c, and n that show the claim is false. The other, of course, is to *prove* that no such tuple (a, b, c, n) exists.

Part II

How to Test It

Chapter 2

A Dialogue: How to Test It

Part III

A Short Dictionary of Modern Software Testing Heuristics

Chapter 3

Abstraction

Chapter 4

Assertions

Chapter 5

Automation

Chapter 6

Bug Triage

Chapter 7

Bugs

Chapter 8

Causality

Chapter 9

Crashes

Chapter 10

Concurrency

Chapter 11

Coverage

Chapter 12

Debugging

Chapter 13

Determinism

Chapter 14

Differential Testing

Chapter 15

Diversity

Chapter 16

Falsification

Chapter 17

Fault Injection

Chapter 18

Fault Localization

Chapter 19

Fuzzing

Chapter 20

Machine Learning

Chapter 21

Metamorphic Testing

Chapter 22

Modeling

Chapter 23

Model Checking

Chapter 24

Monitoring

Chapter 25

Mutation

Chapter 26

Oracles

Chapter 27

Parameterized Unit Testing

Chapter 28

Performance

Chapter 29

Property-Driven Testing

Chapter 30

Random Testing

Chapter 31

Regression

Chapter 32

Reduction

Chapter 33

Security

Chapter 34

Specification

Chapter 35

Static Analysis

Chapter 36

Symbolic Execution

Chapter 37

Unit Testing

Bibliography

[Polya(2004)] George Polya. *How to solve it: A new aspect of mathematical method*. Number 246. Princeton university press, 2004.

[Popper(1959)] Karl Popper. *The Logic of Scientific Discovery*. Hutchinson, 1959.