Alex Groce (agroce@gmail.com), Northern Arizona University

Karl Popper's *The Logic of Scientific Discovery* is the most important work addressing the philosophy of science (as opposed to sociology of science, where Kuhn competes) written in the 20th century. It is, therefore, a perfect *Passages* book.

Software engineering can be divided, like Gaul (or every other engineering discipline, for that matter), into three parts. First, there is the creative design component. Second, there is the technical engineering challenge. Third, there is the management problem, if your engineering task is much larger than a small program a single programmer can develop in a few afternoons.

Fourth, there is… Wait, didn't we say three parts? The fourth part is an oddity, in that it is the problem of testing (or, if you're feeling frisky, verifying) software. This element exists in all engineering enterprises, but it looms especially large in software, in part because it is so difficult and in part because it is unusually *possible* in software. It is fundamentally different than the other three core aspects in that it is both an artform and, quite specifically and clearly a well-defined problem in applied epistemology. And, to loop back to Popper, there is another name for applied epistemology: science.

The core point of Popper's book is that induction is not sound. That is, Popper, like Hume (but in a more optimistic line of thinking) says that you cannot reason from any number of particular observations to a justified belief in a universal truth, in a purely rational fashion. Prior to Popper, most efforts to define scientific method, or place science on a more sure-footed philosophical basis, had attempted to argue for a principle of induction. Popper, instead, argued that science is always provisional and partial, never reaching conclusions that cannot be assailed. In fact, for Popper, the hallmark of scientific theories is that they are welcoming of assault: a theory that cannot be falsified by some experiment is not scientific at all. If you know only one thing about Popper, it is that he said science is about falsification.

This should be familiar terrain to software testers. No number of executions where we observe a program running correctly should convince us that it does what it is supposed to (or, more broadly, that we know what the program does); rather, determined efforts to falsify our current understanding of a program's correctness, or our model of its behavior, are required to slowly refine our notion of what is really going on with a program. Simply passively observing empirical events (induction pure and simple) is of very little value, usually; active effort to distinguish between theory and reality, where the theory constantly crumbles and has to be re-made (or, admittedly, in an option not available to Popper's scientist, the program modified to actually work!) are the path taken by the good tester, the good scientist. Moreover, just as a scientific theory that cannot be "defeated" by some experiment is not very scientific, a theory about a program's behavior that could not be proven wrong by some counterexample is of little use to anyone at all.

The idea that Karl Popper's notions about empirical science as demarcated by an emphasis on

falsification have something to say about software testing is pretty commonplace, at least among people who might think about such things. Classic books on testing, such as *Lessons Learned in Software Testing* will almost always drop Popper's name or even devote a few pages to testing as a kind of Popperian enterprise. "Everyone" knows a test should aim not to demonstrate that a program works, but to falsify the claim that it works. Good tests have high potential to prove the program wrong, just as (for Popper) good experiments are those that might prove a theory wrong. However, actually reading Popper's book, it is easy to arrive at new insights that go beyond this somewhat basic (if useful to keep in mind) point.

For example, it is not completely absent from the literature (see Aichernig's work for example), but Popperism as a way to think about mutation testing has not been nearly as much remarked upon. It's great to use tests to doubt your code, but who tests the tests? Who tests the specification itself? Who watches the watchmen? A key point of Popper's ideas is that there is no non-provisional, not-to-be-tested statement in science, not even "basic" empirical observations. Similarly, in testing, we can decide to also distrust the tests themselves. Program mutations, small random syntactic changes to the program, can falsify not just the program, but its test suite: if you change your program in a random way, and your test suite cannot tell you've changed your program, then your test suite is, in a real (and now quite specific) way lacking. That's why program mutants are a useful and distinct concept, not just an additional hoop for people so bored they have already obtained 100% code coverage to jump through.

The more general statement produced by the analogy between testing and science is interesting to think about. Consider a program, with a passing test suite, that you think is correct. Any semantically meaningful change to a program that you make, that does not cause the test suite to fail, forces you to accept at least one of three claims:

1. The test suite was not complete in specifying interesting behaviors of the program (maybe you didn't even run the changed code). You have falsified your "experimental apparatus" in a sense, might be the correct Popper/science analogy. You need better instruments!

2. The specification was not complete in unambiguously specifying what the program should do. This could be harmless (genuine room for variance), or a hole in your concept of correctness. In this case the spec/oracle is in the role of a scientific theory for Popper, which needs to be refined to make better predictions in order to be potentially falsified.

3. The program and specification were both just wrong. You falsified your code (which is also in the role of a theory).

Actually reading Popper's book makes thinking about such things easy, and makes testing seem more exciting. It can suggest heretical thoughts as well: it is fairly clear from Popper's early chapters that he rejects what might seem to be the equivalent of "test-driven development" which would be "experiment-driven science." Is TDD wrong-headed, for the reasons Popper would suggest science could not operate in a purely experiment-driven way? Or is the analogy

broken?

I won't claim the book is easy reading, suitable for a nice evening on the beach, though the first few critical chapters are not unusually difficult, due to Popper's focus on clear, understandable ideas and distinctions. Later chapters, such as those considering how to determine which of two scientific theories is more falsifiable, are worthwhile but less fun, and even Popper suggests that readers skip a large chunk of the book (defining probability from the ground up) on a first reading. Be sure, however much you skip of the middle, to make it to the last chapter, however, which summarizes all that has gone before, and is (to one in the right frame of mind) a stirring, rousing, "once more unto the breach" speech, a kind of Churchill before the Battle of Britain, for scientists and their fellow falsifiers, software testers:

> "With the idol of certainty (including that of degrees of imperfect certainty or probability) there falls one of the defences of obscurantism which bar the way of scientific advance. For the worship of this idol hampers not only the boldness of our questions, but also the rigour and the integrity of our tests. The wrong view of science betrays itself in the craving to be right; for it is not his possession of knowledge, of irrefutable truth, that makes the man of science, but his persistent and recklessly critical quest for truth.
>
> Has our attitude, then, to be one of resignation? Have we to say that science can fulfil only its biological task; that it can, at best, merely prove its mettle in practical applications which may corroborate it? Are its intellectual problems insoluble? I do not think so. Science never pursues the illusory aim of making its answers final, or even probable. Its advance is, rather, towards an infinite yet attainable aim: that of ever discovering new, deeper, and more general problems, and of subjecting our ever tentative answers to ever renewed and ever more rigorous tests."