When I test, I try to think of three aspects of my test. Let's call them, for the sake of argument, ***Action***, ***Information***, and ***Observation***.

By ***action***, I mean things to do. Things that are done. Things that have been done. Actions: everything is steady in some sense, until something *acts* on something else. Often, I'll think of actions that the tester takes - but I'll also think of actions that the system takes, that a part of the system takes, or that are taken by something outside the system.

By ***information***, I mean things that are. Things already in the system, the state that system is in. Not just the stuff I choose to give to the system. I could call it *data* - and it often is, but the label is over-loaded and limiting.

By ***observation***, I mean things that are perceived. Whatever tool captures information after an action - my eyes, a database monitor, the smoke detector connected to the office sprinkler system - that information has to be *perceived* to be part of the test.

o O o

It's easy to script ***actions***. Many written scripts dictate action rather precisely. ***Information*** is harder to write down completely; most scripts leave at least some information to the tester - or to chance. Scripts can suggest what to ***observe***, but there's simply no way to describe all potentially useful observations in advance.

A three-ring-binder scripted test leaves important elements *of* the test *to* the tester. Even if the ***actions*** are set in stone, the ***information*** may change - by choice or by circumstance - and the tester can broaden their ***observations*** if they choose.

Exploration is an important part of the skilled manual execution of scripted tests: The tester influences the test design during the test by choosing what to observe, how to treat the information, be the actions ever so precisely prescribed.

o O o

The word *prescribed* is a lovely word for the Latin lover. *Scribere* meaning 'write', *prae* meaning 'before'. A test script is exactly that; a writing-down-before, of the test. The amateur etymologist in all of us will notice also that *precision* shares the prefix. Its root, however, is *caedere* - to cut short.

o O o

It is possible, and common, to write a test script that is much more precisely prescribed than a manual test script. Automated tests are, by necessity, both precise and prescribed. That's not to say that a sophisticated automated test script has only one path through the system, only one set of data; an individual

script can be re-used for a range of similar tests. Its observations, however, are precise. They are cut short.

An automated test won't tell you that the system's slow, unless you tell it to look in advance. It won't tell you that the window leaves a persistent shadow, that every other record in the database has been trashed, that even the false are returning true, unless it knows where to look, and what to look for. Sure, you may notice a problem as you dig through the reams of data you've asked it to gather, but then we're back to exploratory techniques again.

I have a lot of time for a certain pure ideal in automated test scripts. They should run any time and often, and without hassling the people who run them. This applies to a great regression test pack as much as it applies to a fine suite of test-first unit tests. They're written to run time and time again, at a greater and greater distance from the point where they were designed, and still bring us useful information. That information is information about **value**. It's not just that the build survived the latest changes, it's that the system does everything we've asked it to. It's not that the regression tests have found no new bugs, it's that the users can still rely on the system to support them in their work.

We *need* precisely prescribed scripted tests. They're great. They tell us about value - the value that exists, and continues to exist, in the artefact we're testing. If what is valuable stays much the same, then so do the scripts - write once, run forever.

o O o

It is also possible, and common, to test in a way that is not prescribed. An exploratory test needs no script, no chosen set of actions. Choice of actions is up to the tester, at the point of testing. Choice of information, of observation, is limited not by pre-existing design, but by opportunity and resource. Moment to moment, the tester chooses what to do, what to do it with, how to check what's happened. Interesting things will be examined in more detail, weaknesses tried, doorknobs jiggled. The tester chooses to try two things together that between them open the system to a world of pain. The tester chooses to use *this* information, with *that* action, not to just to see what's desirable, but what's *possible*. The exploratory tester focuses on **risk**.

We *need* exploratory tests. They're great. They tell us about risk - unexpected, unpredicted, emergent - that goes hand-in-hand with the system that has been delivered. Exploratory tests are immediate, of the moment. The risk is known - and you'll not need to test for it again until you've addressed it, and *written* a test to show you that it's gone.

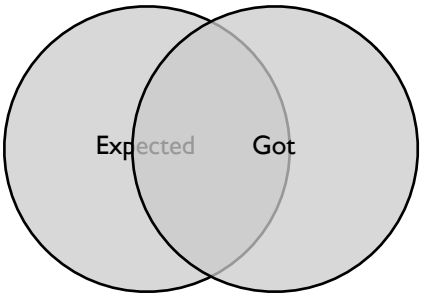
o O o

Let's summarise, briefly. Some tests are designed to find risks. They're made on-the-fly and run once. Some are designed to tell us about retained value.

They're made once, and run forever after. You need *both*: they tell you different things.

o O o

So much for the ideal. Now for a different perspective. We've all seen diagram 1 of software testing:



Let's look at the left hand circle; our expectations. These things we expect, those we don't. If we're going to write a set of tests, we're going to scan through the finite set of things we expect.

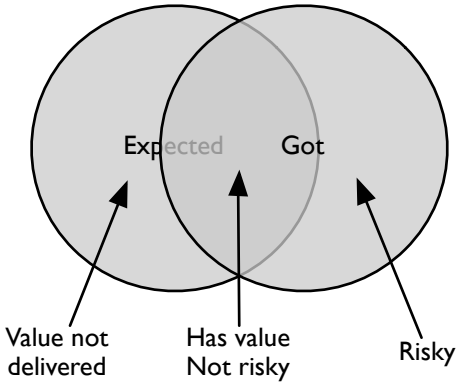
Let's look at the right hand circle; the deliverable. These things we've got, those we haven't. If we're going to do a set of tests, we're going to scan through the finite set of what we've got.

The diagram splits the world into four regions. First, let's deal with the overlap. This chunk is things we expect, that we know we've got. Both sets of tests will identify that our requirements are met by our deliverable. That's duplicate work, but let's not judge (just yet).

There's the region outside both circles. That's all the stuff we didn't want, and haven't got - let's hope not too much work was spent there. It's not exactly a finite set, whichever way you look at it.

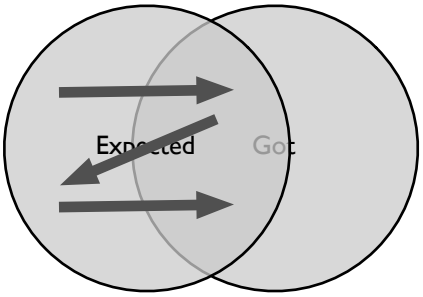
Let's consider the left-hand arc. We expected to get this. We didn't get it. The deliverable is less valuable than we'd hoped.

Now the right-hand arc. "We found the system does *this* and frankly, Bob, that's a bit of a surprise". Is it risky? Yes, in some abstract sense - but now we've found something in that region, we can discover just how risky.

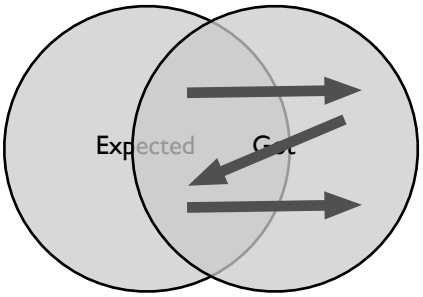


o 0 o

Scanning through the finite set of things we expect can be done before we ever have something to test. It relies on a good understanding of our expectations. If we're going to get the job off the critical path, we need that understanding to be stable. Then we can write down the steps in testing, before we ever test. When we do the test, we'll know about what's there, and what's not. We'll know about value. This is **scripted** testing.



Scanning through the finite set of what we've got requires, as its first step, something to scan. That means we're on the critical path, no way off. We'll need to be fast, and to be fast, we'll have to be prepared, and skilled. We'll find risks, and risks need to be assessed and addressed, so we'll need to be really good, really soon. This is **exploratory** testing.



We *need* both, for a good understanding of risk and value.