The Like-Live Paradox

16 November 2011 · experiment, hindsight, human, missing, mistake, testing

I was recently struck by a glaring difference between how I and a programmer prepared for testing. Unlike the majority of the testing I am involved in, this particular testing ‘phase’ had to be scheduled in advance and we couldn’t “just do it”. This also meant we had more time to prepare and plan than we typically do.

This ‘waiting period’ had its uses. We had time to create tools that might be useful and check the configuration of the systems we would be testing. The team, familiar with the concepts of exploratory testing, were comfortable with an approach that meant we did not spend the majority of the time pre-scripting tests [be they coded or in a spreadsheet etc]. We did however build a high-level checklist of areas to test and used this to drive our program of tools and configuration checking/fixing/building.

The key difference I observed was the absolute nature of the programmers comparisons between our test systems and our live-production systems. As a tester, I was used to the usual concerns of how test systems differ to production systems. Testers, me included, often go through great pains to obtain and maintain ‘like-live’ test systems.

The absolute nature of their concerns, seemed to swing from faith to despair. If a particular configuration did not appear like ‘live’ then few conclusions might be drawn from any future results. And conversely, If some other setup did match the ‘live’ system, then great confidence could be ascribed to a particular result.

While they may indeed be correct (we won’t know until the changes are live and have been exercised there). It was the absolute nature of their views that surprised me. As a tester I tend to be more circumspect. I don’t expect my tests to be exactly representative of future live failure scenarios. I realised I see the problem more grey-scale rather than black & white. While I strive to create test systems that are like-live, I am often happy with just knowing the differences.

I also realised that I had often deliberately tried to make my systems less like-live. I often contrived to create un-realistic situations that could have illuminated behaviour that I might not otherwise of seen. After all, I am trying to disprove an assumption or hypothesis. Such an assumption would probably have been based on previously-observed behaviour. Unless I can create previously un-observed behaviour, I am unlikely to see problems with the assumption.

To cause the system to exhibit some strange new behaviour, I frequently need what is often referred to as ‘unrealistic’ inputs.But while these inputs may seem unrealistic to us at the development/test-time, it could be we have just not been around long enough to see the issue arise. Much like the Turkey in Nassim Taleb’s “The Black Swan”, realistic is being fed well every

day. Until the turkey finds-out that it is “Thanks Giving” and its outlook is somewhat bleaker (http://en.wikipedia.org/wiki/Thanksgiving_(United_States)#Foods_of_the_season) .

We might wish that we create perfect tests or “experiments”, and only vary one item at a time, the reality is a little more human. Our tests and apparatus are always flawed. A bug may only be visible because of un-intended human error or a ‘difference’ from how things were meant to be. In fact this is quite likely the case, as homogenous cleanly built test systems will have been ‘used’ in a particular manner many times before our tests. Any glaring bugs in the ‘happy path’ are likely to of been discovered already.

For example, once while testing a network connection time-out, we uncovered a serious issue - by mistakenly trying to configure the system to talk to the wrong server. The new time-out configuration had worked fine when the remote server had been ‘down’. But when the remote server didn’t exist at all, and the system could not even locate a path to it on the network, the time-out was ignored. Unless we had mistakenly misconfigured the software, we may never have found this issue. This tester led process by which we realised that our solution was too narrowly focused, helped the team to expand their solution to fit many wider failure scenarios.

I’ve developed a learned-trust of the differences, knowing both the benefits and perils of un-live-like systems. This sort of depth of experience with uncertainty is another area where testers can improve a teams ability to handle the practical problems of software development. We can add depth and qualification to the information testing delivers, helping our businesses weigh the relative merit of the data. Further to this we can suggest how we might improve the accuracy of the information we deliver, without resorting to absolutes.