Avoiding Wild Goose Chases While Debugging.

04 June 2020 · agile, critical thinking, devops, experiment, exploratory testing

When I’m debugging a complex system, I’m constantly looking for patterns. I just ran this test code… What did I see in the log? I just processed a metric $^&*-load of data, did our memory footprint blip?

I’m probably using every freedom-unit of screen space to tail logs, run a memory usage tool, run an IDE & debugger, watch a trace of API calls, run test code… And I’m doing this over and over. Then I see it. Bingo, that spike in API calls hits only when that process over there jumps to 20% processor usage when the app also throws that error.

Unfortunately, I may have been mistaken. On a sufficiently complex system, the emergent behaviour can approach the appearance of randomness. Combinatorial explosions are for real, and they are happening constantly in your shiny new MacBook.

Multiple comparisons problem - Wikipedia

(https://en.wikipedia.org/wiki/Multiple_comparisons_problem)

My bug isn’t what I think it is. I’m examining so many variables in a system with dozens of subsystems at play, it’s inevitable I will see a correlation. We know this more formally as the multiple comparisons problem.

I

We’ve evolved to spot patterns and find paths in a forest of information. Sadly, we can be prone to see problems where only coincidences have occurred.

In this situation, it’s easy to follow this fresh lead and spend your next half an hour chasing the new truth. We’ve all done it. Experiment taking a moment in review, try a clean dataset or different tool to falsify your conclusion. If your idea stands up to scrutiny, then the next 30 minutes might not be such a waste of my time.

And if you completely disprove your-self? then relax, you’ve exercised superb technical analysis skills. Get a cup of tea, because you’ve earned it.

> It Ain’t What You Don’t Know That Gets You Into Trouble. It’s What You Know for Sure That Just Ain’t So
>
> - (Maybe…) Mark Twain