How to avoid testing in circles.

06 March 2012 · agile, anchoring bias, bias, regression testing, testing

I once had an interesting conversation with a colleague who worked in a company selling hotel room bookings. The problem was interesting. Their profits depended on many factors. Firstly, fluctuating demand e.g.: Holidays, Weekends, Local events etc. Secondly, varying types of demand e.g. Business customers, Tourists, Single night bookings or e.g.: 11 day holidays. They also had multiple types of contracts on the rooms. For some, they might have had the exclusive right to sell [as they had pre-paid], for others they had an option to sell [at a lower profit] etc.

My naive view had been they priced the room bookings at a suitable mark up, upping that markup for known busy times etc. For example a tourist hotel hotel near the Olympics would be a high mark up, the tourist hotel room in winter would have incurred less of a markup. Better to get some money than none at all).

He smiled and said some places do that, but he didn't. He had realised his team had a bias towards making a healthy profit. "That seems errh good…" I replied, not sure what I was missing. He explained the problem wasn't making a profit. They made a profit, They could do that. There's good enough demand, and limited enough supply in a business and tourist centre like London to make a profit. The problem was maximising profit. What he had done, was present the profit to the team as a proportion of the theoretical maximum profit. That is, the profit given the perfect combination of bookings at peak rates.

It was understood that this was an unlikely, but doable, goal. The benefit was the team could more easily see whether they were making as much money as they could. For example, that business hotel room-stock was making £1 million profit (more than the others), but we should be able to get £10 million (Whereas the others are already at 90% of theoretical max profit.).

This struck me as a useful way of looking at the world. Maybe it was just in-tune with my tester mind-set. In software development, we often try and view what we have achieved, and we see the stories we have completed. And when we come to a release at the end of the week, sprint, month, quarter etc, we test those stories. We also fix and test bugs considered important and we usually regression test the system as a whole. At this point, most teams I've worked with, start trying to regression test - but soon end up retesting the same areas, going in circles around the same code-changes.

The problem is often that our view of the system has been primed. We fall foul of the anchoring bias (http://en.wikipedia.org/wiki/Anchoring), and can not easily see that 90% of the system has not been examined. Our testing returns again and again to checking the recent changes and their surrounds. Even when these are probably the best and most recently tested

parts of the system. Much like the person in the audience called to the stage in a magic show - I've been primed "to pick a number any number, could be 5, could be 11 ...any number you like.". I'm unlikely to suggest any negative or very large numbers by the time I reach the stage.

What I've found to be useful is applying the same concept the hotel sales team used. To help reduce that bias, I invert the game. Instead of looking at the SCRUM/KANBAN board or bug tracking system, that lists the known stories or known defects. I look at a different list or even several lists. These are usually either a checklist of system areas or a something deliberately not in the affected system areas. I then pick an item and investigate its behaviour as if it was new. The very fact I'm not repeating the same ground as everyone else is increasing the chances that I will find an issue. Whats better, these are just the sort of not quite-related but somehow broken-indirectly bugs that regression testing is aimed at.

So rather than having a board of defects and stories that you are itching to remove, instead have a board with a card for each section of your software. Divide up your time before the release and start working through the items. You can prioritise your testing however you like, but remember you have already focused a lot of time on certain areas, in specific release-change related testing. What's left are the unexplored areas, the undiscovered bugs.