A Fair Witness

13 May 2011 · bug reports, stranger, testing, witness

About 10 years ago, I was working with a client who were in the process of developing a new ecommerce website. The new website and servers were designed to replace an entire existing suite of systems, and provide a platform for the company's future expansion. As you might imagine, the project was sprawling, new front end servers, new middleware and a host of back-end business to business systems. The project had expanded during its course, as new areas of old functionality were identified and the business requested new systems to support new ventures.

This isn't an unusual scenario for software development projects, it is for exactly this type of situation that many companies now look to agile methodologies for help. But don't worry this isn't a rant about the benefits of one methodology over another. What interested me was the how the project team members performed and viewed their testing efforts.

Each build of the code would include new functionality, [hopefully] ready for testing. As the release date approached the rate of delivery would increase. More builds were delivered to the testers, in an effort to get more feedback, more quickly. As is normal, as we approached a deadline compromises were made. Programmers and testers would have their allotted time reduced. Programmers were given less time to program and unit test, and the testers less time for their own system testing.

The change in how the software and systems were developed was gradual. For a while things seem to continue on as before, maybe with a few more deadlines looking achievable. But once in a while there would be slip ups. For example, a bug may slip into the system, not have been uncovered during testing and have escaped into live. This situation would shocked the team, but luckily no serious damage was done. After-all bugs like this had occurred even when programmers and testers had been given more time, so who knows whether we 'would' of caught it before timelines were trimmed.

As features were 'completed' and the team moved on to new features, an interesting cognitive dissonance took place. Although a particular system was well known to of been coded in a hurry, and pushed live with only minimal testing, I noticed our views changed over time. Over time the dissonance appeared to cause us to assume we 'must of tested it properly' otherwise we wouldn't of released it. Despite at the time of release, every team member being quite uneasy with the release, due to its limited testing.

Another effect was the normalisation of risk, an accommodation we gained to the ever increasing levels of risk we were taking. As bugs were discovered in Production, or systems failed on the production network we gradually came to see these as 'everyday' occurrences.

In a way this manner of thinking was ‘correct’, the bugs, outages and similar incidents were occurring more and more frequently, if not quite everyday.

This had a powerful affect on the testers. Initially they, like everyone involved, were alarmed at the bugs and outages. They gradually adapted to the new norm, Only more spectacular bugs surprised them and stood-out from the noise of the everyday failures. This environment had a detrimental affect on the ability of the testers to test effectively. While still highly skilled, the heuristics they were working with were getting more vague. For example at the start of the project, merely having not had a feature tested was grounds for heated discussion and probable release delay. It gradually became a question of whether any testing was needed for a new or modified feature.

Or for example, an error message in the logs, was originally always worthy of investigation. It was either a ‘real bug’ or at best ‘correct code’ mistakenly reporting errors. Either way it was worth reporting. But when several releases have gone out-the-door, and unfixed error messages are commonplace in the logs, do you report each one? How many were present in the last release? We accepted those errors, so are these important? Surely a failure is a failure, then and now?

What we can do as testers, is question the risk taking, the same as we question everything else. The change in behaviour is information in itself. Risk taking is part of a software testers job. We gamble time against coverage, in an effort to win bugs and new knowledge of the systems we test. When asked to reduce the testing time, maybe highlight how the risk of you missing bugs might change. You can suggest mitigations or conversely inform people of just what won’t be tested. Are they comfortable with that?

When stakeholders are unhappy that you won’t be testing something, that you missed a bug or just found a serious bug at the last minute, you could suggest that the teams risk taking is probably higher than everyone [including them] is comfortable with. If the testing was organ-ised differently, ran for longer, or better resourced maybe the risk of those near misses could be reduced. The bug might be found half way through testing, given more time, new skills or a helping pair of hands.

As a fair witness (http://en.wikipedia.org/wiki/Stranger_in_a_Strange_Land#Fair_Witness) of the events, software problems and risks affecting the project we are a valuable resource for gaining a view of a project’s status and less an unwanted burden on project plans and deadlines.