Testing as War?

29 September 2011 · automation, exploratory testing, security, tactics, warfare

We are fighting an invincible opponent. The legions of bugs in our software far outnumber our attempts to find them all. Even the simplest of software releases, inevitably contains a 5th column of hidden pre-existing bugs or quirks that combined with our changes could strike at any time. The question we need to understand as testers is, how can we win? or at least: not lose this battle?

Military examples and analogies can be useful in software testing, and not just those in reconnaissance (http://en.wikipedia.org/wiki/Special_reconnaissance). For example: the Millennium Challenge (http://en.wikipedia.org/wiki/Millennium_Challenge_2002). This pre-gulf war 2 military exercise pitted two forces against one another, in the middle-east. In summary the modern US military was fighting a rogue element in a smaller country. The vast resources of the western power should of have faced few problems. But in fact the former US general (http://en.wikipedia.org/wiki/Paul_K._Van_Riper) playing the role of the 'Rogue nation' trounced the western forces in a devastating blow that saw several warships sunk.

How did the 'rogue' general do so well? His tactic was to think asymmetrically. Like actual organisations in such situations he used the principles of asymmetric warfare. He played to his strengths rather than to those of his opponent. Rather than using radios, which would be eavesdropped, he used couriers and signal-lights. Rather than try and communicate directly with troops, controlling every move as part of a strict plan, he gave his players autonomy.

We can't usually win move for move with software programmers and bugs. The idea of solving 'testing' by trying to match each feature with a set of pre-defined tests, is like the rogue-general above deciding to build his own carrier fleet. Those features are each complex and capable of working and failing in a myriad of ways. The tester needs to acknowledge his skills and weaknesses, and use tools such as automation to help (http://www.investigatingsoftware.co.uk/2011/08/test-automation-that-helps-guardian.html) where it can.

Once you accept the asymmetric warfare approach, there are tactics that can be employed to help you test. Concepts such as:

Reconnaissance by fire (http://en.wikipedia.org/wiki/Reconnaissance_by_fire) where you investigate multiple possible features, without necessarily having cause to. If you find evidence or anomaly - you can then focus more narrowly.

When might I use this? If you have some time to test, but not enough for a more exhaustive approach, this technique can help catch issues that would be missed if you only focused on the highest risk areas.

Swarming (http://en.wikipedia.org/wiki/Swarming_%28military%29) **or Saturation** (http://en.wikipedia.org/wiki/Saturation_attack) is another, deploy a large number of people to test a given system at the same time. You might also find it useful to run a load-testing tool at the same time, adding to the affect of 'many users'. Another trick is to use a clients existing test automation, to add the effect of more users. The aim here is not to use the test-automation for its intended purpose, but rather merely to simulate load closer too and exceeding expected load. As such the negative aspects of the customers test-automation; high maintenance, brittleness, flakiness and irrelevance etc are less of a concern. We do not care much for the PASS/FAIL results, but rather how the system behaved while receiving 'the attack'.

When might I use this? If you want more coverage of parallel usage scenarios, rather than single user situations. If your customer complains of strange issues that occur in 'live' but don't seem to be present in the quiet times [When they have time to investigate] or are not visible on test systems. By periodically focusing your testers on a specific area, concurrently, you can help counter-act the affects of having to spread a few testers across a large system.