Forensic Software Engineering: Taking the guesswork out of testing

Les Hatton, Kingston University, London

The talk accompanying this paper is about two things. First, it is about the need for objective levels of confidence in the way we do experiments in computing and the way we analyse the results. Second, it is about searching for patterns when you don't really know what you are looking for.

When we do an experiment in computing, we will accumulate data in some form, under some experimental conditions. These data will contain patterns of interest but when we have extracted our patterns by whatever means, it is vital that we determine if the patterns are meaningful or simply random statistical fluctuations. In the parlance of mathematical statistics, we have show that they are significant. Only statistically significant patterns are likely to help us improve what we do. In this area, computing is very naive indeed. If I had a pound for every time I have seen somebody assert that "A is bigger than B, so A is better than B", I would be rich beyond the dreams of avarice. That's not how it works at all. If I toss two coins and get 53 heads with one and 57 heads with the other, we all know perfectly well that this does not mean that the second coin is "better" than the first at producing heads and yet many computer scientists and practitioners are all too willing to accept an equivalent statement, "Technique A found more defects than Technique B, so Technique A is better."

This is simply not good enough. To be of use to the practitioner, we need to be able to say, "Technique A found more defects than Technique B and the possibility of this happening by chance alone is less than 1 in 20, (5% being a normally accepted statistical measure of confidence)." In other words, the practitioner would then be pretty confident that Technique A would be more effective than Technique B and therefore worthy of investing our precious testing time. As it stands however, with most attempts at experimentation in computing, the practitioner has no idea of how reliable the results are, so as a result, we are simply blind. We have been blind for the best part of 50 years but today, systems are immense, with many consumer systems creeping towards millions of lines of code, if not tens of millions. We need determinism and efficiency in our testing methods more than ever before to prevent the kind of expensive failure which has become all too common. Exhaustive testing is simply not an option and on typical systems has been infeasible for many years.

The second subject of this paper is "Data Rummaging", just one of the techniques at hand to the forensic software researcher. Rummaging is a wonderful word. You don't often see it used nowadays but its dictionary meaning of "... throw about, in searching ..." says it all. Too often we see "Data mining" used in computing, but the trouble with "mining" is that it assumes you know what you are looking for, as in mining for gold, or mining for uranium. When you plunge into the bowels of a failed software system or some great steaming pile of data which somebody has laughingly passed off as a database, you hardly ever know what you are looking for at the beginning. On a number of occasions, you might not know what you were looking for at the end either but that's show biz and a sense of humour is a valuable part of the job.

This is particularly apposite of failure data. There is general agreement and has been for hundreds of years that seeking to understand why something has failed casts invaluable light

on what might happen in the future. Like so many things in engineering, the Japanese have a nice compact graphical aphorism for this: "on-ko-chi-shin" which looks far prettier in its Kanji form and translates as "the past holds the answers for the future." Unfortunately, they do not have an expression for the computing equivalent, "the past holds the answers for the future but we are having far too good a time to be bothered to learn, so we will just invent something even more fantastically improbable than what we are doing already and do that instead before anybody catches on that we haven't a clue."

Very little failure data in computing is structured in any useful way because failure is not at the foremost part of our minds when we design and build systems. In spite of my droning on about it for years, (in the presence of some very good company I admit), testing is still the "crumple-zone" between development slippage and immovable delivery deadlines. Systems still fail catastrophically all too often, and these of course are the good ones. The bad ones did the decent thing somewhere along the development line and threw themselves on their own swords. As a result, it would often be more appropriate to call IT an "Incompetence Tax" than "Information Technology".

So if we do manage to get our hands on some failure data, rummaging it has to be. A perfect example of a prodigious, (around 20Mb of text), highly detailed and almost completely unstructured database is the Common Vulnerabilities security database, www.cve.org. It is a magnificent and assiduously maintained repository as it should be for something as important to us as security, but it is less than easy to extract meaningful patterns from it. This is not the fault of Mitre inc, the providers and maintainers of this treasure trove, because after all these years, computer scientists still do not even agree about the form of words we should use for failure. Neither do we have a reliable taxonomy of failure types, so the data almost by definition has to be unstructured. This has a correspondingly negative impact on the development of testing techniques but fortunately, it is still possible with advances in rummaging techniques such as "Chance Discovery" to extract useful failure patterns from it by semantic analysis.

The accompanying talk for this short paper demonstrates the value of failure patterns in shaping the way we think and act in testing and provides simple guidelines and reminders for determining whether a pattern is important or merely coincidental. Failure patterns from a number of commercial case histories will be presented and I hope it inspires others to design further experiments which we may use to improve the knowledge and practice of testing. The talk also presents some recent experiments with sophisticated and openly available data-rummaging techniques which allow useful patterns to be successfully extracted from the Common Vulnerabilities Database and other similar sources. These too I hope will prove useful methods in the production of more reliable systems.