Quality in a Jar

28 December 2010

Its an oft chanted mantra that software quality can be ‘baked in’. Like flour or eggs you just mix in the right ingredients and out comes perfection. If my wife and I were ever to take part in our own bake-off, then you’d soon see that hypothesis undermined. We’d both begin with the same ingredients, oven, spoons etc. My better half would no-doubt deliver another scrumptious banana cake and I - a new and interesting exterior wall sealant [not my intention].

The problems here are manifold, There’s a Reification error, for quality is not a tangible entity (A favourite issue for Michael Bolton (http://www.developsense.com/blog/2010/07/another-silly-quantitative-model/)). There’s the idea that once built software and its use is immutable. If you’ve ever had to use one of those unmaintained applications (typically a time sheet/billing system) thats gone to seed you’ll agree that it hasn’t exactly aged like fine Claret. But what’s most naive is the assumption that the simple use of a tool or ingredient will make some thing better - it’ll make the bugs go away and stay away.

People and consequently software is more complicated than that. A problem with a piece of software can be almost anything from an incorrect conditional operator, to the use of an incorrect language dialect e.g.: I had the pleasure of testing some navigation/directions software, developed in the USA, as it was being ‘tweaked’ to work in the UK. A good example of an issue was an instruction along the lines of: “Take the off ramp, then rotary 1st exit”. To a British English speaker that sounds more like instruction on how to skate-board while donning a wristwatch than how to navigate a junction (sorry ‘intersection’)

The tool, is just that, a tool and is nothing without the artisan’s skill behind it. In testing, it is the testing skill involved that matters over the particular software tool or library installed. To highlight the oddness of the proposition, Transplant the situation to another team member and see how it fits: Does the programmer with the most feature rich-IDE automatically produce the best code? or wider afield, Would you trust a bad doctor with space-age tools more than a skilled and experienced doctor and his judgement as to what tools and instruments to use?

Just adding a test tool, a Jar file or executable, and robotically implementing checks in its prescribed format does not automatically give you good software. Further-more, approaching a testing problem with the tool in-hand, biases you greatly as to what problems you are going to find as well as possibly miss-directing you away from the problems.

> “When all you have is a hammer, all problems start to look like nails.” - Mark Twain

For example over-using Selenium or WATIR for testing a feature rich GUI is a common mistake. Many companies standardise on such a tool for GUI test automation. If your application is heavily GUI driven, then that’s important to bear in mind. Your feature-changes are human user oriented - so testing could well benefit from well errh more ‘human user’ interaction.

The most powerful tools are your eyes, brain and hands. Let these take a look first, get an idea of where to apply the tools and what those tools might be. Don’t expect that just installing that Jar will bring automatic improvements.

PS: You can learn more about those ‘Sun Jars’ here: http://www.comfycamping.co.uk/Gear/Lighting.aspx (http://www.comfycamping.co.uk/Gear/Lighting.aspx)