ID Skeptic

23 January 2011 · bias, breaking things, skeptic, testing

At a client site, a few years ago, I had an interesting discussion with a ‘senior programmer’. Our discussion centred on a configurable home-page. A user could decide what news or other information etc, they wanted to display on their home page. They’d start off by being given a generic page - and the customer could add or remove certain types of content to customise their page. Once they saved the ‘new look’ site, their choices would be saved on the web-server.

The company didn’t want to force people to login, or even make them sign-up for an account. The goal was to keep it simple for the user. But they needed a way to uniquely identify the users, so they used an existing feature of the website. The first time a user came to the site, they were cookied with a ‘unique’ id ‘code’. We could then use this identifier as a key in our database to store the details of what the users had configured their homepage to look like.

The testers reading this will already be dreaming up but what if’s and other questions regarding the plan. When I was told of the plan, it seemed logical and sensible. It has its flaws, and many of these were known and accepted. For example: The users settings would not easily be transferred from one computer to another etc. The cookies containing the ID code were likely to be deleted over time, for various reasons. They were known, but the benefits were considered greater than the known problems.

One thing did stand out when I heard the plan though. As a software tester, I’ve become more sensitive to absolutes; I suspect I notice certain words more than non-testers. When I hear Never, Always, Unique, Every and similar words it’s like a red rag to a bull. (Maybe we should christen them matador-words, designed to engage the critical instincts in a good tester.)

In this instance, I’ve learned that unique is firstly a relative term and secondly rarely tested.

A relative term because many systems I’ve used and tested have had unique identifiers. Upon closer examination, they often end-up being unique to a development environment - but possibly repeated in test and live systems. Fine you think, until you see that developer’s test-orders might not be filtered out from real outgoing orders to suppliers. Or: unique to a country or language, until they attempt to merge databases to one large centralised and multinational system.

Sometimes it’s more fundamental, they just aren’t unique or maybe they are - but will soon ‘run out’. A good example I’ve seen of an ID that wasn’t unique is a timestamp. The idea being that a timestamp [to the millisecond etc] is unlikely to be assigned to multiple users. To be sure, a programmer might even synchronise the handout of the timestamps ensuring one

is always before another. But what happens when you have 2 or even 100 servers? The chances of a clash soon become quite likely.

Another system I investigated had reliably unique identifiers or ‘IDs’, but the system was greedy and ‘grabbed’ many at start up. This combined with a flaky-system requiring many restarts per day - meant that the servers were ‘burning’ through vast amounts of the IDs and it was forecasted we would soon run out. A potentially quick fix - when we could see it coming, but it could of been a site-outing incident had it not been caught.

Another related issue I’ve seen is when the ID was unique, but the code used to look up matches for the ID was not correct, and didn’t treat it as such. The comparison was essentially, if one value contained the other value - return true. This issue didn’t show up at first, but did when one value was ‘6’ and it was compared with e.g.: ‘156899’ the ‘match’ was ‘good’. Unfortunately that code was for restarting production servers and caused approx 80% of the production servers to be restarted at the same time.

In this example the IDs appeared at first glance to be good. They consisted of a mixture of letters and numbers and were over 60 characters long. This means there was a lot of ‘ID-space’ and therefore: very many possible unique IDs. The programmer correctly thought that this number was so huge that the system could hand out new IDs, essentially ‘forever’, without ever duplicating a single one. Any suggestion otherwise was clearly short-sighted, and failing to ‘get the math’.

But actually, the programmer confused how the system ‘should’ work with how it ‘does’ work. What we are testing is the uniqueness itself – that’s part of the system under test. We shouldn’t assume that there might be bugs throughout the application - except in -that- critical but untested feature. When you question the behaviour of a piece of software, why stop when presented with a sacred cow in the form of the word ‘Unique’ or others such as ‘Random’. It would be foolish to depend on the foundation of a unique ID when testing an application built on it, especially when given no evidence of it’s ‘goodness’, other than faith.

What is often confusing to non-testers is that we question such things as ‘Uniqueness’. As discussed above, the system could be capable of generating good ‘unique’ IDs, but it’s another thing to be confident that it is actually doing so. There are many reasons the system you’re working with isn’t getting the unique input it requires. As a tester, questioning these assumptions and highlighting the risks we uncover provides valuable feedback to our customers.