Controlling software development

28 February 2011 · control, illusion, Scientific Method, testing

Do you ever feel like we do all this work and maybe we needn't of bothered? Things might have worked-out without our intervention. Or we are actually worse off, now, after the work? You're not alone. This is a common problem in any role where you need to investigate the effects of changes. What you're feeling is a lack of control.

A control is a view of the world, without your work. It's an alternate view of the world where everything is the same except for your fix/hack/intervention. They behave like 3D TV, they let your mind's-eye 'see' the effects, by making them standout from the background.

They are commonly used in scientific (http://en.wikipedia.org/wiki/Scientific_control) and especially pharmaceutical research studies. They let the researchers know how effective a treatment was, compared with similar patients who received placebo (or older established medicine) pills rather than the new treatment. The researchers can tell whether, for example, a new flu remedy actually helped the patients. Or whether [like the control group] the patients got better/worse at the usual rate, suggesting the medicine had little or no affect.

In testing you probably already use controls, and sometimes without knowing it. For example, if a programmer claims to have fixed a bug - you probably check that the old buggy code before you check the new [allegedly] fixed code (you do don't you?). This lets you see the difference, and discern whether the fix is really -the treatment- the system needed and did it actually fix-the-problem? Controls can even help highlight that which was thought of as a bug, was in fact useful albeit buggy behaviour. For example what if that ugly error in the data-entry application is 'fixed', so that users are no-longer bothered by that error. Great! the new code shows no error message, thats good right? Well maybe not, if I go back to the old code, I might see that the ugly error was stopping users from typing in bad data into the computer. Bad data that might cause other important systems to fail. The real fix to the ugly error might have been to better report the problem to the user, so they can see the data is corrupt and fix it at source.

Often you might find that there is resistance to your wish to examine and test the older pre-fix code. This can be for several reasons, including limited test-system resources or a poorly set up build and deployment system that doesn't let testers deploy old 'builds' of the system.

Performance and load testing is another area whether controls are often needed and used. Though here many projects proceed blind, not using controls or even a baseline. Not knowing how your latest changes affect the system, compared with unchanged code running on the same systems at a similar time. Often teams use a baseline, established long ago. This of course is better than no comparison at all, but makes accurate measurement difficult. Who

knows what other changes have occurred during the time-period since the baseline was taken. Network upgrades, code-library changes, database growth and other peoples changes are just a few possible things that can influence your results. Unless you have results from before and after your changes [and -only- your changes], you are guessing at the effect of your fix.

Being aware of the usefulness of controls is an essential part of software testing. When asked if the new build is better, we can answer “Better than what?” When we suspect the new feature is more smoke and mirrors than fix, we can ensure we have access to the unchanged system for comparison. If you find resistance to your requests for access to controls, well, you have there some interesting information to put in the test report.