Be Lazy, But Not Intellectually Lazy


A while ago, Jeff Atwood wrote about the merits of laziness for successful software developers. Lest this become the mantra of sub-performing developers everywhere, I wanted to follow up with a clarification.

It's fine to be lazy as he describes in the article, just don't be intellectually lazy. What do I mean by this? First and foremost, when you are writing code, make sure you really understand what the code is doing. The classic illustration of mental laziness is encountering an off-by-one error.

When encountering the error, the lazy developer would simply append a " + 1" to the end of the stament, re-run the code, and if it seemed to work, move on. Or they might change a "< x" to "<= x" For simple cases, this may be the correct solution, but the problem arises when the developer doesn't take the time to evaluate why the error was made in the first place. Sometimes, the simple fix only works for a narrow range of inputs and masks a larger error.

This solution is merely one example of a whole class of anti-solutions I call **"Try It and See" solutions**. The developer simply moves code around a bit and crosses his fingers to see if it works.

Off-by-one errors are only the tip of the iceberg. This class of anti-solutions often come up when when a developer is using a framework such as ASP.NET in which he is unfamiliar.

On a recent project, I noticed one of the developers had put nearly all of the page logic within the PreRender override. I asked the developer why he put it there, since the proper place would have been in OnLoad. He replied that OnLoad was too early to run that code because the controls didn't have their settings from the inline control declaration within the aspx file.

Hmm, I'm pretty sure they would be there by then I told him, and he said in his experience, they are not. So I emailed him the order of events within the ASP.NET page lifecycle and pointed out that the method AddParsedSubObject happens after the constructor and way before OnLoad is called.

I believe that he did encounter a weird problem a long time ago with control declarations not filtering through, but rather than dig into the problem and really understand what was going on, he simply moved the code to PreRender, saw that it worked now, and cleared his hands of the problem.

I can understand that on a rush project, there's a temptation to simply try things till they work and then move on, but you will save more time in the long run if you take a break and dig into the problem to get a real clear understanding of what is happening.

Likewise, spend time getting up to speed on the framework you are using. For example, ASP.NET has a usable form validation framework. Learn it. Use it. There's no point in wasting time writing your own framework for validation unless you know the ASP.NET validation framework inside and out and really need to work around its limitations. And if you are going to write your own, consider buying a package first such as Peter Blum's validation package.

So once again, be lazy, but not mentally lazy. Write unit tests up front where they make sense. Learn the framework you are using. Understand the code you are writing or debugging. And in the long run, you'll be making your life (and your coworkers lives) easier. Perhaps that's the true laziness.

Author: Phil Haack
Copied from: http://haacked.com/archive/2005/09/18/10204.aspx