Before you try to change something, make sure you can change nothing

devblogs.microsoft.com/oldnewthing/20240513-00

May 13, 2024

Raymond Chen

Some time ago, I recommended that before you try to do something, you should make sure you can do nothing. There's a variation of this maxim when you are adding code to an existing component rather than writing something from scratch: Before you try to change something, make sure you can change nothing at all.

In other words, take the existing component and run it before making any changes to it at all. Does it work?

Sometimes we'll get questions from developers saying, "All I did was add this line of code, and I'm getting all these build errors. What did I forget? Do I need to add another dependency?" This often triggers a prolonged troubleshooting session trying to see how that line of code could result in those build errors, until somebody asks, "Did it build before you made any changes?" And then we learn, "Oh, I don't know. I never tried."

This step of building the unchanged component (perhaps I should call it "step negative one") makes sure that your development environment is properly set up: Are the build tools installed? Are the *correct* build tools installed? Did you install all the necessary libraries? Maybe the component retrieves a NuGet package from a NuGet feed: Can you authenticate to that feed?

After you build the component, can you deploy and run it? Did you set your test system into an appropriate developer mode so you can install your component onto it? Do the unit tests pass?

After you've gained comfort with a component, you can start skipping these steps, but these are important steps to undertake before writing a single line of code: If you can't get the component to build, deploy, and run as-is, you'll certainly never get it to do those things after you make your changes!

And doing all this on an unchanged project makes it clear that the problems you're encountering are unrelated to any changes you plan on making. Now you know that you need to troubleshoot how you set up your development environment, rather than debugging

your code.

Bonus chatter: If you're particularly unlucky, the problem will be that the project is *already broken* and nobody noticed. But at least you can get the attention of the people with primary responsibility for the project, seeing as you can clearly show that the problem is theirs and not yours.