Code Reuse is not a Design Principle November 26, 2017

Consider the following scenario: You've been given the task of writing a difficult method or a complex SQL query. You estimate that it's going to take several hours and burn a significant number of brain cells. Then, upon further reflection, you realize, Wait! I've got some existing code from another project that does the exact same thing! You proceed to copy that code into your project and it works perfectly. Time saved and brain cells preserved! As any developer can attest, this is one of the best feelings in software development.

Similarly, imagine that you don't have any existing code to solve your problem, but you google the issue and find a perfect code snippet on stackoverflow. In modern software development, this happens all the time and, needless to say, it has radically amped up the efficiency of developers. This could also be considered another species of code reuse; the only difference being the use of someone else's code rather than your own.

As these two examples illustrate, code reuse is great. It can be a significant time saver. It also allows developers to solve problems way easier than if they had to go it alone. Having said that, there is a dark side of this force. At a high level, I call this potential abyss "designing for code reuse." What do I mean? Let me illustrate with an example. I was on a project recently where the database design was absolutely dreadful (missing relationships, extraneous fields, de-normalized in bizarre ways, etc.). It was so bad, in fact, that they ran SQL jobs at night in order to identify orphaned records and other bad data issues. My initial reaction to this mess was that the database designer must have been incompetent. Sometime later, one of the developers revealed that the reason the database design was so bad was because it had been designed so that it could "possibly" be reused to support a different application. Was it ever used by this other application? Of course not! But even if it had been, designing with code reuse as a primary goal is almost always a bad idea.

Whenever I hear an architect/developer say that they want to design a project with code reuse in mind, red flags and alarm bells go off in my mind. The reason, of course, is that this implies that compromises will be made in order to keep the application "flexible" enough to handle the requirements of another application, or applications. These "compromises" are almost always ugly, and they stand out as "WTF was this person thinking" design elements.

My strategy is almost always to optimize the app that I'm working on, and if I can reuse some of the code later, great. Often what happens is: I'll start a new project with some code from a previous one, but the code will deviate quickly from the original. The reason is because the new code is rewritten with greater experience – experience in general and experience in particular with the old code (i.e. knowing its strengths, weaknesses, and where it can be improved). Additionally, if I ever need to change shared code such that it can no longer be shared with another application in order to optimize each individual project, I will do so (usually) without hesitation. There's nothing worse than being boxed in by the requirements of another project, and I like optimized, highly-performant code (Seems obvious, but I'm consistently amazed by the number of people I’ve worked with who care very little about optimized code).

After over twenty years of experience in software development, the bottom line is this: The costs of designing for code reuse far outweigh the (potential) benefits.