Debuggability is a Design Principle August 18, 2018

A few months ago, I wrote a blog about the pitfalls of designing software explicitly so that it can be reused by other applications (See "Code Reuse is not a Design Principle"). Today, I'm going to address an often overlooked principle of good software design; a principle that should be taken into account before making any significant architectural decisions. First, let's add some context this discussion.

Over the past twenty plus years, I've been hired many times to assist with applications that needed some TLC. Often, these apps are older and they're riddled with hacks, design shortcuts, over-engineering, etc. Often, these issues are unavoidable in apps that have been around a long time because, despite people's best intentions, code bases tend to spaghettify over time. In any case, before I sign any contract to assist with these types of applications, I always ask for a "code review", and in particular, I want to see how the current team debugs the application. If an app is readily debuggable, I usually don't have a problem coming in. On the other hand, if the application's architects didn't take debugging into account at design time, I'm much more reluctant to come in because I have enough headaches already. Let's take a look at a couple of examples, which will illustrate what I'm talking about.

The worst code base I ever saw was about 10 years ago. It wasn't the result of hacks over time, but a case of gross over-engineering by OO hipsters. Allow me to digress for a moment. There was a seminal blog written in 2001 by Joel Spolsky about the over-engineering of software and the type of developers who perpetrate this crime (See "Don't Let Architecture Astronauts Scare You"). Joel refers to these people as Architecture Astronauts; I generally call them OO hipsters, for two reasons: 1) over-engineered solutions usually involve taking OOP to some silly level, grossly out of proportion to the requirements of the app, and 2) "hipster" is more derogatory, and frankly, a more accurate term (I accept the Urban Dictionary definition of this term.)

Returning to this horrible code base that I referred to above, the running joke was that it took 10,000 lines of code to populate a drop down list. This was an exaggeration, of course, but it was probably close to 500. How is that possible? This app was abstracted out to ridiculous levels: object relational mapping, DI containers, factories, interfaces, and inheritance all involved in the population of a drop down list. Needless to say, it was very difficult to debug, and it took a very long time to step through all of the levels of abstraction. After I'd been on the job a few days, I asked one of the veterans what he did when the app didn't work. He replied: "Cry." I cried a lot on that contract. Ultimately, the performance of the app was so bad that I got the green light to bypass all of the trash and write simple, high-performant, easily-debuggable code.

I was on another contract recently where one of my tasks was to evaluate third-party accounting packages that would be integrated with some proprietary software that I had written. One of the primary questions I had for the vendors was how easy would it be for me to debug if something went wrong. Often, this isn't easy when you're dealing with third-party

software, but in this case one of the vendors offered direct access to their SQL tables. This was a major selling point and ultimately tipped the scales in their favor. No matter what happens in the app, I can see exactly what the data is at all times. Needless to say, this has made debugging much easier.

More generally, whenever I'm considering adding a third-party library or component, I always drill down on the debuggability of it. Will I get descriptive error messages? Can I easily step through the code? Is it easy to isolate where a problem is occurring? Etc. If a library doesn't make these things easy and intuitive, I won't waste any time with it.

At the end of the day, most of a developer's time is spent debugging, even in green fields. Doesn't it make sense then to accept that debuggability is an extremely important design principle that should be taken into account up front?