Saturday, March 18, 2017

Why Your C++ Should Be Simple

I have a lot of strong (meaning "perhaps stupid") opinions about C++ and how to use it (or not use it), and I've been meaning to write posts on the thinking behind these stupid^H^H^H^H^H^Hstrong opinions, particularly where they go against the "conventional wisdom" of the modern C++ community.

This one is sort of an over-arching goal:

> Write C++ that is less sophisticated than you are capable of writing.

This idea is a rip off^H^H^H^H^H^Hriff on a Brian Kernighan quote that I think is perhaps the best advice about programming you'll ever hear:

> Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?
>
> -- Brian Kernighan, the Elements of Programming Style

First, that's not wrong, that's dead on. So for debugging alone, don't write the most complicated C++ you can. C++ is an extremely complex language in its spec and if you use it to its full capability, you can write something that is completely un-debuggable.

Limited Brain Bandwidth

Here's my take on the idea. You're welcome to dismiss this as the mad ramblings of a father who is over 40 and literally cannot remember to finish steeping my tea because it's been an entire four minutes since I left it in the kitchen.

There's a limit to your mental bandwidth. That mental concentration, focus, short term memory, and whatever else the squishy gray GPU between our ears does can be spent on a few different things:

- Figuring out how to write code.
- Figuring out how to make code faster.
- Figuring out how to solve non-trivial algorthm problems.
- Figuring out how to solve business problems and design the architecture of a large production system.
- Finding and fixing bugs.
- Multi-tasking.
- Looking at cats on the internet.
- Swearing at your customers.
- Trying to maintain focus while your 2 year old repeatedly smashes your keyboard with your mouse.
- Figuring out the fastest path through a series of coding tasks.

I'm sure there's more, but you see where I'm going with this. You have a lot of things you can use your brain for, and some are more useful for getting your job done than others. You have to decide where to spend your brain budget, and you can't spend it on everything. (If you feel like you're not tight on brain budget, it means you're not working on a problem that's hard enough yet! You can always make the problem harder by shortening the time frame. Shipping things sooner is pretty much always a win.)

My view is that "being clever with C++" isn't a fantastic investment. Using the most bleeding edge C++ doesn't have enough benefit in other areas (code is faster, code is more maintainable, code is easier to debug, code is easier to understand) to justify burning brain power on it. In some cases, it actually moves you in the wrong direction.

Just because C++ lets you do it doesn't mean you have to or that it's even a good idea. Here are some examples where I consider a C++ feature to add complexity and not provide returns:

- Clever operator overloading. Your coworkers will thank you if you give your functions good names and don't make operator+ turn on the coffee maker. You could skip operator overloading entirely and your code will be fine. (I'm okay with a small amount of overloading for a few obvious cases: dereference on smart pointers and math on math types).
- Template meta-programming. I make myself watch talks on this from cppcon and what I see is the smartest C++ programmers in the world spending huge amounts of brain power to accomplish really trivial tasks (writing a program to make a program) in a way that is almost completely impossible to understand. You don't have to use the compiler to do your meta-program generation.
- Heavy Duty Templating. This is a matter of degree - simple use of templates is fantastic and I use them in my code all the time. But at some point, as you add layers, you go past an inflection point and the cure starts to hurt more than the disease. This one is a little bit like smut: I don't have a great definition for when you've jumped the shark with your templates, but I know it when I see it. A good rule of thumb: if templates are making it harder to debug, don't add more. (Debuggers are easily good enough to debug the STL, generic algorithms, traits...you don't have to drop that stuff. It's not 1998 anymore!)
- Overly Complicated Class Hierarchies. This is also a matter of degree, but at some point, it's okay for your class hierarchy to be less pure, clean and perfect if it's simpler and creates less chaos in the rest of the program. For example, our UI framework has one view type - parents, children, leaf nodes, roots, none of these are specialized by type. I've used a lot of other frameworks, and my finding is that clever "factoring out" of aspects of a view hierarchy does nothing to make the code better, but it creates issues you have to work around. Just keep it simple.

I could go on, but you get the idea...I'm one of those old bastards who thinks it's okay to just use C++ as "a nice version of C". Because it is! In my time here in the matrix, I have found that I have written past code that was too complex, language wise, and too simple. Here's the difference:

- For the code that was too simple (I wrote a C struct when I needed a class), the upgrade is simple, easy to write, easy to understand, doesn't produce a lot of bugs, and the compiler helps me.
- For code that was too complex (I wrote something worthy of Boost when I should have made POD), ripping out that complexity is time consuming and goes slowly.

So why not err on the side of simplicity? Here are some other things you could do with your brain instead of writing C++ for the C++ elite:

- Watch John Oliver on Youtube.
- Put debugging facilities into your sub-system for more efficient debugging.
- Put performance analysis data collection into your sub-system so you can tune it.
- Document your sub-system so your coworkers don't poison your coffee.
- Get the code to high quality faster and go on to the next thing!

If you are a professional software engineer, you are paid to solve people's business problems, and you use code to do that. Every Gang of Four pattern you use that doesn't do that is wasted brain cells.

Labels: c++, Quotes, Rants

No comments:

Post a Comment

Links to this post

Create a Link

Home Older Post

Subscribe to: Post Comments (Atom)