title: **Structuring code**
num_comments: 9
num_up_votes: 16
upvote_ratio: 0.88

Post Text

Hi, although I'm a Software Dev for 10+ years (and even coded before that) to me it seems impossible to write decently structured code in a bigger scale. Sure for small projects it works ok, but for bigger ones I simply loose scope and/or get so lost in the details, that I forget to think about other parts, which need to also interact with the parts I'm focused on. In the end it either gets a mess, that I created stuff which sounded like a good idea, but isn't used anywhere and it gets messier because I somehow have to attach the parts I missed out on. (plus: I can't bring myself to refactor, as I feel like I have to do it 100 times to get a halfway decent result) As an example: when I check out the code bases of third-party libs or full blown frameworks, my jaw drops, as I can't imagine to be ever able to create such well-written and well-structured code. Overall I simply feel stuck, even worse, more like a complete failure, that I took the absolutely wrong career path (although coding was "my thing" since my early teens). Do you know that feeling? And by chance, can you share any advice or technique which can help to improve? Does medication help?

Comments

Commenter_2
ID: REDACTED! ~(o.o)~ <3, Upvotes: 15
Refactoring is an important step in adding a new feature.

The existing code was created without knowledge or assumption of the new feature you want to add. If there's a logical place for the new feature to slip into, that's good! Often times there's not. Figuring out how to rearrange the existing code to create that space is the first step in adding a new feature.

Refactoring is a skill that takes some practice, so don't get disheartened if you feel like it takes too long or too many steps. As long as the refactor is an improvement, that's good! Be wary of that classic ADHD perfectionism trap, it doesn't need to be perfect, just better.

> ###### Commenter_2
> ID: REDACTED! ~(o.o)~ <3, Upvotes: 9
> To add to this, your comment about creating things that aren't used is a red flag. You should only write code as you need it. It's too easy to add unneeded complexity because you think you're thinking 10 steps ahead, but by the time you get there you realise you made a bunch of assumptions that just don't hold up. Write the smallest simplest thing that works right now, and delay adding anything until you need it.
>
> > ###### OP
> > ID: REDACTED! ~(o.o)~ <3, Upvotes: 1
> > I try to do that, but the thing is, that to me in some cases it is very hard to tell what is necessary and what not, especially when I build a foundation for something.
> >
> > And when I think I got it right, I usually notice that it's missing something actually important and adding it is a problem, as it actually is quite a fundamental feature (that I simply didn't think about at the time). Compare this to building a house: I would build a foundation at floor level and later notice that I actually need a cellar too.
> >
> > > ###### Commenter_2

ID: REDACTED! ~(o.o)~ <3, Upvotes: 2
Hmm, I think you may be doing too much upfront building the foundation , as you put it. Your aim should be to build the smallest possible useful thing first. Programming isn't like building a house, it's like growing a garden. It's going to evolve and change shape over time as you add new features, it's dynamic. Starting by building things you think you might need later is a trap, because as you said you figure out that you've made a miscalculation and built something mismatched with what you actually need.

OP
ID: REDACTED! ~(o.o)~ <3, Upvotes: 1
Thanks! But to me it feels like that there is much more refactoring necessary, than just extracting a bunch of new methods. It feels to a degree like writing the whole application again and again and again, until it is at a stage, that I would call it "well structured", because with the refactoring any test created needs to be rewritten as well.

I really can’t imagine that this is actually best practice approach, because it will take ages doing these rewrites, without really making progress. And then the question is also: at what stage is it "good enough"?

To make things worse: often I don’t even have a good idea about how to refactor something. I mean, yes, I know that I could extract this and that, but then again I get lost in these details and miss that this has an impact in other places.

Commenter_2
ID: REDACTED! ~(o.o)~ <3, Upvotes: 1
It's hard to provide specific advice here, because we're talking pretty general. But let me explain how I think refactoring fits into software development.

Ultimately the best programme is the simplest one that fills the requirements. That's the perfect ideal, not necessarily a pragmatic or achievable goal, but that's the one to keep in mind. The less code there is, and the less complex that code is, the less room for bugs and errors.

Every time you add a feature, you add complexity. If the new feature fits well into the current design of the system, it's just a bit of complexity. If there's a mismatch between how the system is currently designed and the new feature you want to add, it can be a significant piece of complexity. Maybe there's a special case if statement somewhere far away from where it logically should be, or maybe there's some hacks to work around some assumptions that don't hold true anymore.

Your responsibility as an engineer when growing a system is to manage the level of complexity. More complexity means more places for bugs to pop up in, the code starts to become more and more difficult to follow, and it becomes more and more difficult to add new features. Refactoring is the tool you have to reduce complexity - the goal is to reduce the complexity of the code, without removing any necessary features.

Whenever you add a new feature, you need to ask yourself how much complexity you're adding, and if that's a level you are happy with. If the system is starting to look too complex, you need to step back and ask how you can reduce complexity before adding anything new. How can you reorganise code so there's a more obvious place to put your new feature? Are there repeated patterns in the code that can be structured into something more formal and reusable?

Refactoring is not a distraction from building things, it's an integral part of it. If you let

> complexity build, you will hit a wall where it can feel impossible to add anything without it all coming apart. The longer you let complexity build, the harder it will be to untangle things.

Commenter_4
ID: REDACTED! ~(o.o)~ <3, Upvotes: 1
I would go out and grab a copy of "Zen and the Art of Motorcycle Maintenance" or the Audiobook, infact its probably time for me to give it a read again. It has been about 10 years or so.

Anyway, what I learned from that book is that appreciating Quality comes from experiencing Quality.

This is something that I try to explain to my team often. Suuuuure, its easy to add a method to a data class now; because that is where your cursor happens to be at that time. But think about the future _YOU_. Will the future you remember that this is where that method exists? Or should it be part of some business logic in a specific namespace that is easy to find?

I think that is the unique influence that is possible with the ADHD mind. We can be both be too time-blind to care about the future ramifications about a quick-n-dirty hacky effort; AND at the same time incredibly artful with how we fold code and ideas into a result.

There is a weird balance between the fun of an idea, and the practicality of living with it.

In our team, we have a support tech that can modify code as needed. Having code that is structured a certain way helps him heaps since he wrote zero-lines of code for it.

We build our projects with a similar structure so every time a team member picks up a project it should feel familiar. Like if you had gone back to an old apartment/house you had when you were young. Its the same shell, but of course different people are living in it now.

Those public libraries, like say https://github.com/nozzlegear/ShopifySharp are built against the demands of the users. Which are all developers too! They get thousands of hours per year of review/feedback/design.

Where was we may just look at something for 100 hours. Its just not the same sort of polish that we can do.

So for now, we keep ourselves humble, we are writing code not just for the computers, but for people as well. That "person" has a name, and wants to help you fix the bug you wrote a week ago, so help him.

Commenter_5
ID: REDACTED! ~(o.o)~ <3, Upvotes: 1
Break down the big project into multiple tiny projects. Dont sit in single giant code base it will load in your head too.

Unknown_User
ID: REDACTED! ~(o.o)~ <3, Upvotes: 1
What are your criteria for considering code to be "well" structured?