The Moment I let go of Clean Code

I was thrown in a project, that was a big mess. They told me the guy who built it was a pro, but it...

3 min. read · View original

I was thrown in a project, that was a big mess. They told me the guy who built it was a pro, but it was created under extreme time pressure that's why it's just a bit messy.

I didn't believe that.

I thought: "the guy who build it was an idiot". Functions were all over the place, there was no logical grouping of code and naming was terrible. Yes, naming is hard, but these function names not even came close to what the code was doing...

I thought - clean code is important. I liked the way clean code looked. Small and concise functions. Short names. Comments above functions that had a box of dashes around them to make them more visible. Beautiful.

Different story:

A while ago I have been working with a clean coder on a different project.

He kept pushing back the releases. "I need to refactor that first", "Before the next ticket I need to do 1 week of cleanup", "man, I just spend the full day splitting a 5000 lines file"...

He was very serious about this. And he was doing decent work, but as humans do, he also made mistakes, and his 5000 line refactorings lead to new bugs that needed to be fixed and required another refactoring.

Every time I mention clean code in a blog post i get a few comments why it's so important and how everyone should adhere to the holy bible of coding guidelines.

"Take your time to clean up.", "Leave the code in better shape than you found it.", they said.

It reminds me of the whole inbox zero debate. Imagine a mailbox, that is empty. Beautiful. It takes some housekeeping effort, but man look at this beautiful emptiness.

After all, isn't this the same as keeping your house clean? No one wants trash to pile up in the house. Keep the house clean - keep the desk clean - keep the inbox clean - write clean code?

After taking a vacation my inbox was flooded again. You have to have a close eye on your inbox every day and you better react quickly if you don't want that mf to fill up again. It's a little bit of housekeeping, but doing that shit everyday piles up and then it becomes a lot of housekeeping. And it's annoying, and it stressed me out having to keep the inbox clean.

Tim Ferris' 4 hour work week got me back on my feet. One core idea from the book: We are wasting 80% of our time with nonsense (Paretos law). Emptying your inbox is procrastination. Writing dashes around code comments is procrastination. Every second you deal with a spam email is a waste. It is easy and everyone can do it. There is this quote from the book which is something like: "Am I inventing things to do in order to avoid what's important?".

At some point I just had too much on my plate to deal with all that crap. And I gave up and let my inbox fill. It itches, but I felt relieved at the same time. I let go. I just let go. Mails kept coming in and I did not care anymore.

And guess what - you get used to it.

And you get good at search.

And filtering.

And skimming headlines fast.

And reading ugly code.

And navigating messy codebases.

And every now and then you miss something.

And you need to accept that.

And then a reminder comes in and then you deal with it.

The first version of your code needs to be messy. Then, after using it for a few weeks or months, you realize the flaws. And when you need to extend functionality, you fix it. You fix what's important. If you never touch it again, you leave it messy.

I'm not saying: „be an idiot to your coworkers". Don't produce something messy for the sake of being messy. Just stop painting and focus on what's important.

The difference between a messy house and a messy codebase is, that in coding we have the tools to navigating it.

Don't try to refactor a 5000 line file in one go. It takes forever and you will make mistakes. It's a form of procrastination.

Fix naming that's off when you work with the function. Rewrite a function if you fix a bug with it.

Fix what causes headaches.

Tiny improvements - smaller commits.

Leave the code better than you found it.