DOI:10.1145/3360907 http://cacm.acm.org/blogs/blog-cacm

The Benefits of Indolence

Yegor Bugayenko explains his realization that software developers should go neither above nor beyond.

Yegor Bugayenko
Lazy Developers Are the Best Developers
http://bit.ly/2lEC9KE
July 15, 2019

We are taught from a young age that the hardest workers enjoy the most success. Hard work pays off, or so we are told. But "hard work" can be a bit problematic for software developers, because it often means going well above and beyond the original scope of the project.

This is especially true when it comes to understanding legacy code. When you deal with legacy code, you often find yourself having to engage in so-called "deep thinking." You are expected to understand large problem scopes before you even begin trying to fix the small bugs. For a long time, this stressed me out. Then I got an idea: be lazy.

At my company, Zerocracy, we practice a #NoAltruism policy. We, quite literally, think only about ourselves and our personal profit. This might sound a bit harsh. Isn't it better to play nice and try to appease your clients? In an ideal world, maybe. But here's what we have learned about clients: they also practice #NoAltruism.

Clients want to keep costs low, and if they can, they will pass costs onto outside companies. That's why we decided to "get lazy" and only do what we are paid to do. We won't go out of our way to improve a project, refactor, or fix code unless we are getting paid for it.

And when we find ourselves with a task in front of us and we don't understand how to solve it, we usually don't blame ourselves. This is especially true if the problem has something to do with legacy code. See, here's the thing: we weren't paid to understand the legacy code. We were paid to add a feature, solve a bug, or whatever.

Suddenly becoming experts in a project's legacy code would be outside the scope of our work, and since we're lazy, we're not going to venture outside of our assignment unless we're paid to do so. A project shouldn't expect you to be intelligent or tech-savvy, as far as the legacy code is concerned. Instead, you need to focus on closing tickets.

It's not your fault if the code is a complete mess, or the bug is serious, or you can't estimate how much time it will take to understand the legacy code, let alone how to fix the bug. So whose fault is it? The first guilty party is the code itself. And the clients overseeing the code are also at fault.

Once you accept that, you can put together a basic report by creating new tickets. This report could be lazy-simple:

- There is no documentation for Class Y, can't figure out how it works.
- Library Z is in use but why aren't you using library B?
- This algorithm is a complex mess, can you explain what it does?
- The class naming rules are incoherent, can you provide documentation?

Suddenly, your initial "report" is instead a list of questions. You can't provide the answers because you don't honestly know them and you are too lazy to figure it out. Answering these questions falls outside of the scope of work you were hired for, so it is reasonable to expect the client to provide documentation.

Now, you might have noticed a common thread in the questions here. I didn't ask for help. I didn't ask someone to create something for me. Programmers will often reach out for help, saying something like "which library should I use for this task?"

Here's the thing: your clients aren't hiring you so they can do your work for you. They aren't hiring you so they can be your teacher, either. They don't really want to explain anything to you. For them, it's money and time they would rather not spend.

So your goal, then, is to get your clients to fix the code base so that the code itself becomes more obvious and easier to read. This will help not only you, but everyone else. As such, focus on asking for documentation and code source fixes.

Okay, so you've got the tickets out and you've asked the client to fix their source code and address other problems. So what now? Sit back and relax! You wait for the tickets to be resolved and don't sweat who is resolving the issues; that's not really our business.

Now, your employer may decide to kick the problem back to you, asking you to solve it on your own. That's fine, so long as you're getting paid for it and the employer expands the scope of your work. Instead of fixing bugs, you're now documenting some functionality or refactoring this and that.

As you create tickets and blame everyone else around you, you will continue to create smaller and smaller scopes. Eventually, you may find that the tickets can be fixed in a half hour or less. And keep in mind, when I say "blaming everyone else," that doesn't mean shouting at other people. It simply means not beating yourself up for problems you didn't create, and shifting responsibility for poorly written code to the original source.

Being lazy can take a lot of effort (seriously). We are programmed not to be lazy. Some people will resist the call. They might feel ashamed (stop it!). They want to be perfectionists (only perfect what you're paid to!). Or maybe you lack the passion needed to be lazy (get a new job!).

"It's not your fault if the code is a complete mess, or the bug is serious, or you can't estimate how much time it will take to understand the legacy code, let alone how to fix the bug."

Comments

This is unacceptable practice from ACM's professional ethics guidelines. Zerocracy promotes no altruism and no help. This practice violates the core mission of ACM as an organization, which is "Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing." I request ACM to retract this article. Computing professionals have the obligation to behave in an altruistic manner and help each other for both advancement of business productivity, human well-being, and advancement of computing systems. Zerocracy is a disgraceful movement for computing profession.

—Mehmet Suzen

Mehmet, can you please elaborate on how exactly "contribute to society" leads to the conclusion that we are obliged to behave in an altruistic manner?

—Yegor Bugayenko

I think this policy is created to end the abuse on the client's behalf. #NoAltruism does not mean that in Zerocracy people would create software to support terrorism. Engineering is not altruistic, is precise. The Zerocracy policies are meant to create an efficient culture, not people without values. I think Mehmet misunderstood what #NoAltruism means.

—Eduardo Portal Geroy

"Computing professionals have the obligation to behave in altruistic manner and help each other for both advancement of business productivity, human well-being, and advancement of computing systems." As much as they have an obligation to not waste their time for free, increasing the engineering level in the company, helping others do their job, and saving time to help others and contribute to society in their free time, doing really altruistic things, not what you are talking about.

—Nikita Puzankov

"Computing professionals have the obligation to behave in an altruistic manner and help each other."

There's a difference between purposeful altruism as a means to improve the system, and blind altruism as a fanatic ideology. The thing we need to keep in mind is, the human psychology is never without its flaws, no matter how hardcore a saint you would be trying to play here. I myself have seen numerous examples of a biased altruist doing much more harm than a selfish but rational person in a similar situation.

Zerocracy is about regulating those psychological flaws, not trying to abolish them, which would most certainly end in (yet another) wasted effort. Being truthful with oneself, first and foremost, is the key in building all sorts of constructive professional relationships. Ignorance of that is bound to amplify guilt and fear in performers many times in the end, which might be appealing to certain moral fundamentalists who believe a scared programmer's guilt complex is like some sort of a virtue. The truth is, it just doesn't work out like that.

—Ilyas Gasanov

This occurs whether one is a consultant or contractor, or a salaried employee of the organization that owns the software. I have been both.

Even within the organization that owns the software, the deep thinking required to document otherwise undocumented systems or to fix underlying design problems is discouraged, and the attitude of "fix the immediate problem" prevails. This causes the organization's maintenance costs to increase steadily over time as technical debt piles up unaddressed, deeper and deeper.

This works similarly to the principle of conservation of energy, which pops up in infinitely varied guises whenever one attempts to create a perpetual motion machine: it is always thus regardless of which trendy or modern "methodology" is used in an attempt to manage the problem solved without doing the actual, necessary work.

In the end, one is doing one's client or one's employer a disservice by not warning them that a failure to solve the deeper problems will cost far more in the long run than any immediate savings they will realize by ignoring those problems for the present.

—Robert Watkins

Yegor Bugayenko is founder and CEO of software engineering and management platform Zerocracy.