On Tech Debt: My Rust Library is now a CDO

written on Tuesday, March 26, 2024

You're probably familiar with tech debt. There is a joke that if there is tech debt, surely there must be derivatives to work with that debt? I'm happy to say that the Rust ecosystem has created an environment where it looks like one solution for tech debt is collateralization.

Here is how this miracle works. Say you have a library stuff which depends on some other library learned-rust-this-way. The author of *learned-rust-this-way* at one point lost interest in this thing and issues keep piling up. Some of those issues are feature requests, others are legitimate bugs. However you as the person that wrote *stuff* never ran into any of those problems. Yet it's hard to argue that *learned-rust-this-way* isn't tech debt. It's one that does not bother you all that much, but it's debt nonetheless.

At one point someone else figures out that *learned-rust-this-way* is debt. One of the ways in which this happens is because the name is great. Clearly that's not the only person that learned Rust this way and someone else also wants that name. Except the original author is unreachable. So now there is one more reason for that package to get added to the RUSTSEC database and all the sudden all hell breaks lose. Within minutes CI will start failing for a lot of people that directly or indirectly use *learned-rust-this-way* notifying them that something happened. That's because *RUSTSEC* is basically a rating agency and they decided that your debt is now junk.

What happens next? As the maintainer of *stuff* your users all the sudden start calling you out for using *learned-rust-this-way* and you suffer. Stress levels increase. You gotta unload that shit. Why? Not because it does not work for you, but someone called you out of that debt. If we really want to stress the financial terms this is your margin call. Your users demand action to deal with your debt.

So what can you do? One option is to move to alternatives (unload the debt). In this particular case for whatever reason all the alternatives to *learned-rust-this-way* are not looking very appealing either. One is a fork of that thing which also only has a single maintainer, but all the sudden pulls in 3 more dependencies, one of which already have a "B-" rating. Another option in the ecosystem just decided to default before they are called out.

Remember you never touched *learned-rust-this-way* actively. It worked for you in the unmaintained way of the last four years. If you now fork that library (and name it *learned-rust-this-way-and-its-okay*) you are now subject to the same demands. Forking that library is putting cash on the pile of debt. Except if you don't act on the bug reports there, you will eventually be called out like *learned-rust-this-way* was. So while that might buy you time, it does not really solve the issue.

However here is what actually does work: you just merge that code into your own library. Now that junk tech debt is suddenly rated “AAA”. For as long as you never touch that code any more, you never reveal to anyone that you did that, and you just keep maintaining your library like you did before, the world keeps spinning on.

So as of today: I collateralized *yaml-rust* by vendoring it in *insta*. It's now an amalgamation of insta code and yaml-rust. And by doing so, I successfully upgraded this junk tech debt to a perfect AAA.

Who won? I think nobody really.

As for the title: a CDO is a financial instrument that became pretty infamous during the financial crisis of 2007. An entertaining explanation of that can be found in “The Big Short”.

This entry was tagged rust and thoughts