⚠ Cons

Y **Hacker News** new | threads | past | comments | ask | show | jobs | submit sktrdie (2416) | logout

* Depending Less on Structure (lmatteis.github.io)
62 points by sktrdie 21 days ago | hide | past | web | favorite | 51 comments

▲ hacker_9 21 days ago [-]

1. Example is too simple, functions could modify anything in a model depending on context.

2. Good luck trying to optimise this (just merging all behaviours together at the end does not equal optimisation).

3. You're just pushing the problem elsewhere - now I need to understand all behaviours from a black box, then wrap them, all while dealing with all the other wrappers people have already put in place. Imagine dealing with that complexity.

Y **Hacker News** new | threads | past | comments | ask | show | jobs | submit

* B-threads: programming in a way that allows for easier changes (medium.com)
167 points by sktrdie 6 months ago | hide | past | web | favorite | 92 comments

> However I think that this approach comes with some cost. For instance if (isMultipleOfThree(x) && endsWithDigitFive(x)) is very easy to understand, but isMultipleOfThree(x) and endsWithDigitFive(x) in different modules is a lot harder. Personally I see this more as a trade-off: trading control flow for extensibility.

I think this is a massive understatement. Imagine replacing most of the ifs in your program with new top-level constructs that run in parallel and exchange events. The maintenance overhead of this idea seems astronomical, for relatively little benefit. I would be terrified to add new modules to such a beast, knowing that they could affect the behavior of any other module, in complicated cascades.

https://news.ycombinator.com/item?id=22033987
https://news.ycombinator.com/item?id=20556217

I'm going to make a conscious effort not to come off sounding like an asshole, but please excuse me if I slip up. I find many of the ideas in post to be fundamentally at odds with the direction that "good software development" should be travelling. The core of my feeling is best captured by the following quote from the post:

>As a system grows in complexity we don't necessarily care about how old b-threads have been written, hence we don't care about maintaining them.

This post is essentially formalizing the process of creating a Big Ball of Mud[0] that is so complex and convoluted that it is impossible to understand. The motivation for formalizing this process seems sane and with good intentions: to add functionality quickly to code you don't really understand. Normally, doing something like is considered cutting corners and incurring explicit technical debt, and must be used sparingly and responsibly. However, the process of "append-only development" is embracing the corner cutting and technical debt as a legitimate development process. I can't get on board with this.

To be more specific, with an example (and maybe I am wrong in understanding the post, this would be the time to point that out to me), let's suppose you have a massive complex software system that was built over the years with this "append-only" style of development. One day you find a nasty bug in one of the lower layers, and to correct it, you have to change some functionality, which moves/removes some events that subsequent layers are depending on for their own functionality. Suddenly, you are faced with rewriting all of those layers in order to adapt to your bugfix. What you're left with is a nightmare of changes that disturb many layers of functionality, because they're all based on this append-only diffing concept: the next layer is dependent on the functionality of the previous layer.

This is what programming APIs are for: to change functionality in lower layers with minimal influence subsequent layers. This post and process seems to be imagining a world without APIs.

0. http://www.laputan.org/mud/

Nothing about this approach seems easier to reason about nor maintain over the long term. Transactional integrity, migrating data structures, and modifying business logic/control flow are key building blocks required for many production applications, certainly line of business ones.

I'm reminded of the golden hammer fascination with event sourcing, and I continue to see event sourcing applied inappropriately. Is it useful for some specialized applications? Yes, and it's a great tool there! But should it be the default tool you reach for to solve most problems over a relational DB and your most trusted, high level, stable programming language with a mature library ecosystem?

No to that question, and no to this.

However I think that this approach comes with some cost. For instance `if (isMultipleOfThree(x) && endsWithDigitFive(x))` is very easy to understand, but `isMultipleOfThree(x)` and `endsWithDigitFive(x)` in different modules is a lot harder. Personally I see this more as a trade-off: **trading control flow for extensibility**.

Trade off spectrum