Thread by @GeePawHill: "How I Work: (Just Programming Mix) When I'm programming, I am centered broadly on the cyclical application of small textual changes, each on [...]"

How I Work: (Just Programming Mix)

When I'm programming, I am centered broadly on the cyclical application of small textual changes, each one producing value I then harvest to identify, enable, or energize the next.

Before we dig in: This is how *I* roll code. It is not a prescription of any kind for you. I have a prescription for you, and I can offer it, but it has almost nothing to do with what you'll read here. These are details. The prescriptions are at a much higher level.

You do you.

A couple of words about environment. I roll code using an IDE almost exclusively. I use Intellij's family these days. I've tons of experience w/Visual Studio & Eclipse. I've used a number of others at different times, and gradually set them aside when I found ones I liked more.

I use an IDE because I like my tools to know as much and sometimes even more than I do about my language, my libraries, and my project.

I've found them quite useful to me in dynamically typed languages and invaluable to me in static ones.

Left to my own devices, I use git for my source vault. (Like any long-time geek, I've used'em all.) I use git *very* simply. I avoid cherry-picking changes. I avoid manual merges. I live branchlessly, pulling from head, pushing to head, testing at head.

90% of my git usage is contained in just three commands: pull -r, commit -m, and push. I use the root dir for all of that. When I *do* get stuck with a manual merge of any kind or size, I normally just emit obscenities, throw away my change, and do it again.

Meta 1: as a long time geek, I read maybe 25 or so programming languages. I can write 8, am fluent in five, and am expert in 3: Forth, C/C++, Java/Kotlin. (A little early to claim Kotlin "expert", but the fallback to Java helps.) My dynamics are Python & Ruby.

Meta 2: I have written every kind of software professionally with one exception, that of professional video game development. I've written hard realtime, database-to-web, desktop apps, analytics, process control, device drivers, UI's, spreadsheets, word processors.

Meta 3: Collaboration is central to professional software development, but I'm going to elide most of that in this mix. In a full team, I like to solo, pair, and mob, roughly 1/3 time each. All three are skills, and I have studied them and practiced at them to get better.

To cases, then. The first thing I do when I roll code is reassure myself by whatever means that I meet two criteria: 1) I have a problem I -- more or less -- understand. 2) I have a problem I -- more or less -- can solve in one and a half "good days".

How I do that is neither here nor there for this mix. The point is that I do. I'll push both those limits sometimes, though not usually at the same time. Two and a half "good days" I might live with, or I am pretty sure the problem is "somewhere in that pile over there".

So once I have the problem, the code rolling begins, yeah?

Unless I'm working on a problem I'm already quite intimate with, I start rolling code by, ummmm, furiously not typing any code into the computer.

You ever watch a king-hell drummer set up a full kit for a serious concert? She'll get things into a rough approximation of her standard arrangement. Then she tinkers. Raise that a half-inch. Move that just a little further to the right. She'll hit a thing twice.

As she proceeds -- it might take an hour -- she'll play short phrases where she hits two things, or three. They get progressively closer to sounding like whole little chunks. (It's really quite cool to watch. I, for the life of me, can never quite tell what she's sensing.)

What she's doing is situating herself, centering herself in the elements of her problem and her range of tools for solving it.

I use "situating" on purpose. She is building what the army calls "situational awareness."

Now, her problem and her tools for solving it are very different from mine. But I totally recognize the situating, because that's what I do, too. I even thump a few things sometimes, and make little adjustments to them.

As a pure code monkey, my greatest asset is unquestionably my ability to organize, to rapidly arrange and re-arrange ideas out there in mind-stuff, where the Platonic forms live. To do that, I have to know where things are now, what kind of things they are, their size & shape.

But situating myself isn't just remembering what's there. It's ultimately formulating some kind of change-plan. I do that by doing things that amount to a kind of labeling: "blocker", "detail", "totally irrelevant", "bingo", "scary", "hackable".

Now, "change-plan" sounds like I'm making a list or something. And I might be, if the problem is hard or the existing design is bad or I'm having a nervous day. But just as likely I'm just banging around in the code, scanning things, following links, checking and double-checking.

I might draw a picture. I might rubber-duck it with my long-suffering wife or my infinitely patient dogs. I do some of the work pacing around outside smoking. Sometimes I mentally practice giving a presentation to a roomful of geeks.

Finally, I reach a place where my confidence and my nervousness are in some right tension, and I'm ready to start making some damned changes.

The changes I make are small. In fact, I sort them that way by asking questions in a certain (rough and sometimes varying order).

What's the smallest change that 1) the language will think is legal, 2) is detectable by me, 3) is detectable by a test, 4) is detectable to a user, in that order.

But it isn't as simple as sorting on a single criterion. I also throw stuff in

the mix like "enabling" -- changes that enable other changes come sooner -- "sexy" -- changes that are really cool come later -- "no-brainer" changes I have complete confidence in come sooner, & so on.

In a typical day-and-a-half problem, I might spend two hours just wandering around in the code doing all this. Of course, it's much faster if the code is expressive and tested.

Notice, btw, how much wild uncertainty there is involved in all this pondering and sorting and plan-making and goofing in the code. It's basically guesswork shaped by experience and taste and the gradual refinement of my situational awareness.

And then I change code. :)

In some problems, the bulk of the changes are actually enabling ones. That is, I'm schmooshing the code around typographically w/o changing what it does to get it all lined up for a string of *real* changes.

The other word for schmoosing code around typographically like that is "refactoring". Because I use an IDE, I use and prefer a lot of automated refactorings, because they're fast and (generally regarded as though not guaranteed in all circumstances to be) safe.

But I can't use automated refactorings for everything, so I mix and match manual and automatic. Depending on my confidence level in the size of the change, I may start these changes by writing some tests around "what it does now". I may not.

When I get to a "real" change, I do it by first deciding whether or not it's testable. If it is, I write a test for it. If it's not -- damnit -- I have to

decide whether I can make it testable with some enabling changes or whether it's simply not worth the cost.

Some folks "always" work outside-in, meaning that they start by changing things near the top of the dependency graph and work their way down until they're done.

Some folks "always" work inside-out, meaning that they start by changing things near the bottom of the dependency graph and work their way up until they're done.

I don't "always" do anything, and have and do work sometimes in both of those ways. But mostly, what I do is think my way from the top of the dependency tree down to that core change, then test & code the "enable" changes below it, then test & code my way back out.

I obsess over the size of the changes I make. A typical pull-change-push cycle, in a kotlin app, is something from 3 minutes to 15 in duration. I can go 25-30 if it feels "good, just more time-consuming than I thought". I very rarely persist after 45 minutes. I toss & start over.

That thing, "toss and start over", is *super* important and comes as a surprise to a lot of geeks, young and old, who seem to me a little too attached to what they have already typed.

Typing is easy. Knowing is hard. When I fail at a change after 45 minutes, I'm failing at it because I didn't know what to type. Further, what I typed since I started that change was *ipso facto* not enough to close the change. It is just by-product to me, to be disposed.

(That's also why I don't normally do hand-merges. Good lord, it's just

typing, and it took me as long as it took me largely because I didn't know what it was going to be. Now that I know, I can type it one helluva lot faster.)

So.

Let's wrap this. I can do more of these "How I Work" muses, focusing on aspects other than "Just Programming".

The real point, I think, for me at least, of this one, is that my "system" for rolling code is one centered heavily around cycles of small changes, harvesting of partial value, and using a lot of judgment. There's no lurking algorithm in how I code.

Anyway, it's a wild and crazy Saturday night here. Virginia and I are going to watch a Midsomer Murder episode and eat fantastic carry-out I picked up earlier.

I hope you, too, have a wild and crazy Saturday night with good food, your sweetheart, doing something you enjoy!