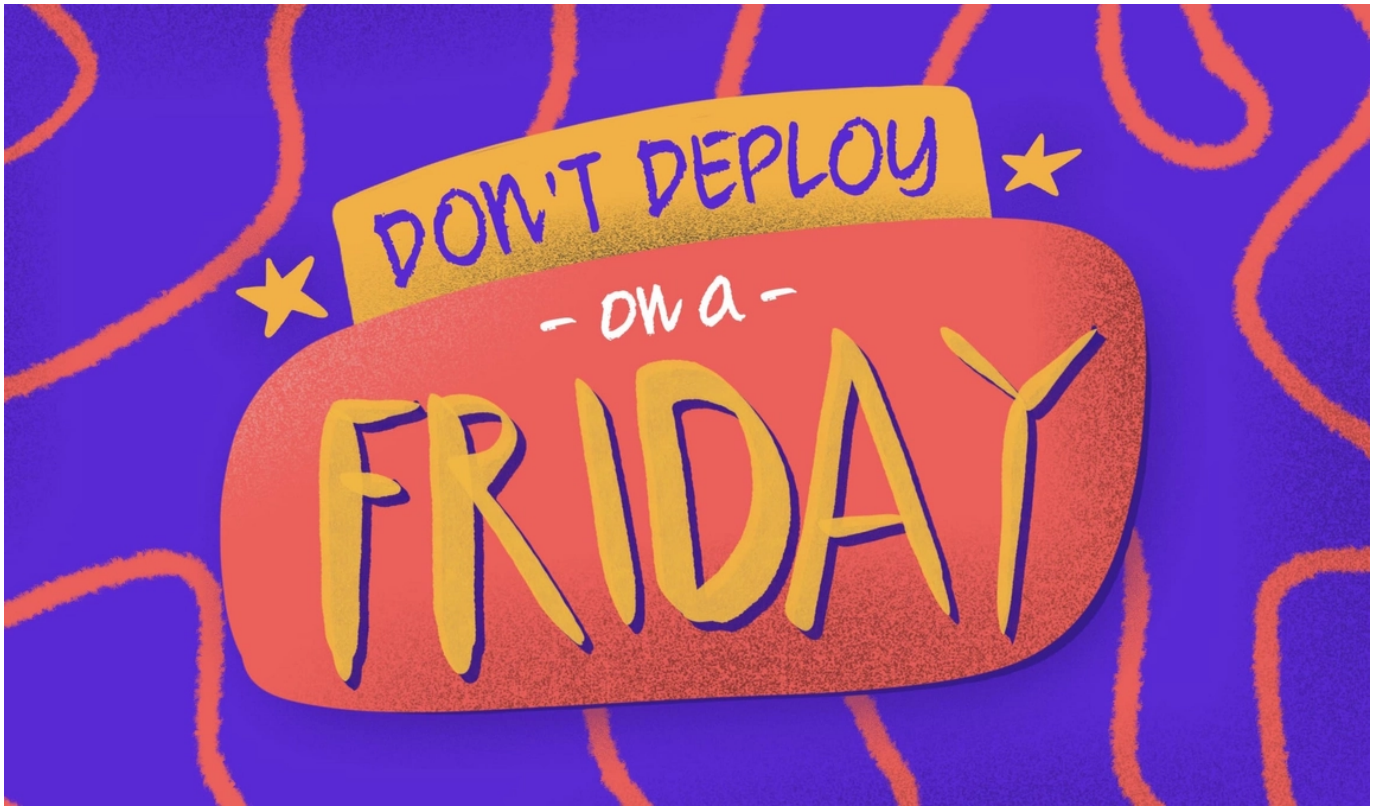
Clubhouse

Software Engineering

"Don't deploy on Friday" and 3 other "unwritten rules" of software engineering

March 13, 2019

Tobias Merkle
Software Engineer



Everything old is one day new again, and there comes a time when even seasoned programmers encounter ancient wisdom in day to day code-slinging. It is impossible to enumerate the “unwritten rules" of any discipline, partly because many of these rules tend not even to be *rules*. Instead, "unwritten rules" often seem to be a discipline’s way of paraphrasing an abstract and timeless truth, rather than a specific insight unique to the art of scrapbooking (for example). That technical stuff is easy to write down.

Marie Kondo has made a career applying principles of efficiency, purity, and beauty to the ubiquitous task of housekeeping, and it turns out that many people just need a translator between timeless wisdom and their daily lives before they really “get” what’s going on. (See also Zen and the Art of Motorcycle Maintenance.) We sincerely hope you enjoy our attempts to do the same.

1. Don’t deploy on Friday.

Yes, even if you have continuous deployment. Fridays are just about the worst possible day to push to master for what should be a clearly apparent reason: you’ve only got half a work day to fix whatever goes wrong. Companies commonly witness spikes in customer support tickets following application updates, especially in the wake of new features and, inevitably, new bugs. People are sleepy on Fridays, not paying attention, prone to push things off… you know how it can be.

No one is immune to the forces that conspire to make Fridays failure-prone! Observe best practices at all times, and look to industry leaders when you’re not sure what best practices are. Apple purportedly ships on Tuesdays, leaving ample time on both ends of the critical deployment event: a Monday to practice and catch last-minute bugs, and Wednesday through Friday to patch

in any urgent changes. And with Apple-sized revenue on the line, you know their deployment model evolved in the field of battle.

2. Maintain regular backups.

Overstating the importance of regular backups is difficult. Our industry is built from the bottom up on a single software idea that is, necessarily, an abstract system of progressive backups. Of course, I am talking about Git, but that's not enough on its own. Your database, cryptographic keys, configuration files, VM images, images/videos... even imported packages ought to be securely stored where they can be reliably accessed in the event of an emergency. Before your dev shop is too big, this might even take the form of a couple devs tarballing the home folder and saving it to an external hard drive. Yes, a couple – it's hard to be too redundant.

Probabilistically, you'll never need 9 out of 10 backups you ever make. But when, as I had to today, you want to roll back the clock and figure out "exactly when did we introduce that invisible but utterly sinister bug that nobody noticed until just now?!" you will be elated you took the time. Or, you know, when a new hire accidentally erases 100,000 rows of code. It happens. We don't hold mistakes against people, but we do learn from them. Back your stuff up, even if it's hard!

3. Get a complete specification before you start building.

I'm often found suffering from "started too soon" syndrome. I have had to rewrite large chunks of a program because I didn't ask enough questions during the first few "please build this" meetings. Our brains are really tiny, and they hate being precise, especially when precision is complicated and expensive (read: essential to the client). They will gloss over stuff that appears similar to stuff that it isn't actually similar to at all. This is especially true in software design –

ten alike buttons can do a dozen different things in a complicated enough application. You've got to hammer these subtleties out before you get hacking if you want to get the most out of your time and the clients'. Don't be like me, or my brain. Sketch it all out on paper, make sure that you "get it" and that you *could* redraw the system from scratch (given enough prep time – we're all humans here).

No one will ever master this process. I still presume far too often. But I've made fewer unreasonable presumptions since figuring this out. It's thinking like software, which is unambiguous and indiscriminate in its logic. Probe corners and edge cases, be the "QA tester's advocate" of your conception of the system design and you will likely invent a way to build it that eliminates whole classes of errors that a less intelligent design would have permitted.

4. If you see bikeshedding, say "bikeshedding."

Try to gently stop or prevent unproductive discussions before they spiral out of control. Our time on company property is almost as valuable as the 3 to 8 hours we spend unconscious every night – which is to say, it's mighty important! Large meetings, in the eyes of the company, had better have significant ROI since a dev's hourly pay multiplied by 20–500 devs in one room/dance hall/etc. has the potential to be an eye-watering sum, especially in sunny California. Time is money! And we work for that pay – twisting our minds around the tentacled monsters that lurk behind every code base, lying in wait to dash innocent expectations with cruel bugs and crashes.

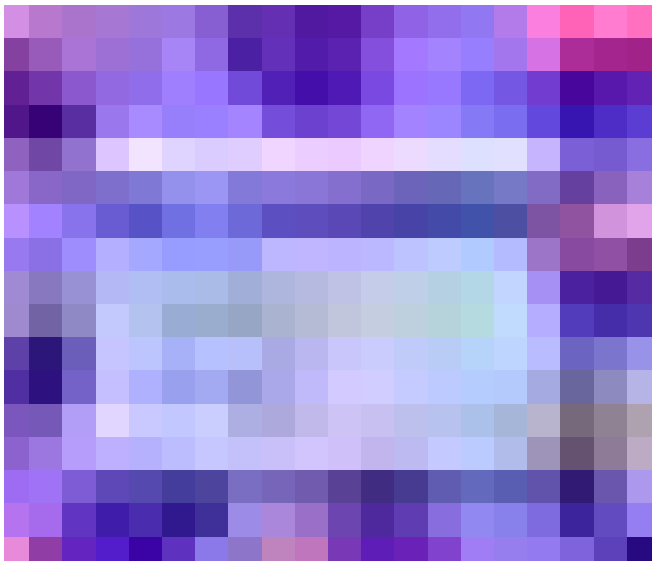
But we're not always wandering forgotten code catacombs with a faltering torch and a sweaty brow. Sometimes we are in meetings, or discussing things intently while not formally in a meeting, but unable to do anything else. Slack is famous for its role in increasing connectivity between team members, but, at what cost (aside from the monthly fee)? In my experience, Slack dramatically increases your brain's "nonsense time" – time spent filtering yet another constant stream of (often irrelevant) buzzes and beeps while trying to get going

on the thing the company actually pays you to do; that is, to build and repair useful things. And folks tend to slide into unnecessary disputes rather easily.

It's okay to ask for a discussion to end if it's taking too much time. The kids these days like to say that "real recognize real": the people who are at work to work, to make valuable things that help clients, users, and the world – they'll thank you for tabling the question of whether the new fridge should come in black or chrome. "Flip a coin for all I care... we have real problems to solve." And many of the people who are sure to notice will be those in management.

Thank you for joining me on this journey. May you see the wisdom that those before us saw when they first wrote these rules. And please, if you have collected any rules on your own travels, write them down and let us know!

More stories from Clubhouse



Clubhouse News and Updates

Execute commands more efficiently with the Action Bar

February 5, 2020

Clubhouse News and Updates

Collaborate faster at scale with Group @mentions and notifications

February 3, 2020

Clubhouse News and Updates

Update on the Clubhouse Write private beta and what's next

January 28, 2020

Name

Marcus Vipsanius Agrippa • 10 months ago

Get a complete specification before you start building ? That won't work well with Agile...

1 • Reply • Share ›

Francisco Quintero • a year ago

I learnt the hard way to follow the Read Only Friday rule xD

The other unwritten rules are also very important though bikeshedding even has a webpage.

1 • Reply • Share ›

R. Scott Perry • 8 months ago

Please try to find a better source for "purportedly" than a 5-word comment on StackOverflow from over 9 years ago (or, if you can't, simply omit a source).

• Reply • Share ›

Bob • 8 months ago

I wish you would have properly defined 'bikeshedding'...or better yet, not used a word that many people are not familiar with. Using idiomatic expressions in a 'professional' piece is not...professional.

• Reply • Share ›

Doug • 8 months ago

I had never heard the term bikeshedding before, so thanks for that.

On incomplete specifications, the old term for that is 'bring me a rock'. Don't play that game. Even for agile development, the current sprint requires specifications for that sprint.

And yes, backups matter. I know of one particular horror story from about 1990 where a crucial machine had not been 'successfully' backed up for about a year and it crashed a disk head. They were able to recover the project, between data on individual developer machines and through an external forensics team that disassembled the platters and reconstructed most of the drive. But it cost about two months of time and a couple million dollars.

• Reply • Share ›