Code Awareness – other thoughts

We are increasingly dependent on software in all areas of life and business. Software is still being built using traditional assembly line methodologies. Even the most agile companies have only adjusted their business communication layer, while relying on the same old divide-and-conquer, siloed engineering tactics for development.

I think software is grown, not built. Like any organism it grows from a concept, a prototype, and is structurally changed multiple times until maturity. If we allow ourselves to continue this metaphor we can imagine our software product as being the entire population of a certain species. Then the evolution of this software product will be peppered with forks in the road, competing factions, die-offs, diversification, and some truly radical changes too.

As much as we’d like to call our work “engineering” and automatically benefit from the rigorous standards and guarantees that true engineering came to hold over millennia of thought and practice, the truth is we are doing something different, something that encompasses engineering principles, gardening principles, social principles, and even politics.

To make things work better we need to obtain more metrics and improve several aspects: education, collaboration, communication and soft-skills.

Sometimes trying to explain your vague idea to the AI is hard, and it would be useful to just play with the diagram yourself. It is exactly this reason why we still need regular tools.

If we can somehow create an AI assistant that learns the maintainer’s patterns and line of thinking, it could guide the contributors to submit code closer to something the maintainer would approve.

Code Review issues today:

* In Code Review no one can hear you scream: <https://stackoverflow.blog/2024/07/05/what-can-devs-do-about-code-review-anxiety/?utm_campaign=the-overflow-newsletter&utm_medium=email&utm_source=iterable>
* Code Review LGTM: <https://medium.com/convoy-tech/code-review-tactics-3367baaa57ea> (thorough code reviews at Google)
* “Avoid changes during the code review”: <https://www.freecodecamp.org/news/code-review-the-ultimate-guide-aa45c358bbf5>

• ⁃ “I have seen something similar across my career of 2+ decades. I think I have found a pattern, when there aren’t enough problems to solve, people start owning code they have implemented. Owning leads to becoming very territorial. Their identity in the company is very tightly coupled with the code they own.”

WHAT PROBLEM DO WE SOLVE AT CODE AWARENESS

* Trying to break the silo work style and bring more joy to the field of programming.
* While working, you have a clear visibility on which tasks affect which files, and specifically which lines of code in those files
* Management has a clear visibility on the amount of work being done on each business task
* The team can clearly see where the most probable conflicts can occur
* The team can better estimate impact of changes both upstream and downstream in the OOP model (class modifications and extensions, template changes, etc)

MUD programming tools, existing vs Code Awareness — distributed mud programming instead of today’s local

Maybe I should re-think about this whole thing from a more baseline level. So far I’ve been working around the idea that I am providing a platform for communication between different applications.

What if I am just focusing on effective means of collaboration between people? What could be a next-level methodology or platform, that truly increases the effectiveness of teamwork?

And we could start from listing the problems we know exist in teamwork today.

Challenges unsolved

* get an architectural overview of a repo, just by looking at code. AI.
* ML patterns of code that cause issues. Using a certain library can be correlated with bugs/issues, or using a certain version of it. The length of a method chain. Etc
* Serverless and microservice architectures are difficult to reproduce locally, which makes development a bit painful.

Security:

* need to be SOC 2 and ISO27001 compliant, example [Security Overview](https://docs.swimm.io/security/)

OBSERVATIONS

As a human being we don't have the ability to adapt too much at once. We can follow a procedure blindly, but we need incremental improvements on our daily routine to feel comfortable and not break things, including our mental health. For example, you would encounter pure chaos if you try to teach students programming, as well as deployment, linting, CICD, systems architecture, all in two weeks. People need to get comfortable with one thing before moving on to the next. However, it is this transition that causes the most problems. We don't like to break our habits, we don't like to learn something only to break it and do it another way. We are creatures of stable rituals. And yet, it is precisely the breaking of rituals that makes us shine. Like breaking glass, then melting it smooth again, then breaking it again, it is the constant but not rhythmic breaking of my rhythm that keeps me from falling into stagnation.

Problem: we forget to update related information.

Example: we change the code, and forget to update the online (web) documentation.

Imagine you own a software development company, either a startup or an established one, and you have to figure out the best way to get your team working together. You might take a traditional assembly-line approach, and hire an architect, a few business analysts, some product managers, a bunch of team leads and two programmers, and just let them cut job tickets on Jira. Or you might go the opposite way and hire a bunch of brilliant people, and give them free rein, and a few hundred Slack channels.

Sometimes I find myself lost for days in trying to find the best library, or the best build system for my project.