Thoughts on existing software development process

Code Awareness does code approval and comments in a non intrusive way, without the need to push to GitHub or GitLab.

Different approaches:

1. Deep Planning. We go and architect everything in minute detail. We spend months doing just planning and estimation. Then when it comes to implementation we discover new things, new gateways, new obstacles.
2. Into the fire. We start prototyping. Then we evaluate it and rewrite the prototype a few times. Then we expand on it. Then we realize we need to rewrite it again, from scratch.

Different workflow paradigms:

1. We fill out a form, verify it thoroughly, then click save. Or we modify a document, then click save.
2. We don’t have a save button. Instead we have auto-save and unlimited undo.

Speaking of paradigms, I’m thinking today’s most successful one is the Javascript prototype inheritance model. Why? Because vanilla JS is memory efficient, and quite fast. Mapping this model over document collaboration would be something like this:

* one person creates something, any document that can be represented as text (maybe pixel matrix in the future).
* all other people can see the document
* when someone makes some modifications, it automatically becomes a fork of the original document
* **the system captures the fork’s differences with the original, which become highlighted zones in for users looking at the original document**
* the fork and the original can be partially or totally merged; a complete will reunite the two documents, and **now people are looking at a single document again**
* the fork or the original may also be deleted, leading to the same result: a single document again

This is essentially how git works, except we added one important component to the versioning system: the ability to see at all times the blocks of code that diverge in different branches.

— PROBLEM: startups are often losing programers because they don’t get along, fight over code, don’t communicate well enough. Code Awareness solves the communication and code fighting problem, by engaging programmers with each other as they work.

— PROBLEM: how do you combine people in the best way? What does “best” mean?

Tactics:

* Context Items can be merged when they are marked “clean”, meaning they don’t affect the existing functionality.
* Context Items that are not completely isolated are called “diffused items”, and they can only be merged manually
* Context Items can be modifications of a popular library, for example, in which case we can track the “mutation rate”, or how far from the main trunk we have departed.
* Show metrics related to the line of code you're currently on: dependency index, documentation etc
* New agile methodology: plan, create documentation, then create code based on things that will be created by others; once the other have completed the dependencies you’ve been expecting, it should all fall into place.