Twitter Thread by Michael Feathers



Michael Feathers
@mfeathers

One of the things I learned from @KentBeck years ago was to see cohesion in terms of divergent change rates. If you have a class and there's a set of methods that you tend to change together while leaving others alone, that set of methods could be a separate responsibility.

Once I saw this, I went so far as to write a script to mine classes in Git and find clusters of methods that all seemed to change at the same time.

Often the results were not surprising. You could've see new responsibilities emerging just by looking at the names of methods and variables. But, sometimes this time view of the code was the first indication that a class was trying to split.

Code isn't special that way. This is all systems stuff. If you have a group of 10 people and 3 of them often change their minds together, you can start to see them as a separate group. They may even start to see themselves as a separate group. Why does this happen?

At its base, it's because of the tension between N and N^2. As the number of things increases, the number of possible interconnections grows (bounded by N^2). It's harder for 10 people to coordinate than 3, so we shouldn't be surprised to see groups from cliques periodically..

..it's a natural tendency in systems. Sort of "the grain of the wood."

Code is the same way. It grows through human attention. We're loosely bounded in the number of concerns we can keep track of at once. But, again, it's not the number of concerns that is the significant limit, it's the number of ways that they can interact.

So, we focus, and form little areas of code around concerns even when they aren't even consciously apparent to us yet. If we ignore our felt-sense of these dynamics we end up with legacy code.

Legacy code.. So, let's relate this back to divergent change. I think that the core problem of software development is that code and team change at different rates.

If team turnover is faster than code turnover, you have knowledge loss. You need to prop up your code with more tests and documentation, and hope for the best.

There’s only so much that can be done about this problem. People are going to move on. The code that they wrote will remain.

A hardcore development organization would rip out the code of each developer as they leave.. by the roots, and rewrite it so that the knowledge of it is fresh in the remaining team, but I’ve never seen that as a consistent practice.

Code is perceived to have too much value. In reality, the active knowledge of a cohesive team is the value.

Mob programming is one manifestation of this understanding. It's one way to go.

But, to me, the important thing is to understand these costs of coherence and coordination. When we focus on them, we have many possibilities.