Lightweight Code Review Episode 5: Team Building for the Cold, Dark, and Alone

In Episode 4 we presented data from the largest case study of lightweight peer code review ever done. Now let's explore how peer code review builds teams.

Teams are for sissies!

"Team building" is a load of bull. It's happy-talk that changes nothing. It's laughable trust exercises. It stifles the best developers to serve mediocrity.

Right?

This essay isn't rah-rah go team go. Suspend your judgement for five minutes.

I'll give you the punch line first. Peer code review not only finds bugs, it creates an environment where developers work *together* instead of in parallel. It's team-building in the real sense -- people helping each other to achieve something greater than each could achieve alone.

Think you have a team? You don't have team.

I'll bet most of the code in your application has been seen by only one person. Think about that for three seconds. It's a frightening thought. Teams supposedly work on projects together. Do you really have a team?

Developers write code while squirreled away behind doors and headphones. After the architecture and interface discussions are over, code is written in isolation. Does that sound like a team to you?

New hires want to get into code fast and get their hands dirty but they need a safety net to make sure nothing bad happens. How do your developers handle this? Isn't "teaching and

supporting" a team activity?

"Two or more draft animals used to pull a vehicle or farm implement." That's how the American Heritage Dictionary defines "team." This is the extent of "teamwork" in most software development organizations too. But being a team doesn't mean we're merely marching in the same direction.

You're not a team just because you know everyone's name and you eat lunch together. That's just being acquainted. Real teams actually *work* together. Let's see how peer code review can help build a real team.

One is the loneliest number

"It's scary how much code I've written that no one else has ever seen." I was told this by a developer who had just started using our software to do peer reviews. "The worst part is, I know it can't all be right."

Yes, you *should* be scared. The average professional novelist makes 9 mistakes per page. No surprise; that's why there are editors. In fact the 2000 Census showed that in America we have more professional editors than professional writers!

So why do we expect developers to write reams of technical code without the safety net of an editor? Aren't we *guaranteeing* errors?

A startling amount of code has a bus number of one. Your "bus number" is the number of people who have to get hit by a bus until no one is left who understands the code. A software process is easily derailed if a significant amount of code has a bus number of one. If that someone is sick, leaves the company, or really is hit by a bus, parts of a project can stall or might be implemented incorrectly.

Code review provides a safety net for both developers and managers. Managers like the project stability that comes from having a bus number of two or more. Developers like the comfort of knowing that someone has their back. In a real team, people look out for one

other.

"Those guys suck."

It was a large software company. A small overworked software group in Austin had been asking for additional developers; finally their wish was granted in the form of 20 developers in India.

The group in Austin was pissed off. They were responsible for running the remote group and incorporating their code into the local base. "Those guys suck." (Yes, that's an exact quote.) "They don't know how to write code. And they don't do it the way we ask them to. And they don't write comments. Forget about comments in English, they write no comments at all!"

The Transformation

The Austin group started using our code review software to check everything being done in India. This doesn't just mean looking at code -- the software facilitates actually communicating on individual lines of code and having a two-way conversation like a chat room or a newsgroup.

We visited them four weeks after the reviews started. "They're pretty cool" said the same developer who had told us they "suck" only 29 days earlier. "They're doing a good job now. They just needed a little direction. We're actually getting a lot of good code written."

How is it possible that developers who "suck" suddenly became "pretty cool?" It turns out that the Indian developers were inexperienced. They were smart and wanted to learn; they were just green. Once the two factions had a mechanism for easily talking to each other about code, suddenly the knowledge transfer could begin.

It wasn't enough when they were only communicating through project specs and long emails about how Austin expected them to work. They needed to talk about specific pieces of code and they needed a process that let this happen naturally.

Emergent Behavior

Once the code review process was in place, the rest literally worked itself out. The developers in India became far more productive and the developers in Austin became happy. The entire group produced far more working code without increasing the budget.

The developers *organized themselves* into a team. All they needed was an environment of communication and a tool to make it easy. They taught each other without managerial pressure or rules or strict process controls.

Working together, teaching and sharing, being happier, all without being told... sounds like a team!

Teamwork, for real though

Software architecture and design is typically coordinated and even invented in meetings. We take it for granted that these activities are enhanced by collaboration.

Writing the software is no different. You may not need a formal meeting or the time investment of pair-programming, but it does make sense to give your developers "editors." Besides the obvious bug-removal benefits of code review, you'll turn your isolated, individualistic developers into a true team.

I hope you teamwork-disbelievers out there will at least give code review a try. There are lots of ways to do it; just pick one that sounds doable and try it for a month. If you don't find lots of defects and you don't see developers teaching each other, let me know.

What's Next: *In the next article I'll talk about more social issues with code review including the "Ego Effect" and dealing with hurt feelings.*

