

# Jay Fields' Thoughts

experiences in software development

---

Monday, September 15, 2008

---

## Is Distributed Development Viable

---

I've never seen distributed development succeed. However, before we get into what I've seen, I need to be specific about what I'm describing.

[Distributed Development](#): A group of individuals who work across time, space, and organizational boundaries with links strengthened by webs of communication technology

The link above for Distributed Development redirects to *Virtual Team*. Wikipedia also has an entry for Distributed Development; however, I found the description of Virtual Team to be much closer to what our industry generally labels Distributed Development.

Just to be clear I'm not talking about [outsourcing](#) (subcontracting a process, such as product design or manufacturing, to a third-party company) or what you might call off-shoring.

This entry will be written with that in mind.

### Take 1

Several years ago I worked with one of the nicest and most talented developers I expect I'll ever encounter: [Badri](#). We worked together for about a month before he was forced back to India to deal with US visa issues. We knew this was going to happen and decided that it was so important to keep Badri on the project that we were going to give distributed development a shot. Badri knew the code, the team, and he was great on so many levels that the choice couldn't have been more obvious.

After just a few weeks, we gave up on distributed development. Ever since that experience, I've been highly skeptical of the feasibility of distributed development. I had actual experience working with one of the greatest developers I'd ever encountered, and we still failed.

Failure can be good. There are always lessons to be learned when you fail. Here were a few that I picked up from that project.

- Time difference is probably the single largest contributing factor to failure. Badri was in India, and we were in Delaware, USA. There was almost no time overlap to our workdays.
- Having the *local* or *on-site* developers feed requirements back to the *off-site* team equates to an order of magnitude efficiency loss.
- Having the on-site developers decide what pieces are going off-site quickly translates to

the off-site team being pissed off about getting the boring work

- Businesses expect the same level of quality and productivity from the on-site members as the off-site members.

Given those circumstances, we definitely failed. [Epic failed](#) in fact. And, to make things worse, I was the one that had to tell Badri that we were going to give up on distributed development. Imaging having to tell someone you have nothing but admiration for that "it's just not working out". I still feel guilty about it to this day.

The time difference was a huge killer. Every time we spoke to Badri either he was half asleep or we were. I think Badri was working 12 hour days also, just so he could try to talk to us at the beginning and end of his day. It also meant no wine with dinner or having a bit of a buzz for the 11pm stand-up. Either of those options is unacceptable. Worse, if we didn't communicate what we needed in detail, the off-site team would be off in the wrong direction for up to 12 hours straight. We threw away several days worth of code because we hadn't given them enough detail to go in the right direction.

Truthfully, we (the on-site developers) shouldn't have been deciding what they were going to work on in the first place. It was a tough project and we were all learning. None of us felt like we had the time to do both the development required of us, plus the analysis required to give the off-site team good direction. Instead, the off-site team got minimal direction and delivered code that was of minimal use to us. It was beautiful code, but it wasn't what we needed. Again, epic fail.

Even though we clearly were not functioning well, the business expected that the off-site team deliver at the same pace as the on-site team. The business had met and loved Badri. No one could understand why our pace had suddenly dropped off. Not good. More on this later.

## Take 2

A few years later I joined a project that was just beginning to give distributed development a shot. I hadn't been there 3 days and I was already in the CEO's office explaining why I thought it was a bad plan. Of course, I was gun-shy, so that was to be expected. Luckily, [Fred George](#) was also there and he was more optimistic. In the end we went to an on-site team, but Fred showed me that distributed development is possible.... maybe.... eventually.

We experienced several obstacles, and we overcame some. Again, there were plenty of lessons to be learned.

Time difference was less of a factor. This time I was working in London and the off-site team was in India. Our workdays overlapped fairly well. This was obviously a huge move in the right direction. We also collaborated on who would work on what, which helped ensure that everyone was happy with what they were working on. And, the analysis came from a BA that collaborated with both portions of the team. The BA even traveled to India occasionally.

However, things were still quite broken.

Some members of the team had never met other members of the team. There's something about

working, collaborating with someone in person that is almost impossible to replicate over a phone or IM conversation. It wasn't that we didn't want to get everyone together, but there were visa issues that couldn't be overcome. It was a big mistake with no good resolution. Some long time team members couldn't travel, but if we replaced them with people who could travel we would lose domain knowledge and context.

Communication was a constant problem. The telephones on both ends presented problems. I have plenty of Indian friends that I have no problem understanding, but with the off-site team speaking into a bad connection and it being sent out of our bad connection, I caught every other sentence, if that. To make matters worse, people weren't talking nearly as often as they needed to be. We tried to address this by creating a chat room and mandating daily checkpoint phone calls. Both ideas were abandoned due to lack of buy-in. In the end we never did solve the communication issue -- in my opinion. I think the reality is that most programmers are introverted, and being off-site just gives you one more excuse for not talking to the business, even when you really should.

Connectivity was also an issue. If you're in the US, you don't really even think about the reliability of your internet connection. However, in many other places in the world connectivity is much less certain. There were several occasions where the off-site team was simply unable to check-in their code. We never did figure out if the problem was on their side, our side, in our vpn, or some other bizarre location. If your off-site can't check in, IM you, or get to the wiki, you obviously lose productivity.

There was one common problem: the business expected the same level of productivity from both teams. I'm not sure there's a way around this issue. Have you ever heard that programming is about people? Of course you have, and you obviously believe it. But, does the business know that? Even if they've heard it, do they believe it?

I doubt your business does. Frankly, I doubt most of your colleagues in the industry do. I still know far too many programmers who think that their only responsibility is to write code that creates features. If that were true, if we were nothing more than line workers delivering feature after feature, it would be plausible that productivity shouldn't suffer no matter where the factory exists.

But, it's not true. Being a software developer is about communication, collaboration, analysis and coding, at least. Being a phone call away instead of a face to face conversation away impacts communication and collaboration. Thus, productivity is impacted. There's no getting away from that.

How can you convince your business of that? I haven't solved that problem yet, unfortunately.

### **What could be**

I do think Fred had the right ideas. He described scenarios that had previously benefited from distributed development, and what made those situations succeed.

The first suggestion is to get everyone together. You want the team to gel as one entity. Do version one entirely on-site if possible. However, don't do it with all local resources. Bring people

from the desired off-site location to work on-site for the first release. The team members will build trust and friendships that last up to 6 months.

Once you are on version two you can move the desired team members back off-site. However, the travel isn't over for anyone. The off-site team should always have at least one member from the on-site team with them, likewise the on-site team should always have at least one member of the off-site team present. These aren't week long trips either. Each member of the team should visit the other location for a month, once every five to six months. That level of in-person communication should lead to high levels of trust and understanding.

The travel situation is even more drastic for the analysts and stakeholders. They should split their time between the two teams, if possible. Neither team should feel like the A team or the more important team. Any implication that one team is above the other team will lead to negative productivity impacts.

That might sound drastic, but it's the price of doing off-site development. The cost doesn't stop there.

Both team locations are going to need to invest heavily in infrastructure. The best video conferencing software and highly reliable bandwidth will also need to be purchased. The idea is to foster communication in every way possible. Without communication and collaboration, the project is doomed.

### **Open Source**

Before anyone points out that it's hard to argue with the success of Open Source, I'd like to be clear -- I'm not. Open Source is obviously successful and developed most often in a distributed manner. However, there are a few differences that, I believe, make it a different situation entirely.

First of all, most people aren't paid to work on Open Source. When someone isn't paying you, you can often do whatever you feel like, whenever you feel like it. If someone is working in the same area of the code as you need to, you can just put off your changes until they are done.

However, the reality is that most people aren't usually working in common areas when they work on open source. Most open source projects are maintained by a few people who work within specific portions of the codebase. If changes need to happen in "your" portion of the codebase, you often queue them up to work on after you finish your current task.

Since there's little conflict between what you work on and what other team members work on there's significantly less communication and collaboration required, and what is necessary can happen at a much slower pace. If you need to make a change, it doesn't often need to happen right away. It can be put off until the team member on the other side of the world wakes up.

The codebase also evolves at a much slower pace. Six to ten people working in the same codebase 8 hours a day move much faster than the average Open Source project that sees 3 developers working a few hours a day.

Distributed development does work for Open Source, but that's not what I'm talking about.


### Conclusion

I have heard of companies successfully doing remote pair programming and distributed development. One recipe I've heard is that everyone is off-site in different locations. I can see how that would work since it requires everyone to adopt a new work routine and make the best of it. While I believe it's possible to be successful, I think it's still bleeding edge at this point. You probably don't want to "try this at home" quite yet.

I think Distributed Development is probably the way of the future. As bandwidth and experience is more available the industry will continue to evolve in that direction. However, I think it's still probably about 5-10 years from being mainstream.

Labels: [developers](#)

Share: [Email](#) | [Del.icio.us](#) | [Digg](#) | [Reddit](#) | [Shadow](#) | [Rojo](#)

# posted by Jay Fields : 11:00 PM   
[ general focus: [Ruby](#) - [Ruby on Rails](#) - [Ruby Rake](#) ]

### Comments:

So i have to say that a lot of the issues you presented sound symptomatic of distributed, *multi cultural* development. In each Take above, one of the development parties were located in India.

I would be interested in hearing any experiences you have with distributed, monocultural development.

I've been working in a distributed development environment for the past 2+ years and we've flourished. Not only as a dev team, but as a company. However, all parties are, at most, a time zone apart.

# posted by  Griffin : 1:02 AM

Distributed development is viable and more and more folks are doing it successfully.

Your "what could be" section has some of the known good practices in it. See my [2005 paper](#) for some more.

Meanwhile, at Agile 2008 Jeff Sutherland at al [reported](#) on a successful distributed project and concluded that not only can it work, it can work so swell that maybe it should be your preferred choce of project arrangement

# posted by  keithb : 4:00 AM

Jay, I'd like to report briefly on our success with distributed development. We have 3 in-house developers, and 4 developers spread across the U.S. with one over in Europe. We