CODING HORROR
programming and human factors
by Jeff Atwood
*

November 18, 2007

Pair Programming vs. Code Reviews

Tom Dommett wrote in to share his positive experience with pair programming:

> The idea is two developers work on the same machine. Both have keyboard and mouse. At any given time one is driver and the other navigator. The roles switch either every hour, or whenever really. The driver codes, the navigator is reading, checking, spell-checking and sanity testing the code, whilst thinking through problems and where to go next. If the driver hits a problem, there are two people to find a solution, and one of the two usually has a good idea.
>
> Other advantages include the fact that where two people have differing specialities, these skills are transferred. Ad-hoc training occurs as one person shows the other some tricks, nice workarounds, etcetera.
>
> The end result is that both developers are fully aware of the code, how it works, and why it was done that way. Chances are the code is better than one developer working alone, as there was somebody watching. It's less likely to contain bugs and hacks and things that cause maintenance problems later.
>
> In a bigger team, the pairing can change each week so each team member is partnered with somebody different. This is a huge advantage, as it gets developers talking and communicating ideas in the common language of code.
>
> We found this to be as fast as working separately. The code got written quicker and didn't require revisiting. And when it did need to change, more than one person was familiar with the code.

It's an encouraging result. I applaud anything that gets teams to communicate better.

I'm intrigued by the idea of pair programming, but **I've never personally lived the pair programming lifestyle**. I do, however, enjoy working closely with other developers. Whenever I sit down to work side by side with a fellow developer, I always absorb a few of their tricks and techniques. It's a fast track learning experience for both participants. But I've only done this in small doses. I'm a little wary of spending a full eight hours working this way. I

suspect this might be fatiguing in larger doses, unless you're very fortunate in your choice of pairing partner.

I've written about the efficacy of code reviews before. That is something I have personal experience with; I can vouch for the value of code reviews without reservation. I can't help **wondering if pair programming is nothing more than code review on steroids**. Not that one is a substitute for the other-- you could certainly do both-- but I suspect that many of the benefits of pair programming could be realized through solid peer review practices.

But code reviews aren't a panacea, either, as Marty Fried pointed out:

> My experience with code reviews has been a mixed bag. One of the problems seems to be that nobody wants to spend the time to really understand new code that does anything non-trivial, so the feedback is usually very general. But later, when someone is working on the code to either add functionality or fix bugs, they usually have lots of feedback (sometimes involving large hammers), but then it may be too late to be effective; the programmer may not even be around. I think it might be useful to have one anyway, but it's hard to get a fellow progammer to tell his boss that another programmer did a bad job.

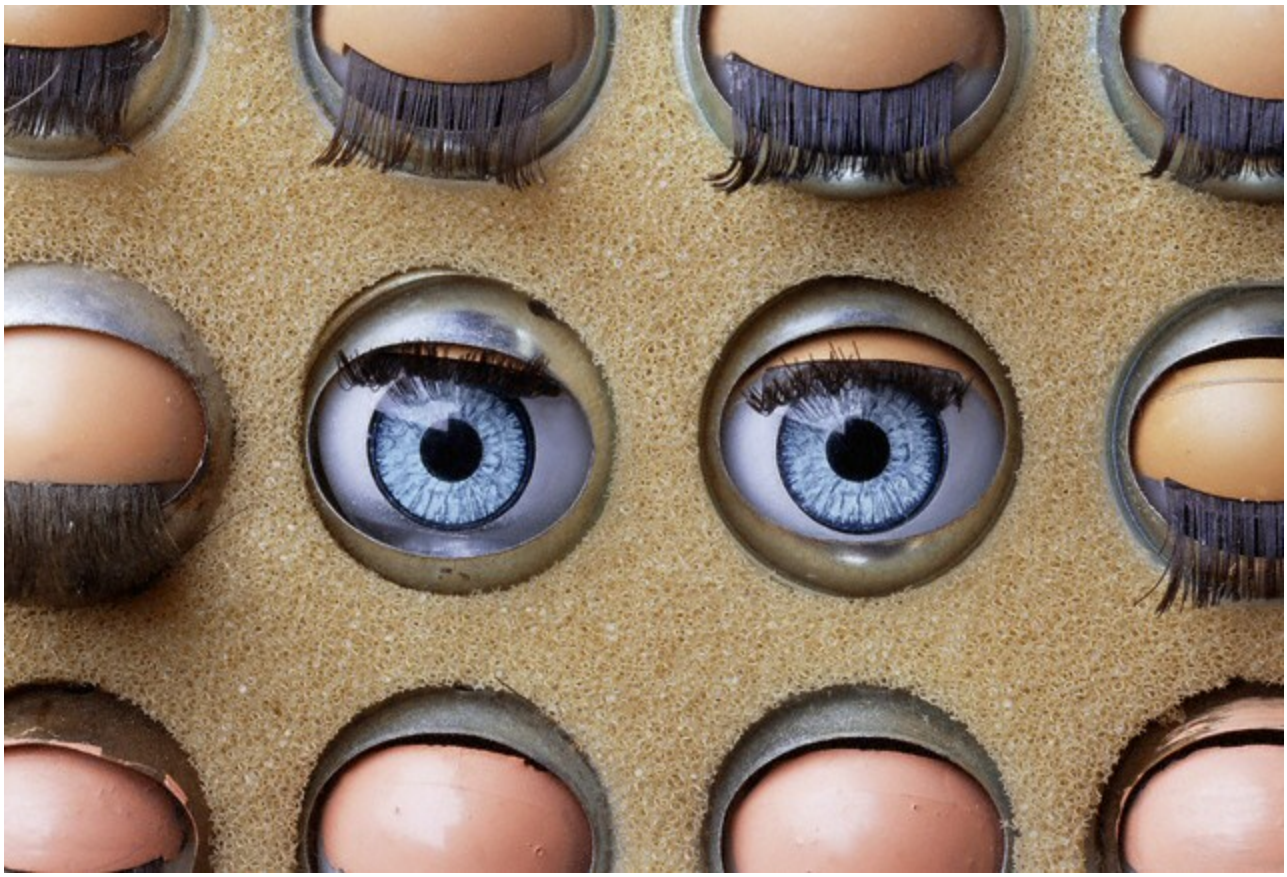
The advantage of pair programming is its gripping immediacy: it is impossible to ignore the reviewer when he or she is sitting right next to you. Most people will passively opt out if given the choice. With pair programming, that's not possible. Each half of the pair *has* to understand the code, right then and there, as it's being written. Pairing may be invasive, but it can also force a level of communication that you'd otherwise never achieve.

On the other hand, peer review scales a heck of a lot better than stacking physical bodies in the same area. Consider the experiences of Macadamian with code review while working on the WINE project:

> There were two processes in the WINE project that we weren't used to: public peer reviews, where new code and patches were distributed in a mailing list to everyone involved in the project; and single committer, where the project leader had the final say over which patches were accepted into the source tree.
>
> We soon found out that Alexandre Julliard, who has been the maintainer of WINE and one of the key developers since 1994, was very particular about code going into the source tree. Our team's patches were scrutinized, and when some were rejected, there was a lot of grumbling. "My code works, who does this guy think he is? We're on a deadline here!" But as the project progressed, we realized we were producing our best code ever. Producing clean, well-designed code that was admitted into the source tree at first pass soon became a matter of pride. We also found that, despite the fact that the project was huge and spread

> worldwide, we knew exactly how the whole project was progressing since we saw every patch on the mailing list. We now conduct code reviews on every project, and on larger projects, we set up an internal mailing list and designate a single committer. It may be painful to set up code review at your company, and there may be some grumbling, but you will see big improvements in the quality and maintainability of your code.

I think both techniques are clearly a net *good*, although they each have their particular pros and cons. I encourage people who have experience with both pair programming and code reviews to share their experiences in the comments. Is one more effective than the other? Should we do both?



In the end, I don't think it's a matter of picking one over the other so much as **ensuring you have more than one pair of eyes looking at the code you've written**, however you choose to do it. When your code is reviewed by another human being -- whether that person is sitting right next to you, or thousands of miles away -- you *will* produce better software. That I can guarantee.

Posted by Jeff Atwood View blog reactions

« You're Now Competing With The Internet Living the Dream: Rock Band »

Comments

Over at our office we have some sort of an in-between: four-eye commit. You develop on your own machine, but whenever you want to commit something to revision control, you call in one of your co-workers to show him all changes you made.

Since we combine this with small commits, this is like a code review multiple times a day.

Rick on November 19, 2007 12:40 AM

Evolution, not revolution. OpenBSD.

Joe Nathan on November 19, 2007 12:54 AM

One factor you don't mention, Jeff, is management buy in. If your organisation isn't currently going in for pair programming, it's very hard to convince management that it is a good idea. They can't believe that it can do anything but halve productivity. Code reviews, on the other hand, is an easy sell. Another factor is geographical spread of the team.

I am a contractor, and so move around quite a bit. My last contract was in a London-based XP team, and so involved a lot of pair programming. My current one doesn't, although I have now introduced a system of code reviews using Crucible. We have a geographically spread team of developers (four in London, two in Mumbai), and so in many ways code reviews work better for us than pair programming would. But I still miss certain aspects of the pair programming. I certainly code better when pairing than not. Sure, code reviews help pull up the standards as well, but to my mind not to the same extent.

Rick's approach is interesting as well. Having looked at TeamCity recently, with its delayed commits, I wonder if a second pair of eyes could form part of such a delayed commit approach? Anyone know of any tools that allow for this (Mumbai is a long way to walk for a second pair of eyes ...).

David M on November 19, 2007 1:07 AM

In my experience, "peer code review" does not result in all the bugs being caught. For any large code changes, I only see general comments in "peer code review".

In my opinion, "pair programming" gives me a better bang for the buck! Having two

engineers is better for the morale of the team, improves communication, provides real training and helps me do away with the paper work of a "peer code review" process.

There are some disadvantages of "pair programming". They are not big enough for me to be worried about but some other people might find the information useful -

1. The day-to-day progress is slower than when a single programmer is working on the project. However the end result is a product of much higher quality so its worth it! In those organizations where each phase of the project has its own deadline, it is very difficult to introduce "pair programming".

2. It requires great discipline and oversight to make sure that the "pair programming" routine is being followed. The strong programmers have a tendency to takeover the coding process while the weaker programmer would be happy to stay in the reviewer's role without significantly contributing to the project.

Adnan on November 19, 2007 1:11 AM

One of the main problems I see with this (that you didn't mention) is that the obvious conclusion to this is double the hours per project, at minimum (and I'd expect that you would work slower if you had to discuss or explain stuff to someone else the whole day). Double the hours puts a massive mountain for the advantages to climb out there. Is pair programming really that much better that it's worth twice the money ?

Also, I personally don't really like to talk a lot. I picked an office job in programming in part because it lets me spend most of my day in silence and concentration. Having to work that closely with someone else all day would certainly put a dent in job attractivity for me.

J. Stoever on November 19, 2007 1:17 AM

IMHO it might even be better on the days when you aren't feeling up to much for your productivity/momentum to pair program - because you may not feel like doing a whole lot yourself, but you can usually get into it if someone else helps you stay interested.

I know it would help some days when I start to procrastinate and lose more time than I realise.

But then I'm in a case where I'm very isolated and don't have much feedback other than a fortnightly dev session with the others in the group - and they don't develop in my section

and I rarely see their code.

We've discussed Code Reviews etc, but it always seems to be the same guys reviewing each others code because they feel each other know and understand the code best - and that doesn't really help anyone learn or grow outside of those guys.

So yeah, the few times I've pair programmed I've learnt a lot and been a lot more motivated in what I was doing.

Andrew Tobin on November 19, 2007 1:21 AM

> I can't help wondering if pair programming is nothing more than code review on steroids.

The practice of pair programming was popularised via Extreme Programming, which has as part of its philosophy that "if a programming practice is known to be effective, take it to the extreme and do it as much as possible". That philosophy applied to both "code reviews" and "collaborative code design" give the practice of pair programming. http://c2.com/xp/PairProgramming.html

But it's a misconception to think that code review is the *only* benefit to be had from pair programming.

For pages discussing objections, see http://c2.com/cgi/wiki?PairProgrammingObjections http://c2.com/cgi/wiki?PairProgrammingQuestions http://c2.com/cgi/wiki?PairProgrammingDoubts http://www.c2.com/cgi/wiki?PairProgrammingMisconceptions
and for testimonials see http://www.c2.com/cgi/wiki?ProgrammingInPairsTestimonials .

bignose on November 19, 2007 1:26 AM

Got to jump in again here. Can those of you claiming that pair programming is double the number of hours per project clarify whether you've ever worked in an established pair programming environment? My experience is that the overhead is far smaller than that because productivity stays higher - there's less wandering off down blind alleys, losing focus, banging your head against a brick wall working out how to do something. As has been observed already, code quality is much higher, so there's also less tidying up after the event.

David M on November 19, 2007 1:28 AM

Back at my old office there was nothing like a code review; the developers were like super-heroes and the development style hero-driven development.
Here, I sometimes get it reviewed by my team lead, VOLUNTARILY, because i think that at least he should know what was done for waht reason. He actually thinks that I am good enough not to make a mistake! Anyway, code reviews when wetried doing them went pretty well, but the overall impact of that on us was not that significant.

Jehanzeb on November 19, 2007 1:28 AM

I'm still a student in CS, but I've already had experience in both of these team practices.

From my experience, it's much easier to get read and understand code as it's being written than once everything is already laid out. Pair Programming was the way we were able to get the tougher projects out fast and done right. Whenever one of us hit a wall, the other had something to continue moving forward.

Peer Review was a tool we opted for much less as once the many lines of code were laid out, it wasn't as easy to simply look at it and know what the other had done and why, at least on the spot. Also, we risked keeping erroneous solutions in our heads for too long before they were corrected, something invaluable when time came for exams.

In my opinion, Pair Programming allowed up to produce and understand lots of good code and understand exactly what we were doing in record time.

Jon on November 19, 2007 1:40 AM

I find that "everyone doing everything" works great in combination with "guru watches the n00b" technique. It sure keeps the quality of the product up - but the core value for such approach is teaching developers new tricks. Everyone gets to learn C#, ASP, SQL, X-stuff, JS...
It's easier to get up in the morning and go to work if you know you'll learn something new today!

Goran on November 19, 2007 1:42 AM

The most effective pair programming that I have done in my past is that once everyone has done what they can do alone, and they need help with a very complicated debugging, or a very tough problem, then for a few hours we would pair up and go at the problem together and even tag team other people into the programming sometimes too.

But the most difficult problems and debug sessions I've seen were multi day single sessions where someone only stopped to eat and sleep and they did it alone.

I am a big believer in code review after the fact because it forces everyone to explain to others on the team what they have done and why and to clean up the code. We also discovered a lot of errors in the code review that would have gotten put into production otherwise.

You can still do code reviews even with pair programming. It's not an either or choice.

Jimmy the Geek on November 19, 2007 1:44 AM

As the only developer in my shop, I don't have the luxury of either, so instead I have an imaginary friend - she keeps me honest, and ensures I don't do those short-cuts and hacks which might be quicker now, but will cause me pain down the line. Plus she's really hot, which is always a bonus.

/ Might have been doing this too long ;)

Syd on November 19, 2007 1:54 AM

I'm not "against" pair/extreme programming. As Jeff mentioned, it's not bad in small doses. But after 6 hours it begins to feel like you're in the back seat in a long, long drive and you end up forgetting where you're going. I think I'd just prefer code reviews over pair programming.

Speaking of code reviews, I was once in an hour long code review (more like a standards review) that had the sole argument of code formatting of class properties...

```
public String MyProperty
{
get
{
```

```
return _myProperty;
}
set
{
_myProperty = value;
}
}
```

verses...

```
public String MyProperty
{
get { return _myProperty; }
set { _myProperty = value; }
}
```

Yeah, that's when I knew I wanted a new job.

Steven Rogers on November 19, 2007 2:10 AM

I can see educational value of prolonged pair programming sessions. However, I find full-time pair programming counter-productive and damaging to software quality because the programmers will disrupt each others' flow (http://en.wikipedia.org/wiki/Flow_%28psychology%29). So, in my view Code Reviews is a more suitable tool for improving quality of software.

Maksym Korotkiy on November 19, 2007 2:12 AM

I have some experence with pair programming I think it is important the 2 programmers are of a similar experience level and probably most importantly get on.

I remember when I first started as a programmer I was paired with someone with about 15 years experience so alot of the time he was explaining to me what or why he was doing something (and doing most of the typing). Perhaps the section of the program was a bit too complex for a relatively new programmer and it would have been better to pair me at that point with someone else relatively inexperienced on a simpler section so we would learn from each other not more of a one way street.

Still I am very grateful for what I did learn but I think if the programmers are at such different levels a code review is better as a newbie will probably read the code in their spare time to improve their skills.

Pete on November 19, 2007 2:18 AM

I should have clarified: yes, the increased productivity is the advantage (obviously). However, to start out, you still pay two people for one programming job. Hence, double the cost. The question is, is the production increase really high enough (100%) to cover paying two programmers to do one programmers job ?

And no, I never tried anything like that, I'm just curious.

J. Stoever on November 19, 2007 2:38 AM

I would freak out if someone would watch me every the time I code (and also has a keyboard to interupt me lol)

jan.g on November 19, 2007 2:49 AM

On a previous job we introduced absolute and relative reviews.
Absolute review: an official code review, usually after very big changes or for new modules from scratch. Like a document cannot get status 1.0 without a review so can't code.
Relative review: for small changes a developer only needs to have his code checked by one other developer before committing. So if you solve a bug with minor code changes, you ask a fellow dev to check your code. This developer should be absolutely convinced what you did is correct. If there is doubt, the architect is called.
Relative review is of course only possible on code that has had an absolute review. A task based CM also helps a lot for this.

Advantages:
- easier to introduce as pair programming
- dev cannot check in as easily at nine o'clock in the evening :-)
- devs know each others code better

Disadvantages:
- slows devs down (but this is also an advantage, see above)

- ?

I've been in a few agile projects, but in my experience real agile (pair programming, user stories, ...) is very difficult and very easy to fail. And when it fails it will be much slower and harder than with the traditional methods. And there is always someone willing to prove you wrong...

I've also noticed many customers were not fans of code reviews, but if they call it code reading, it's usually a good indicator they are doing it wrong ;-)

ChrisVB on November 19, 2007 2:58 AM

I used to work for a fairly large software firm, where in theory we had to have code reviews by the group leader before check-in. In practice, only fresh hires' code was checked for the first couple of weeks. The reason: the codebase for the software we worked on was very large, old and of low quality, so most of the changes we made were "not pretty", and people didn't want to look at other people's unfortunate projects ("You have to add that feature? Ha! It can't be done."). At our particular group, an additional factor was that the group leader was an ass ("Let's think in terms of complex solutions"), and asking his opinion about anything usually made matters worse (since then you had to do what he said). There was a fairly experienced programmer on the team, so I would periodically ask him to review what I was doing (despite the looks the group leader gave us) and that really helped. He always had great insights due to his experience, and this is the way I learned the most. The guy & I ended up quitting and went over to another company, where we still hold these discussions, now on more equal terms.

Anonymous on November 19, 2007 3:00 AM

My understanding is that PP is seldom a whole-day activity. Nor, in some cases, are pairs static - I don't know if they're apocryphal, but there are stories of pairs switching once an hour, at the ding of a timer.

And it's not just about code quality, spotting problems and the like: it also promotes broad knowledge of the code base throughout the team, which reduces the likelihood of duplication/deviation and increases the chance of highly cohesive and lowly coupled solutions.

I strongly suspect that, like many XP practices, it works best when applied with others. Small increments of development (stories), TDD and continuous integration would seem to

be pretty essential for a successful long-term application of PP.

I live in hope that I may one day be able to find a job that lets me experience XP properly without needing to take a cut in my overly comfortable salary....

Mike Woodhouse on November 19, 2007 3:03 AM

Some of my best work has been sitting right beside someone and working through a project. Having another person there, especially at the start of a relatively complex project, 100% of the time really does increase productivity - that extra pair of eyes picks up small things (typos, mismatched brackets) which allows the driver to focus on the larger problem, as well as being another person to bounce ideas off to find any problems much faster and save you from a lot of dead ends. I recommend it fully.

nickf on November 19, 2007 3:45 AM

Pair programming rocks!!

My senior and I pair program a few projects a year. It really, really, works. We both have two completely different programming styles but when combined they really work.

Caveat, pair programming doesn't work for all projects and all scenarios. Pair programming is great for small projects and/or utility type software.

--mis dos centavos

whocares on November 19, 2007 4:45 AM

My experience of pair-programming(not on a professional basis) is that I'm much less likely to put in a quick hack just to get something working right now because someone else is watching and bad code is embarrassing.

Jesse McNelis on November 19, 2007 4:57 AM

I am taking an intro to java programming class to fufill a requirement for school. The professor has us program in groups of three every class. One person drives and the other two pretty much dictate the code.

Although it feels like running with weighted shoes at times I have been able to quickly pickup some quick tips, shortcut keys and other little tricks that I would have never known about.

Also when I am back to programming all by myself it feels like I have removed the weights from my shoes. I can run that much faster after practicing with the weights.

bryan arendt on November 19, 2007 5:02 AM

I recently started at a pretty cool company who develop and sell Crucible - a pretty cool peer code review tool. We use this in conjunction with pair programming daily in our product team (another product with ~ 10 developers) and I gotta say that even though I'm new to the whole process I think it really works.

From my experience, pair programming is a great way to bring a new developer up to speed with the code base, amongst the other reasons already cited. We don't pair on everything though - some tasks just aren't suited to pairing (e.g. writing tech specs, some maintenance and bug fixing). We also found that pairing doesn't work for small teams so well (depending on your product, of course) as you just can't allocate enough resources to spiking new functionality.

The code reviews come in on top of the pair programming. When a pair is working on a piece of new functionality, our team lead will usually delegate someone to review their commits using Crucible on a regular basis (or once at the end, which I find doesn't work as well). This allows the pair to get some much needed grounding, as spending too much time pairing can sometimes make you feel invincible and you might start to miss tiny things. However, when you combine pair programming + code reviews + continuous integration + a full test suite, you're running a pretty tight ship!

If you want to check out Crucible, visit the site: http://www.atlassian.com/software/crucible/

Michael Tokar on November 19, 2007 5:14 AM

Way back in 1989, I worked for a manager who decided pair programming was worth a

shot. We did it for about a year and a half. I was surprised to find that it was nearly as time-efficient as solo-programming, mostly because we'd overcome obstacles faster. I have a tendency to want to really chew on a problem before asking for help. Sometimes, that 'chewing' transmutes into distraction - surfing the net, daydreaming, etc. In a paired environment, the moment a roadblock shows up is usually the moment the navigator says 'here, let me drive', and he takes a mighty swing at the problem.

I'm sure that interpersonal chemistry is a critical ingredient to successful pair programming. We were all lucky enough to be about the same skill level and more-or-less egoless. I've worked jobs with a few prima donnas on the team who I'm sure wouldn't cope well with pair programming (though, if one paired two prima donnas together, they'd at least stop disrupting everybody else! :)

At a later job, we were paired intentionally lopsided, so the old hands could mentor the noobs. I found the best way to work was for the noob to become a 'voice-activated typewriter'. I'd dictate code, and anytime I did something curious, the noob would ask why and learning would ensue. I'd also frame problems we were about to code and ask the noob how he'd go about it. It wasn't the fastest way to code, but having an audience did make me more fastidious, and I'm sure it was the fastest way to transmit coding craftsmanship. A good investment.

The one downside of pair-programming for me was the ceaseless nature of it. Just as Brooks points out a fivefold variation in programmer productivity, there is at least a fivefold variation in my own productivity from day to day. In a paired situation, the person who was more on the ball would usually drive, but there was still a nagging feeling that the less on-the-ball person should get with the program and he didn't have the luxury to just stare at the screen all day. Good for the schedule, but hard on morale. Sometimes my partner and I would agree to take separate problems and go solo for a few days, just to get back into the groove.

On the whole though, I'd recommend pair programming, provided the chemistry works between the participants. I believe it produces higher quality code at about the same rate as solo programming, and it ensures a lot of cross-pollination in the team. It also helps members accept the chosen coding standards of the team, some of which may be foreign to their natural style.

Greg on November 19, 2007 6:00 AM

Excellent report on how pairing was used at Silver Platter Software:

http://svn.arlim.org/arlo_papers/Promiscuous%20pairing/Agile%202005/paper.doc

Bill Christie on November 19, 2007 6:23 AM

I've done a bit of pair programming off and on at work here, and all I can say is that there is a lot of code that just plain doesn't deserve it.

Anything complicated or interesting, where having a second person checking your work stopping you from staring at a screen for an hour figuring something out - it's amazing. It's awesome.

But for the boilerplate setters and getters and obvious implementations - seriously, the guy writing the code is bored enough already. The guy watching him goes back and forth between half asleep and "oh, you missed a semi-colon."

The best process I've found is having a small team working on the project in the same room, programming on their own for the trivial implementation, and brainstorming or pair programming for any of the trickier parts of the code. There are a million times where you don't really need someone watching you constantly, but you could use a "hey, quick check - in what order do I need to lock these objects so I don't get a deadlock?" without having to wander down the hall to their office or wait on IM.

Now, that said, I think we need more overall code reviews here, just because it's always better to have more people looking at your code (and to some degree, because I and my normal partner have similar weaknesses in our coding style and knowledge). But I can say that I can't imagine continuous pair programming to be actually productive - in over half of the code it's more like a union construction job - one guy working, one guy standing there leaning on a shovel.

Mike on November 19, 2007 6:25 AM

I personally use almost exclusively code reviews rather than pair programming. The main reason I think is that management buy-in is almost a necessity for full-time pair programming. But I too have doubts about doing pair programming a full eight hours a day.

Both pair programming and code reviews can be ineffective if people just go through the motions. You know this has happened when you just get those 'general' comments and nothing specific.

As a senior developer I do a lot of reviews, and have been thinking lately about how to make them as effective as possible. It starts with believing in the benefits of reviews, but how you tackle the review also makes a big difference. I recently wrote an article that goes into more details:

http://www.basilv.com/psd/blog/2007/strategies-for-effective-code-reviews

Basil Vandegriend on November 19, 2007 6:28 AM

Paired programming can be very effective with folks with the same skill level, unfortunately, the trend I see as companies adopt paired programming and Agile style development is the it tends to average the skill levels down. For example, were we to pair a level 2 developer with a level 5 developer, we see about a 3.5 skill level's quality of work. This is obviously a big win for the 2 but not so much for the 5.

And as far as peer programming instead of code-reviews, I have not seen an implementation of paired programming that resulted in less system testing effort, and in fact have found the quality of the code to have dropped due to rubber stamping your buddy that "knowns what he is doing"

But if done right, I imagine it's fine, my question is, whatever happened to teamwork and communication. Companies seem to go to these development methodologies to rectify the fact that they have terrible inter-group communication.

Just my $.02

Jminadeo on November 19, 2007 6:29 AM

Good article. From what I've observed, you have the balance between pair programming and peer review right. As I see it, pair programming is simply a special case (a particular

style) of peer review. That doesn't make pair programming better or worse than, say, formal inspection, just different.

Everyone has an opinion, but where's the science? I haven't followed the literature on peer review closely. It strikes me as a discipline that's amenable to scientific method. Has anyone done rigorous experiments that show, under controlled conditions, which review techniques are the most effective?

Dennis Linnell on November 19, 2007 6:30 AM

Excellent overview of the two major approaches to getting more eyeballs on things. I've done a lot of code review at multiple companies and see it as invaluable (if sometimes tedious). I've done some pairing and while I like it, I wouldn't want to do it all the time.

What I'm struggling with now is how to do either of these in a distributed team. I'm currently leading a team with people in four places and I myself have no one co-located.

We've started to experiment with code reviews and pairing over screen/app-sharing programs and so far it's been just ok. Still learning. Any tips from anyone?

Alex Miller on November 19, 2007 6:46 AM

I did a summer internship while in college, and we did pair programming there. I was paired with one of the other interns, and we worked together for 8 hours a day for 3 months. That was the idea anyways. It turned out that the guy I was partnered up with knew almost nothing about programming (don't ask me how he got the internship), and so I ended up writing the entire thing myself.

Pair programming might work well if you have 2 competent programmers that work well together, but this has been my only pair programming experience, so I wouldn't know.

Brandon on November 19, 2007 6:58 AM

I think some sort of compromise is needed between these two positions. I can't imagine coding with someone looking over my shoulder and making comments. Nor, can I imagine suddenly releasing my coding in the middle of a program and handing it to someone else. I've never tried that, so maybe it does work.

On the other hand, code reviews seem to take place after the damage is already done. Someone spent 24 hours coding, and by the time they finish, it's too late to really do anything about it. As one person told me, code reviews are like a group of people looking over a deadly traffic accident and critiquing what the drivers should have done.

I think pairing developers is a great idea, and having them involve in overseeing code as it develops is a great idea, but tying down two very expensive resources at a single terminal is going to be a lot for management to swallow.

Maybe a compromise: Pair developers, have them discuss their tasks, and how they will manage their tasks at the beginning of the day. Then, at the end of the day, review each other's code. It will take some firm scheduling (9am all developer pairs meet with their partners to go over their day's activities. 4pm: all developer pairs meet and review each other's work.) But, it will allow both developers to work separately while at the same time understand exactly what their partner is doing and how they plan on doing it. It would also allow the code review at the end to go a bit smoother since the partner will understand exactly what their co-developer was up to and why they wanted to do it that way.

David on November 19, 2007 7:16 AM

Jminadeo you hit it on the head.

A lot of hipe to replace proper work habits, ethics and communication.

When you write poor code, you know it, do it right the first time, as you should. There is NO excuse for hack code, if you don't have enough time to do it right the first time, what makes you think you will have enought time to fix it when your hack code breaks!

Get back to basics people, Proper Work Ethic, Good Solid Code, Communications, that will eliminate all of this foo-foo crap.

Do your work and be responsible for what you produce!

Eric on November 19, 2007 7:24 AM

I admit that my 24 years of embedded programming experience have left me very cynical towards this idea. The first thing that bothers me is I have an introverted personality. Doing the job is far more important to me than talking about it. I can't talk through what I'm doing! If

it is a difficult project I depend a lot on refactoring - totally blow yesterday's code away and start over. Having anyone looking over my shoulder would be very boring for them and it would slow me down to have to explain.

I worked at a place that was going to try this. They wanted to pair a junior programmer with a senior programmer (me) and have the junior programmer at the keyboard. This is a good way to teach but not a good way to get anything done!

If Management is involved in this process it becomes too cumbersome to use.

Big Dave on November 19, 2007 7:29 AM

Has anyone tried any of this in a distributed environment? At my company, we all work from home. There are program like SubEthaEdit on the mac to facilitate file collaboration over a network.

With our new code base, everything is in an svn repository where all the commits send emails to everyone. Its made a big difference, as everyone can see and have some familiarity with the code, and catch possible issues by replying to the commit email with your concerns. Usually whatever is off then gets fixed and committed pretty quickly after the discussion of the issue is done.

This works well, but I do think there are situations where more transparency to the actual writing of code would really help, especially in cases where code one person writes is immediately needed by the other to finish their work. We mostly all work on the same server, and can open our working directories for read access by the group, but that can be a bit clunky.

Joseph Annino on November 19, 2007 7:38 AM

Personally, I think that pair programming simply solves common problems that could be solved easier in other ways. Yes, I can pound in a nail with a sledgehammer. But it is often easier to use a carpenter's hammer instead.

Matt on November 19, 2007 7:47 AM

pair programming is a great concept, but does not work for most. So far as i have seen expert coders are introverts (with due respect for their skills). moreover it is very difficult to

find a pair that can coexist and contribute, just like a good marraige! now i hope you know how complex this is !

maruti j on November 19, 2007 7:47 AM

Rather than anecdotal experience reports, why not look to the research community? For example, a large quantitative study of pair-programming in Norway: http://catenary.wordpress.com/2007/03/12/pair-programming-evaluated/ (not my site).

I realize it's sexy to ignore academic results, but occasionally we have something to contribute.

Neil on November 19, 2007 7:55 AM

Hey Now Jeff,
I always enjoy code reviews, I learn many things to improve my code.
Coding Horror fan,
Catto

Catto on November 19, 2007 7:58 AM

One important thing to remember with pair programming is that it tends to be relatively noisy, at least compared to one developer working alone. It doesn't work quite as well if you've got your developers sitting out in a cube farm.

Joel Coehoorn on November 19, 2007 8:03 AM

In my first programming gig, I was awe-inspiringly green. I was hired as an intern, then due to the demands of their staffing requirements, I got promoted a week later to mid-level programmer. The good news is that the company was a firm believer in mentoring. My mentor became the Director of Development. My job entailed maintaining a production program in VB6. If I got stuck I was to go to the Director and talk things through with him (not any of the other developers... there's a business rules reason for this, but it's irrelevant to the story).

Wouldn't you know it, but about a few weeks after, I got a task that the Director's exact

words were "normally I'd ask you to spin your wheels on this for an hour, but since I think I'd spin my wheels on this... why don't we just sit down and you can look over my shoulder."

Long story short, the Director did the coding. I'm familiar with VB6 syntax (Director programmed in VB6/.NET, C#, Python, etc. etc. etc.) He was impressed with the "paired programming" thing. I'd catch syntax errors and he'd start coding in C# (his primary language) and I'd say something like "wrong language".

Of course, since it was designed to be a learning opportunity, I was the one who commited to CVS all the code... after I could tell my boss what it did and why in plain english. I'd say I learned a good bit in those few days (and the selling point is that my boss estimated it would take a week for the both of us, turned out taking only like 3 days).

Of Course, YMMV, IMHO, "past performance is not indicative of future results", etc.

wes on November 19, 2007 8:16 AM

This doesn't just work for programming. I manage the Internet group for a regional CLEC/ISP. When I got here I instituted change review for everything. My network people don't make a routing, ACL, VLAN or IP change without review by at least one peer. The server team reviews each others configuration changes to services. This has stopped the knocking out whole cities worth of Internet access accidentally.

Finn Cool on November 19, 2007 8:27 AM

I'd love to try pair programming or peer review... unfortunately, our project leader is unlikely to agree. He himself modifies developers' code without telling anyone (once, a change of his caused the loss of 95% of a database).

sam.k on November 19, 2007 8:42 AM

I'm surprised no one has explicitly brought up this point (although a few have touched on it). I can't stand having people look over my shoulder while I'm working. The anxiety can get intense enough that I would just want to leave work early. I became a programmer partly because I like working alone. It sucks that the typical introverted computer geek as professional programmer may become a thing of the past.

KG on November 19, 2007 9:05 AM

Code reviews are a useful tool if they're thorough. Most teams only dedicate an hour or two per week for this, so they don't eat up a lot of resources. Of course, sloppy code reviews are worse than no code reviews at all, but that's just common sense. If you have a good senior developer who cares enough to delve into the non-trivial modules, they are a generally good practice for most teams.

Pair programming, like many things XP/Agile, is wonderful according to its practitioners and testimonials, but doesn't do so well in the realm of hard data. It's a useful technique to aid a difficult debugging session (especially anything multithreaded), and also useful as a short-term training tool (I'm talking about maybe 1-2 weeks, tops). As a constant regimen, however, it's monumentally inefficient and generally counterproductive.

If you take any pride in your craft, then please remember that these things are merely tools, not panaceas, and should never be hard-coded into your team's development process. Don't fall for the XP snake-oil; just because something a small dose of something is good for you, does not mean a megadose will be.

Aaron G on November 19, 2007 9:21 AM

"I can't help wondering if pair programming is nothing more than code review on steroids."

@Jeff: I've heard Peter Provost and Brad Wilson refer to pair programming as "Real Time Code Review". Where I work, we have called it that as well, from time to time, to remind ourselves of some of the additional advantages of pairing.

@Joseph Annino: yes, there are teams that do this (various technologies to assist). I can't remember anyone specifically off the top of my head, but I have read of people doing this and using certain tools to make it work.

As to Pair Programming: our team has done pair programming a couple different ways. I think Peter Provost's "Pair Programming Ping-Pong" is the most effective way to make pair programming work: http://sean-carley.blogspot.com/2006/04/ping-pong-pair-programming.html

With a ping-pong methodology (TDD), both parties tend to stay engaged in the process a lot better. Focus is maintained and you get higher quality code. The keyboard switches hands every couple of minutes. Design decisions are tackled in tandem; better ideas

emerge.

Pairing is more intense. Solo developers probably don't realize how much time they spend doing things other than writing code, which tends to break up their day. What we've found with pairing is that the pair of developers needs to know how to manage their time: they need to take breaks, do spikes, research something, etc. They need time to fly solo to break up the intensity of the day.

Chris Holmes on November 19, 2007 9:28 AM

It's also pretty amazing how well READING YOUR OWN CODE works for finding bugs... vs typing code like a monkey until it compiles.

engtech on November 19, 2007 9:39 AM

This sounds eerily familiar:

>One of the problems seems to be that nobody wants to spend the time to
>eally understand new code that does anything non-trivial, so the
>feedback is usually very general. But later, when someone is working on
>the code to either add functionality or fix bugs, they usually have
>lots of feedback

because that is exactly what happens when writers ask developers to do a tech review on documentation. No one has time to sit down during the review period and think through what the text actually says (and doesn't say). With rare exceptions, the best you're likely to get is comparatively trivial comments on code samples. (Not, say, a comment that suggests an entirely new sample.) But oh boy, once you've actually published documentation that was only cursorily tech-reviewed, and that then proves to be incorrect or incomplete or lame, _then_ people have opinions about what the docs should say.

I would say that unless it's a central focus of your work -- whether you're a developer, editor, or whatever -- reviewing other people's work is never going to get enough priority, and will never be done in depth. Pair programming puts review front and center, which is probably why it's more effective than code reviews.

mike on November 19, 2007 9:56 AM

For pair programming to succeed, I think there is a significant dependency are pairing the right people together.

Lennon + McCartney = inventive pop music
Lennon + Starr = ???

When a company almost randomly assigns two people to pair program together, I don't think expectations should be particularly high.

gunther on November 19, 2007 10:23 AM

My experiences with pair programming have been fantastic. Incredible focus and productivity, better code, reduced knowledge silos. I'm a big fan. Far more effective than code reviews.

Kevin Dente on November 19, 2007 11:53 AM

I'm fine with pair programming, except for the people-sharing-the-keyboards-and-mice part. Germs, man. And grime.

John on November 19, 2007 12:32 PM

I have worked in pair-programming situations that have worked as well as those that haven't.

When it works, it works very well. My best experience was working with someone who was about the same skill level, but who had been on the project much longer. It was always a very enjoyable and motivating situation, and we got a lot done and did it well.

But the times it just didn't work were when I was paired with low-skill programmers who didn't care. When they sat in the reviewers seat, they would sometimes even fall asleep! Pair-programming is worse than useless in that situation.

If you can find someone you can pair well with, it's worth it. Do it.

As for code reviews, I haven't ever seen them work well.

Terhorst on November 19, 2007 1:00 PM

My first experience pair programming was sitting down with someone and figuring out how to script Active Directory actions. This was ~2001 or 2002 and we had a devil of a time working on it individually. MSs docs were not always helpful. The technology was still new to the IT department. What we did in one day would have take 3-5 days individually. For risky and experimental code, I believe pair programming can save a great deal of time. After that day, both of us knew the code inside and out including many of the variations that had proven not to work.

Stephen on November 19, 2007 1:17 PM

For most simple programming problems, especially ones I've solved before, I think working with another person can be pretty inefficient.

But when working on particularly difficult problems, two heads are a lot better than one. If one approach doesn't work, then you have another brain to suggest alternatives.

Chris Papadopoulos on November 19, 2007 2:00 PM

I've noticed a lot of comments saying things like, "it's nice for a while, but you can't do it full-time every day" or "it works for the really hard problems, but not so well for day-to-day stuff." So I guess the real question is how do you create an environment where you can do pair programming part of time, and work on just the things during that time that lend themselves to pair programming. How do you identify those items? From what I've heard, places tend to be either nearly full-time pair programming or pair programming only by accident, with almost no middle ground. Anyone have experience working in a shop that does does pair programming part-time?

Joel Coehoorn on November 19, 2007 2:58 PM

I think for the more complicated parts or "cutting edge" parts of a project pair programming can be great. For simpler, more run-of-the-mill software, it isn't necessary and can be costly. I've found some personalities are not compatible with it.

I think pair programming is a great idea when used appropriately.

As with any good idea, it all goes wrong when a large project or company adopts it across the board and forces it in places where it is not appropriate. Yep, I'm a corporate monkey...

Spide on November 19, 2007 3:18 PM

I think gunther is particularly on the mark here in suggesting that the people in question are going to make much of the difference. There are some people I know who you in fact COULDN'T pay me to work with—they're either poor collaborators or are on such a different wavelength that there's no point in us trying to see things eye-to-eye (I'm skeptical that there is a "universal language of code"—as with any language, there are lots of ways to say that you are skinning a cat, to pervert a phrase). However there are some collaborators with whom work would come quite naturally and easily.

And as usual of course I'm suspicious of anything that labels itself as one-size-fits-all. Perhaps its the fact that I have apparently a strangely-shaped body (it's impossible to find pants that are my size, which isn't extreme in any respects but must lie at the ends of a couple bell curves), but I've rarely found myself within the "all," and I suspect there are a lot of other people who are quite similar.

Shmork on November 19, 2007 3:21 PM

My 2c worth on the cost side. Yes, having two people working on the same thing intuitively doubles the cost, since they could be getting two different things done in parallel.

But it's not that simple. For starters, a pair tends to actually be working almost full time while together, since it discourages both from slacking off and web-surfing while waiting for builds as a single developer may be prone to doing (speaking from experience). Quality is generally better than code that's simply been reviewed, since in my experience, after-the-fact reviews tend to be somewhat superficial and lacking the understanding of why decisions were made.

And it's good for training and knowledge sharing - when one person works start to finish on

a job, they frequently end up as the only person who knows enough to maintain it. With pairing, at least two people know it, more if the pair are regularly being rotated (i.e A and B for a few days, then C replaces A, then D replaces B, etc).

In any case, those are the reasons my workplace does pair programming - we find that the benefits outweigh the cost of not having so much work being done in parallel. It may hurt a little short-term, but we find it's much more sustainable over the long-term.

Simon on November 19, 2007 3:30 PM

When I first read Kent Beck's book on XP back in the late 90s I could be heard muttering, "he's right", ... "he's right", ... "he's right". However, as soon as I hit the bit on Pair-Programming it just felt wrong to me. My objections are selfish but here they are:

: I do not want a mosquito muttering in my ear as I write code
: I want to think at my own pace so that I can understand what I am doing
: I do not want someone else to lean over my shoulder as I work

Basically it is all selfish stuff about how I work best. For me best is a low interrupt rate and time to think about what I am doing. PP gets in the way of this by disrupting the way I think and not allowing me to soak an idea up until I can understand it.

I have done some PP work and the tasks I found it really suitable are exploratory ones where no-one has an idea about how to do something. However, the same effect could be achieved by jaw-jawing in front of a black/whiteboard.

I have worked with someone that believes the complete opposite of what I have written above. We get on well but we do not agree on this point. Having said that, he is great to work with on PP.

I work with someone else that is totally awful at PP. He slinks off 15m into a PP session to do "something more useful". He thinks it a waste of time and does not participate at all. Of course he is a bit of an AA [http://www.joelonsoftware.com/articles/fog0000000018.html] and appears to think of pairing as beneath him.

For PP to work you need co-workers to agree to do it and you also need to believe that it is a good idea.

Frankly, I think PP is a bad idea for complicated tasks for which someone needs to think a lot. Individuals do not think at the same rate, nor do they usually work the same way. PP

does not take individuality into account.

Having said that, PP can be a good way of brainstorming awkward tasks.

The trick is to choose the correct tool (i.e. PP, Whiteboarding, solo programming) for the task at hand.

Experience has taught me that SP combined with whiteboarding often works out better for __most__ things. YMMV.

Gordon J Milne on November 19, 2007 3:39 PM

Continuing my previous post, the main downsides are that a) it's hard to sell to managers, and b) it's not a style that suits everyone.

We were lucky on the former, in that a new project was started under a pro-agile manager, allowing the team to organise under it's own terms. As for the latter, we've found that it does seem to work for the majority of people, even those who didn't think much of the idea before they joined us. The small minority who don't tend to be people who simply don't work well in a team environment in general, never mind paired with someone else. Not much you can do about that.

(Speaking fairly generally in both these posts - obviously there are exceptions to any rule).

Simon on November 19, 2007 3:40 PM

I have some questions about people's experiences on pair programming.

1. In my time, I've noticed that some programmers are really just gifted and much better than their peers. If you mix one of these with a mediocre programmer, how well does pair programming work. Or, should one perhaps only match up programmers of comperable cognative ability. And, I really do mean cognative ability, not necessarily their current skill level.

2. How big does an organization have to be for pair programing to work effectively? It obviously won't work in a 2 person company (or perhaps not even in a 10 for that matter).

3. Code reviews (for the bean counters who hire) seem more efficient on a per dollar basis. Has anyone managed to get cost savings by doing pair programming?

This is a col topic and one where the results cannot be easily measured. The results may be, in fact, very chaotic due to people's varying abilities. But then, I suppose that since even Alan Greenspan cannot predict how people in the economy react through the best computer models (as he said in one of his interviews), we have little chance here.

Cheers.

Carleton on November 19, 2007 3:57 PM

This was an interesting read. Thanks!

Steven Klassen on November 19, 2007 4:20 PM

Carleton -

1. No doubt, the best work comes from pairing people with equal skills. That said, as long as the difference betwen the two isn't too great, there's good training benefit in mixing junior and senior. It's more work for both though, since they need to find a compromise that doesn't leave the junior lost or the senior bored.

2. Our team has varied in size over time, but let's say between 10 and 20 developers of varied experience (i.e 5-10 pairs). Ideally you want enough people that you can rotate people around, not having the same pairs working together for months. Down that path lies madness - working that closely with a single person doesn't work well long term.

3. The actual $ cost is a little out of my domain, so I can't give a specific answer. But I'd not be surprised if it came out well - it's my experience that code reviews are too superficial to make much difference to code quality, so teams relying on that have always spent almost as much time fixing defects as writing new code (and defects). In a pair-programming team, we certainly still get defects, but not so much - maybe a tenth of our time spent on that kind of activity?

Simon on November 19, 2007 8:55 PM

I think there's one other advantage to pair programming that hasn't been touched on much here: it keeps people from goofing off. Face it, the internet is a huge temptation. Hell, I'm at work right now :-)

I love doing pair programming because I want to learn all the tricks that other people might know. People who are too introverted to handle it should not be working in team environments anyway. And people who think they are too good to be working with someone else should be shown the door.

schnitzi on November 19, 2007 9:22 PM

Another helpful post, Jeff. Would you please fix the behavior of the "Read Older Entries >>" link at the bottom of your blog? When I click on this, it takes me to a _specific_ older blog entry. It should take me to a previous _page_ of blog entries.

Damon Cutler on November 19, 2007 9:34 PM

When I review code, I am looking for:
Excessive callbacks to the databases and/or poorly formed queries (or poor performing)
Potential memory leaks.
Code not written in the correct Tier(Viewer, Business, DAO)

I rarely have time to check the business logic. So, I just look for things that could bring the system to a halt.

rekounas on November 19, 2007 9:54 PM

I don't see pair programming as an good option in many cases. The pair doesn't get along, the flow breaks all the time, too much talk noise is generated, management doesn't see the benefits, there are not enough resources for some projects/tasks so how can you put two people into every project/task, difficulties to create schedules who works with who, customers call and interrupt both programmers who are also maintainers in four other projects, etc...

And who reviews the code of the pair? We shouldn't be confident that the code is of course perfect, because hey there was _two people_ coding it. Some programming standards are still needed plus good architecture for the software and so on. And someone should make

sure (review) that the code goes according to the architecture.

In the end programming really should be so simple, that you don't need the help of a pair. Or you should team up with a person with other skills than you have to achieve some goal. But then that would be team work, not pair programming. Why would you use four people for a task of two?

Still I think that talking is good, because that way information is shared. But talking should occur so that you don't disturb other projects or interrupt others. For reviews the reviewer can prepare when he has time and the review session is kept in a separate room not in cubicles.

Don on November 20, 2007 1:18 AM

I don't really see how pair programming would help that much. Benefits such as "people understand the project better" and "there's more team communication" are difficult to quantify and seem more like pie in the sky arguments to me. Intuitively one might say that these things are good things, and it is very easy to falsely believe that this adequate justification for doing it; however that is not the question here. To take a reasoned approach to the problem, this question should be thought of in terms of expectation as it relates to productivity efficiency, and ultimately profit and cost.

The real question is, are these benefits MORE profitable in the long run than the traditional method of one programmer per computer? It's not good enough to say "well I think pair programming is more profitable because...". Show us the facts that prove your claim. Show us the quantitative evidence that proves pair programming is more profitable in the long run than single programming. It's not good enough for it to be "good", it has to be better than the alternative. Until you can show this, I remain skeptical.

As for the other arguments such as "you won't write a hack because someone is watching", nobody should be writing hacks anyway regardless of whether or not anyone is watching. Do you really need someone watching your every move in order to produce professional code? Does a manager have to sit around and babysit you so that you don't write crap software? For those to whom this applies: you are the programmers the rest of us dread. Stop being lazy and do it right the first time.

Dave G. on November 20, 2007 2:15 AM

I agree with others that some of the ideas behind pair programming sound nice, but until

someone is able to show some solid evidence that it works good in practice I am not ready to embrace it. I have a hunch that with pair programming in practice at least one person is wasting about 80% of their time, and that the constant communication between partners disrupts concentration of both. I'm not convinced that's good for productivity or quality.

I think code reviews are much less time consuming and much less intrusive and (as Jeff also argues) they already give huge benefits if performed correctly. Unfortunately, in practice there is rarely time reserved for code review, and what management cares about most is meeting the next deadline, not ensuring the quality of components that are already more or less functional. In such an environment, introducing pair programming while you are not comitted to performing code reviews properly doesn't seem right.

These considerations aside, I'm fairly sure I would go batshit fucking insane if I had to work that close with someone the entire day, every day!

Maks Verver on November 20, 2007 3:30 AM

Having two programmers at the same skill level working in pair would benefit to both code and programmers, but having two progrmamers at different skill levels can pontentialy damage the code, programmers and project in general.

There is tendency for the better programmer to take a lead of the whole project (same as if he was developing by himself), and put 'slower' progammer in more passive role (doco dude). Saying that, frustration can rise from either of two, each wodering why the hell they did pair up. This also raises the question of 'pairing two coders'. Do you just put them togehter, or you ask them first? Do you match their skill level or does that matter at all? etc..

All this wouldn't be a problem if we were not both humans and programmers (worst combo) each wanting to play significant role in history or/and the code. Other than that, its a great concept.

elijan sejic on November 20, 2007 4:42 AM

The point is you're never gonna redesign something at a code review, assuming it works. By having the review being a continuous step - the eventual code layout and design is much better. Especially if you're using TDD.

This is only my experience. YMMV.

JamesR on November 20, 2007 4:59 AM

Pair programming definitely wouldn't work for me. I'd end up choking whoever I was paired with. I usually put my headphones on when I code and block the world out. One of the reasons for this is my office is full of chatter and the other is that I normally code better and faster while I'm listening to loud music. I think it clears my mind or gives me a little dose of inspiration.

Brian K on November 20, 2007 5:12 AM

I'm with the other introverts here. I work best alone. I like nothing better than sitting down with a guru or mentor for an hour or so to learn something new, but all day? And if pair programming is anything like peer reviews I hate the idea. Peer reviews, in my experience, either turn into witch hunts, tickle parties or a forum for the office egomaniac.

PaulG. on November 20, 2007 5:47 AM

It seems the argument for either is feedback. There are thousands of studies out there that prove that timely feedback saves $$$. It's reflected in all the lean manufacturing methods, JIT inventory, etc. It seems logical (but may not be true), that the same can be extended to code. It also follows that the sooner the feedback comes, the more valuable it is.

I think the closest analogy that works for me is writing. I don't know many authors/editors that can share a typewriter. But the feedback loop is still adequate for the task. Whether per page, or per chapter, the author seeks out feedback as soon as possible and the reviewers actually drop everything and try to look hard at the problem. That's obviously the issue with code reviews, we have coders who don't have much invested in other people's code doing a crappy job of reviewing. Pair programming is one attempt to make the feedback loop tighter since we can't seem to make the reviewers do better. So two other alternatives spring to mind, 1) A editor type position who has final say on what goes out the door and who's job depends on the code being the highest quality (the single committer model), or 2) Increasing the number of reviewers and getting them to do qualitiy reviews. It seems the latter is a much harder thing to pull off (again pair programming tries to force the issue), because programmers look at a lot of tasks as beneath their consideration. We're artisans after all, we don't do administrivia and it's a bad use of dollars to force us into it. But artists do take pride in reviewing others' work, if only for their own inspiration. Anyway, it seems like team makeup is the best guideline for doing reviews, if you have one superstar who

you can spare from actual coding, you make that gal the grand pumba. If you have a team of superstars then you can probably do peer code reviews, presuming everyone has enough respect to do the job properly. Anywhere else you're left with trying to force crappy reviews or pair programming, basically relying on a process to fix your people. Pair programming seems like a better choice to me, if only because you might learn something.

Steve Jackson on November 20, 2007 7:26 AM

When my boss first proposed XP in 2001, I was horrified. One reason was that I just *knew* that pair programming was a bad idea. I was certain of it. The first few times, I had stage fright. It was hard. Things did not go well. My first pair, to this day, does not let me forget the body-slamming. It took me about 6 months to warm up to pairing, partly because it was hard to get over bad habits. If this sounds like a traumatic experience, that's because it was. Now I realize that pair programming demands more skill than I had initially reckoned, and depends a lot on who I'm pairing with, and what we're working on. Pair programming didn't turn out to be what I thought it was going to be, and when I finally went back to solo programming 3 years later, it was an eye-opener. It's like the old saying, "the purpose of a journey is to come back to the place where you started, and know it for the first time."

If you just throw 2 programmers together, there's no telling whether they'll pick up the right skills, or spin apart because of personality issues. Also, it's hardly worthwhile with cubicle furniture where you have to sit in a corner like a bad child -- you end up looking over the other programmer's shoulder. I hate that. I won't do it. Either ditch the cheap cubicle furniture, or don't bother trying to pair program. Side by side collaboration is great. Looking over shoulders is not.

If the two programmers are at different levels, it's automatically a teacher-student relationship. Most programmers are familiar with being a student, but few have any teaching skills. That's a real problem. Teaching skills are a subset of pair programming skills, and they're important.

These days, I usually prefer pair programming, but I look at it in a fairly agnostic way. It's like when a friend called me a "Mac zealot" because I bought a Mac, and I replied, "No, that's not the issue. The issue is that I know 3 operating systems, and you only know 2." Same thing: I can pair program, and I can solo program. Can you? There are many programmers that can only solo program, and aren't even aware that they're missing something, and will even denigrate me for knowing something that they don't. (Of course, they spin it a different way.)

I've been a professional programmer for over 15 years now, and my experiences with code reviews have been mixed, but I have yet to see a code review produce quality anywhere near as good as pair programming. It's just not in the same league. But maybe we just never did it right. I hope so. YMMV, as many have wisely observed already. Recently I met a programmer with 24 years of experience, who had never worked on a team, and that shocked me because my experience has been nearly the opposite. Most of my career has been on teams.

I'm currently on my 4th pair-programming team, and once again, I've learned something new. I previously thought that pairing made even-numbered teams preferable. But with 7 people, I find that having an odd man out keeps the pair swapping fluid. With an even number of pairs, there is always a problem with pairs locking up, not enough people wanting to rotate. Live and learn. "A man with a watch always knows what time it is. But a man with two watches is never quite sure." Probably after my next pairing team, I'll have an even more nuanced view of things.

As for productivity: I think that pairing is somewhat less productive in the short term, but more productive in the long term (on the scale of ~10 months, roughly, YMMV, standard disclaimers apply). This is because of the knowledge sharing when pairing; solo programming teams simply bog down sooner.

awh on November 20, 2007 7:56 AM

One of the interesting things McConnell says is this:
"The variety of errors people found was so great however that any combination of two methods (including having two independent groups using the same method) increased the total number of defects found by almost a factor of 2."
(This is on page 565 of the first edition of Code Complete.)

This is an amazing statement. Translated into specifics: if you have two independent code reviews, the set of bugs found is almost disjoint! (Please note that having two independent code reviews with small teams in each is completely different from having one review with a lot of people in it.)

This leads me to a refutation of Brooks Law, which states "Adding more people to a late project makes it later." A more useful law would be: "Adding more people to a late project can make it deliver earlier, if you add people only to the quality control portion of your development process."

Thoughts?

Fraser Orr on November 20, 2007 8:01 AM

I think pair programming might work if you have the right people. If you have two people with similar skill levels there probably are benefits. If you stick a skilled developer with a novice then obviously there are going to be problems in that the skilled coder will see it more as a teaching session and the novice will be pressured to keep up. I think code reviews are more practical but there is no substitute for hiring skilled people. I can't count how many projects I've been on with people that had no business being there. If you have people churning out crappy code a code review could turn into a code rewrite.

Brian K on November 20, 2007 8:12 AM

I was lucky at a previous job, where coincidently, our development team of 5 people consisted of 4 of whom went to the same college together (myself included). We generally worked in pairs on most projects, and was lucky to be teamed up with another programmer on the same level as me. We both learned from each other, bounced ideas off each other and had a deep understanding of the code we were writing. It worked for us because we got along..

However, later on...this approach seemed to turn into more of a starter/closer role. Someone would start a project, and I would end up finishing it. You all can imagine how frustrating that must be. :P

Kevin on November 20, 2007 9:03 AM

Pair Programming to the extreme can rule out it's benefits, I have been in environment where I got small doses of it and really liked it. At the same time I realized that I can't be productive with my 'Pair' for just about every type of work, say debugging.

Nirav Thaker on November 20, 2007 9:24 AM

> I can't help wondering if pair programming is nothing more than code review on steroids.

A bit more, yes. It's requirements review, design review, test case review, test results review ... on steroids. And since it can replace all those reviews, the cost is far less than double.

Further, the two partners can play different roles, with one buried in the code and the other considering higher level designs and implications.

So it's not two people doing one job, it's two people doing all the jobs. :-)

Jim on November 20, 2007 10:19 AM

Pair programming needs the right people.

One team I worked on had a rule that all coding had to be done in pairs, but the team was so full of ridiculously strong personalities that I had to battle verbally any time I wanted to get my hands on a keyboard. In this shop, being the Navigator meant hours of passive watching without any feedback as to what the person driving was thinking.

At another company, one developer kept sneaking off to program alone while everyone else was pairing on scheduled tasks. Eventually he developed some really cool stuff made the team's life easier and which he then of course got all the credit for. I found it humorous that management laid down a policy of pair-programming and then heaped rewards on the dev who paired the least.

These are (hopefully) rare and abnormal cases of course. But even in more normal experiences I have had with pair-programming, it has not been ideal. For example, I think it is the responsibility of the person coding to keep their pair-partner in sync with what they are doing at each moment. I have always done this assiduously when pairing, but I find it interrupts my train of thought to explain what I'm doing with each new piece of code.

Now I work in a job where we use most of the XP practices, but not pair programming. I find this ideal. I can't tell you what a relief it is to be able to sit and stare at a piece of code and really analyze it and think design decisions through without always having to keep mentally in sync with another person. I'm in heaven.

James on November 20, 2007 11:38 AM

I've never had the experience of 100% pair programming. Did something a little similar while working on projects in school, but it wasn't the same.

Code reviews? My one experience with that was a guy who basically wanted to dictate 100% of the formatting and nit-picking of little implementation details of the code.

Okay, sure, if I was the one in charge, I'd love to dictate that kind of stuff. Whatever my little preferences happened to be, I'd force others to comply, thereby possibly making their lives miserable or viewing me as a major annoyance.

But when you're just one senior guy (and not like say, the CEO/founder of the company), it's highly annoying to have to change, say:

forms.first.name[0] to

$('foo-form').name

Or whatever his arbitrary preference would have been.

Shanti Braford on November 20, 2007 10:04 PM

WOW!

We instituted code review before check in years ago and as long as it is done "appropriately" it is fantastic. Critical architectual issues get close attention, and boiler plate or trivial stuff is given less emphasis.

Peer Programming is great especially when dealing with knowledge transfer on a project. It definitely helps with cross training and generally does not slow things down very much, while allowing better coverage.

Honestly I think that a mixture of Peer Review and Peer Programming works the best depending on the work being done and the quality of your team.

Does this work for every development group?

This requires a very compatible team of high quality developers to be very productive, but obviously that helps in any environment.

We turn away lots of talented developers that won't play well with our team. This means that we don't hire developers without good communication skills, introverts who want to

hide by themselves, or have "issues" with ownership are primadonnas, etc...

If you have a lot of people who have ego issues and hate sharing their programming experience, there will be problems. Depending on your organization there are two ways to handle this. 1. You are a small organization or can't afford to lose these people and you cater to their whims to keep them happy. 2. You give them a chance to adapt with some help, and replace them if they don't work out.

To people who bemoan the "invasion" to their privacy and who want to sit alone in a quiet room without dealing with others all day, I think you are being very short sighted.

It is always hard to leave your "comfort zone", but don't write yourselves off. We are in a profession that has an information half life of about 18 months to 2 years. That means we are constantly dealing with change. Learning a new tool is great with short term tangible benefits, but actually changing yourself is forever. Put some effort into overcomming your personally social "issues" and your entire life will improve. I'm not saying it is easy. With a prevalent number of Asperger(ish) people in computer science, social graces are not natural for many developers, but most can learn and adapt if they are willing to try.

Agile advocate on November 24, 2007 7:00 PM

Pair programming levels the playing field. An idiot paired up with a super programmer, you get an average programmer.

My worst code ever written was done in pair programming. We both comprised our styles to produce a mismatch crap.

Ownership of code is good as far as the good old Perl virtue goes:

HUBRIS: Excessive pride, the sort of thing Zeus zaps you for. Also the quality that makes you write (and maintain) programs that other people won't want to say bad things about. Hence, the third great virtue of a programmer.

The only thing you need to do is hire good programmers.

10yearEx on November 25, 2007 11:19 PM

Orror, yup!
To me it's like to sit in two on the bog.

Really, when I am in programming mode I have to be alone in the deep of my mental cyber-world. How can I tolerate a nuisance like that? Not speaking of his stink, and worse! Orror, orror. :(

redcat on November 26, 2007 4:19 AM

There should be a place for all personality types to work. It's kind of crappy to take a profession populated heavily by introverts and force them out for the benefit of people who would do well socially in any job.

Erik on November 26, 2007 6:25 AM

There are places for introverts and people who don't like to work closely with others.

There will always be plenty of small companies with a very small number of "God @ All" developers with management that has no clue what is going on, but manage to trust the technosapiens making things work. These environments will continue to allow introverts who don't do as well socially to exist the way they want to.

There will also be many waterfall process shops with managers who assign tasks out to the worker bees in their cubes. For many that will also allow them the "privacy" they want. There are some types of projects that this actually makes sense for. Some are classified and such that upper management doesn't want many discussions of work happening.

There is also contract work if you want to work at home and have enough demand for your work. Of course this means having some business skills, but it does allow you to work how you want to work.

But at companies that are imbracing things like Agile process, developers NEED to communicate better and interact well socially. That does mean that developers in these environments need to have skills outside of technical abilities. These companies will hire those that can be productive in this environment, and let go of those that can't.

The bottom line is that if you can communicate well and interact well socially you will have far more opportunities for employment and be more valuable.

Agile advocate on November 26, 2007 2:13 PM

Arlo Belshee has done some experiments with "promiscuous pairing" and other methods to increase the efficiency of pair programming, see:

http://stabell.org/2007/07/13/arlo-agile-experiment/

Personally, we've used code reviews for the last couple of years, and have had fantastic results in terms of improved quality and cross-learning. We've experimented with pair programming on-and-off in many different ways, but haven't achieved the same efficiency that Arlo's team did.

Bjorn on November 28, 2007 6:39 PM

It's not "either/or" -- one does not replace the other. You should do both.

The advantage of pairing is that it makes code reviews far more effective. Instead of spending most of the review time explaining (and often defending) what you've done to the team (since the last review), you can focus on more critical things. Everyone will be far more familiar with ALL of the code, not just their part of it. You'll spend the time discussing important issues that everyone is already aware of.

In a nutshell, you spend the time solving problems, not explaining or defending them.

Cheers,
Clinton

Clinton Begin on December 15, 2007 10:11 PM

Having tried pair programming for the first time this weekend I wholly recommend it.

My friend and I managed to develop a simple 2d physics simulation engine in a few hours... virtually from scratch with very few problems other than the usual "getting it to build for the first time on another machine" because of the original project's lazy set up... but the actual coding was faster and more productive with someone looking over my shoulder and stopping me from naming variables badly or optimising needlessly. :)

Jheriko on January 28, 2008 8:42 AM

Every study on pair programming is flawed. They always compare 2 programmers versus 1. Every single study does this.

That norwegian study linked to earlier put 98 programmers up against 196 programmers working in pairs and found that there was NO difference in output between the 196 programmers versus the 98. They did find that the 196 programmers "worked harder".. but that just means they weren't as efficient as the solo programmers.

Sean on February 5, 2008 2:21 PM

Interestingly enough, I don't buy pair programming that much after experiencing it first hand. Before pair programming i had experienced code reviews as a developer and did code reviews as well. (off course how effective code reviews are depends on how "constructive" they really are, (refer to http://anirudhvyas.com/root/2008/05/04/ever-controversial-and-sometimes-annoying-code-reviews/)
But keeping that topic aside, pair programming can be harmful when a senior guy sits with a medium to fairly experienced guy and starts emphasizing on right way to do things. Off course there is no end in sight as the "right" way might not sit well with other programmer who might be passionate about programming styles too.

I have seen this happen quite often, a senior developer doing more things, which allows the so called "junior" or may be not so junior programmer to sit idle and become kinda lethargic in thinking. Off course bookish knowledge tells us that this should not be, because if one is programming on something, other might be thinking on something else, but how often is that true ?

Regards
Vyas, Anirudh
http://www.anirudhvyas.com

Anirudh Vyas on May 4, 2008 2:54 PM

Personally, I love pairing, and have for years. I'm sure some people never will, but I find a lot of people who hate it are doing it wrong. So I made a list of 21 wrong ways to do it:

http://agilefocus.com/2009/01/21-ways-to-hate-pair-programming/

This is also a nice set of FAQs for the novice:

http://aydsoftware.blogspot.com/2009/01/riddle-me-this-mr-pair-programmer.html

For those who are worried about introverts, I wouldn't sweat this too much; even introverts can pair. In fact, many of them are especially good at it, as they are thoughtful, considerate, and polite. Plus, interacting as experts around a shared task in a comfortable environment with familiar people isn't usually too painful for them. Keep an eye out for introvert/extrovert pairing, though; the extroverts sometimes need a little help in learning to not hog the keyboard or wear the introvert out.

William Pietri on January 16, 2009 10:34 AM

Once you decide to try, don't mess it up!

http://mikevalenty.blogspot.com/2009/05/5-ways-to-fail-at-pair-programming.html

Mike Valenty on May 10, 2009 5:15 PM