Pair Programming

My mind to your mind. My thoughts to your thoughts... -- MrSpock

Pair Programming

> An ExtremeProgrammingPractice in which two engineers participate in one development effort at one workstation. Each member performs the action the other is not currently doing: While one types in UnitTests the other thinks about the class that will satisfy the test, for example. A single, unsubstantiated, unscientific, undergraduate's survey has shown that, after training for the "PeopleSkills" involved, two programmers are more than twice as productive as one for a given task.

```
Once upon a time there was an engineer
Drove a locomotive both far and near
Accompanied by a monkey who would sit on a stool
Watchin' everything the engineer would move
   - Jesse Fuller, "The Monkey and the Engineer"
```

"LaurieWilliams of the University of Utah in Salt Lake City has shown that paired programmers are only 15% slower than two independent individual programmers, but produce 15% fewer bugs. Since testing and debugging are often many times more costly than initial programming, this is an impressive result." *(http://www.economist.com/displayStory.cfm?Story_ID=779429), September 20th 2001* For more on this, see PairProgrammingBenefits and LaurieWilliams.

I see ... so if you reduce the rate of progress, and if you measure the number of "bugs", they seem to be less!! hmmm I guess at slower speed, we get somewhere "clean".

Done right, pairing should produce some lines of code which neither pair can remember who wrote them.

How do we know that this is an improvement over a lone programmer spending 15% more time to get the code right the first time?

If you look more carefully at the numbers, there is actually a 50% reduction in the number of bugs. (The passed tests went from 70% with solo programming to 85% with pair programming. This is 30% failed tests compared to 15% failed tests, or 1/2 the number of failed tests). It's also conjectured, but not proven, that this reduction in bugs will later gain you back the 15% in "lost" productivity. *(warning about comparing percents: you need a 100% gain in your stock portfolio to break even after a 50% loss)*

Next, because a small bug introduced in development, and not cleaned up, can often cause an order of magnitude more trouble in the field. So a gain of 15% in rote development stands to save 150% later on. To put it another way, a colleague once humble-bragged he was a good enough gambler he was net-positive on his junkets to Las Vegas. I requested he subtract the costs of hotel rooms & meals. This is the FramingProblem; when you frame some data and analyze it, then switch to a larger frame and reanalyze, the answer might be very different. PairProgramming has immeasurable benefits beyond the low bug rate & clean code.

Where are the other studies to back this up? The only one quoted by the Economist (or indeed anyone) is LaurieWilliams' now famous "15%" study.

Medical quality studies cost millions. Just look at the companies that mandate pairing. How are they doing in their markets these days?

*(And note: From a scientific perspective, the "15% more time" figure is **not statistically significant.** That is, given the small sample size, there could be no additional cost. Or it could be over 15%. You just don't know.)*

It is also a pattern; see PairProgrammingPattern.

Before contributing to this page, please consider if your comments might be better suited to one of the following:

- **PairProgrammingQuestions**
- **PairProgrammingStatistics**
- **PairProgrammingDoubts**
- **PairProgrammingObjections**
- **PairProgrammingLimitations**
- **PairProgrammingGoneBad**
- **PairProgrammingMisconceptions**
- **PairProgrammingVariationsAndAlternatives**
- **PairProgrammingNotDoingItPain**
- **PairProgrammingNotDoingItPleasure**

On many of the above pages, communication seems to have broken down between proponents and detractors of pair programming. The debate seems to have two sides: "pair programming always works" and "pair programming never works". It's pretty clear, after hearing all the testimonials from both sides, that it works sometimes and doesn't work other times. Please consider contributing to the discussion in the mode that there are features of a problem, project, and person that make pair programming more or less effective. Try to assume that there are valid reasons for these differences and they are not simply the result of deficiencies.

""Purr Programming""

A evolution of pair programming where a human dev pair is substituted for a feline dev pair. http://www.purrprogramming.com

---

PairProgramming Markers

WardAndKent lead a BOF at OOPSLA'97 that included looking for patterns in the code and in the recorded dialog between programmers as they work together. These are the patterns they found ...

- **Let me drive.** (One of the idioms of PairProgramming)
- **x - 1 // n + 1** (FencePostErrors?)
- **inject:into:**
- **role suggesting variable name.**
- **trust me.** (Sometimes you have an idea for how to do something, but not the words to explain it. "Trust me" gives you a couple of minutes to type it in without your partner stopping you.)
- **look where you would write it.** (When looking to see whether a method you need already exists, look in the place you would put it if you had to write it.)
- **AskTheComputer.** (Don't reason about what will happen if you do X. Do X and see what happens. Send the message and see what it does.)
- **multiple pens**

- **BlameYourselfFirst.**

Contributors: MichaelFeathers KatyMulvey and others.

PairProgramming History

Just saw that I had marked (and forgotten) a section in "Constantine on Peopleware" in which he described a visit to Plauger's group, where they were programming in pairs, with the same descriptions as WardAndKent use. Constantine called it "Dynamic Duo" development.

That makes, so far, 3 or 4 separate inventions of this technique that I have encountered, each finding it advantageous. -- AlistairCockburn

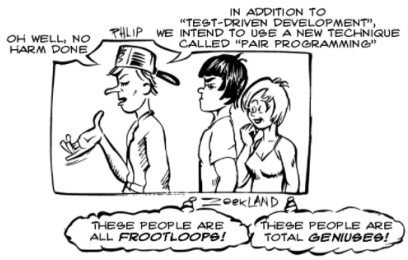
And a 5th time ... I was leading a team of 4 programmers working on a flight simulator in 1996. Each programmer was assigned a separate aircraft system. The programmers came to me and told me that, as they were always helping each other (this was FORTRAN on a weird system...), they wanted to work in pairs on each system. This seemed inefficient to me at the beginning, but the big improvement came when the weaker programmers improved within a short time and there were fewer bugs in the fixed time we had for the project. -- NissimHadar

See also DijkstraPairProgramming

PairProgramming Tidbits

- Anyone else tend to type "ProgramminginParis"? (Must be wishful thinking.)
- It is sometimes useful to have DivergentPairs work together, to spark "creative abrasion". -- YonatSharon
- When people say that PairProgramming reduces productivity, I answer "that would be true if the most time consuming part of programming was typing" -- MartinFowler
- One more pattern in PairProgramming that I found very useful is RecordYourCommunicationInTheCode. -- JuneKim
- Those of us who spent time training others know that this is virtually the only way to really teach someone something that involves hands-on work, which programming definitely is. -- James Clover
- KingCrimsonOnPairProgramming

PairProgramming Case Study?



Full write-up at: http://zeekland.zeroplayer.com/Uncle_Wiggilys_Travels/43

What programming, especially in pairs, feels like, at least in one case:

http://www.salonmagazine.com/21st/feature/1997/10/cov_09ullman.html

Yuck. I read the article with growing horror. Just another example of pseudo-heroic sloppy cowboy debugging. PairProgramming appears to give back more than it takes - the opposite of what the article illustrates. Having not done PairProgramming (yet!) I can't speak from experience, but I know someone could... -- RodneyRyan

It's a weird book (CloseToTheMachine). She has a very strange, dark outlook. But much of what she writes expresses a truth that many have lived. -- RonJeffries

I agree. However, I'm asking if people who have practiced PairProgramming view the article as a good example of PairProgramming. Is it a situation that indicates movement in the most effective direction? -- RodneyRyan

This would require me to read the article which I shall shortly, but I have informally used pair programming many times throughout school, graduated in 2010, and found it to be very good at reducing later bugs. Anyone who has ever programed a non-trivial program should agree that the issue isn't the time it takes to write the first iteration of the code but really the very very long tail of removing the bugs in the program. PairProgramming takes a bit longer to write the first time but it produces much cleaner code. I for one would think that monoganamous pairs are a good idea, or atleast in a relatively small group as learning each other's quirks is one of the personality issues with forming a GoodPair, -- AndrewRicketts

Ullman got the mind-meld part right. And she understood that pairs didn't have to be equals to be effective. But she couldn't keep the meld up through completion with the first developer and couldn't even start to meld with the second. Amateurs. -- WardCunningham

If you took out all the macho midnight brainburn stuff, the experience isn't atypical. Done calmly and peacefully, however, it's more a gentle flow than this orgasmochistic explosion of sensation.

If both of you have a beer before doing PairProgramming in my experience sometimes seems to help the process! -- DanielPoon

---

Pair Bug-Finding

Although we don't practice Pair Programming, we've been using it then trying to find bugs. In this case it seems to be a GoodThing.

However, our team is too small to use it all the time.

-- GeraldoXexeo

I'd argue that small teams will naturally start some pair programming (and pair everything else: planning, design, debugging, etc.) naturally because of the low communication costs. Oh wait, I did argue that: http://blog.markturansky.com/archives/69.

Maybe PairProgramming should be renamed ContinuousReview?

A couple more thoughts on why I think XP is not, as Steven asserted, "just a codification of bad programming practices in bad environments." One key point I haven't seen in the arguments here are the constraining (read improving) effects that PairProgramming places on rogue programmers with a JangIt mentality. Even if you tend to have cowboy coders in your group, pairing them up with an experienced XP'er virtually guarantees the code will be written properly and the necessary testing will also be put in place. Even if two JANGmeisters are paired together (and the rest of the team will know when they are) the resulting code will still be better than if either went off on their own. While not the most desirable scenario, as discussed in other pages (VcapsProject, PairProgramming), it still provides a certain set of checks and balances between the two since no two people think alike. Since the rest of the team knows about the volatile pair, any offending code will be corrected during ExtremeProgrammingCodeReviews. In other words, the team knows

what code is going to have to be reviewed more thoroughly when certain pairs are put together (it's just a reality of working in groups). When combined with the other factors of XP what results is the **elimination** of bad programming practices and bad environments.

Pairing also gets rid of any problems resulting from *mediocre* programmers writing poor code and documentation, as mentioned above. It seems the dissenting opinions keep forgetting that arguing against one particular aspect of XP won't work because it is the other aspects that balance out most of the possible negative side effects when taken individually.

FYI, I'm speaking from the point of personal experience in that I didn't think PairProgramming was going to help me improve my code and make me faster. So I went in quite skeptical. As you can tell, my views have changed just a bit. -- TomKubit

So what, PairProgramming is a BandAid on bad hiring decisions?

*How about, I dunno, **ReplaceMediocreProgrammers**. Might be just crazy enough to work.*

Arguing against particular negative aspects of XP is good for discerning which parts of XP might hold some value for you, when you're reluctant to just swallow the XpReligion whole. Take what's useful. Leave the rest.

XP is not a 'pseudo-methodology' as Steven asserts above that is allowing 'proponents of this technique have simply given a name to poor programming practices'. The XP methodology does, however, require a mental attitude that embraces change, and a willingness to learn, not only from ourselves but from others. One of the strongest strengths in XP is PairProgramming. We have found that the interactions that happen within a pair team is vastly more than the sum of the parts would produce. Even the most experienced coder finds that they are producing a design/code that is at a much higher level than they could do alone. We have found on the VcapsProject that there is typically less indecision in a pair team, ideas are thrown back and forth, and the solution space is quickly narrowed as advantages and disadvantages are discussed within a pair team. XP is a philosophy, a methodology that I strongly believe in. -- KevinBradtke

---

Is it just me, or is PairProgramming one of the most difficult XP practices to implement (if not the most)?

For some people, it is. For some people, it's perfectly natural.

Granted. I meant that, in testimonials about attempts to implement XP, I read of more resistance to PairProgramming than any other practice. I think.

UnitTests are my reviewer. If the code works according to the UnitTests then why is continuous review necessary?

Who wrote those UnitTests? If it was you, how can you be sure that the UnitTests cover all bases? A continuous human reviewer can catch issues like that, and can also make the development process much faster by suggesting improvements that cut down on development time or potential bug count. -- BrentNewhall

There is also the argument that adding a reviewer reduces quality. Each person tends to rely on the other to find the problems, and neither becomes responsible for discovering problems.

Well I hope I'm not intruding as I'm quite new to this wiki and the general WikiWorld. But I have a curiosity about PairProgramming. Now I've seen the doubt raised before, but the response was not very clear. Doesn't PairProgramming interrupt the ProgrammingFlow? I find that when I work with other people I always have to 'snap out of it' when they ask me something? Most of my projects I did at the university were with the same person. Now this person didn't have as many programming skills as I did. But every time he'd ask me a question it would interrupt my ProgrammingFlow and I'd have to say 'wait up' and try to get back into it. Now this was mutually frustrating. First of all, I couldn't continue the part I was working on, and secondly it made my partner feel like he was incompetent. PairProgramming can be good (though I've never tried it) if you divide the tasks, one does testing the other does implementing. But sitting with two people at the same screen seems a bit overdone. Also in the projects I've worked with I've always found that time for writing tests is never available. That doesn't mean manually debugging of problems aren't done (like when something won't work the way you want it). -- AnonymousDonor

For one, in PairProgramming, you're operating according to a different MentalStateCalledFlow than you are when SoloProgramming. Note that generally PairProgrammingIsDoneByPeers, which goes against the example of programming with someone who doesn't have the same skill. There should be fewer explanations and more collaboration. Also note that PairProgramming is somewhat CounterIntuitive. It works, even though it doesn't necessarily seem like it would work. -- BrentNewhall

I would also like to comment on the above. If you were programming together with somebody else and this was at the university shouldn't you have tried to find another lab partner then? It feels like you just destroyed your friends possibilities to actually learn anything. He doesn't want to ask questions and you just do all the difficult things that he should learn. What's the point with having a partner that isn't as good as you? And about spending time on tests. For really small projects I can agree, but on any medium to large university projects (that means mostly everything after the first year or so) you will actually gain time if you write tests. I have spent many long nights looking for bugs that would have been found immediately if I had written tests in advance. -- DanielLuna

My answer to this would be to have the less experienced developer drive. The observer should talk him/her through it, incorporating ideas from both. A craving for one's own gratifying MentalStateCalledFlow has to take a back seat to maintaining a common understanding and a productive dialog. It may take a while before the more experienced developer can drive in this situation. If it takes too long, either the more experienced developer isn't doing it right, or the less experienced person is in over their heads. -- ChrisDailey

Several of AnonymousDonor's problems seem to arise from failing to follow the rules - rules that I tell my students every semester:

1) As ChrisDailey and LaurieWilliams both say, let the less experienced person do the typing. That way neither person can zone out, the weak partner will learn from time-on-task, and the strong partner will learn from having to explain things. At first, it'll be slower than the strong partner doing it directly, but the weak partner will quickly become a useful member of the team.

2) PairProgramming is not about divide the tasks, one does testing the other does implementing; it's about both partners being involved in everything.

3) time for writing tests is never available. You can't possibly be in that situation if you write the tests before writing the code they are to test, as recommended in TestDrivenDevelopment. -- SBloch

---

Random Thoughts on Pair Programming: Knowledge transfer

I read through all of the above. I feel there are important points missing from this discussion. (Arrogant Bastard that I am).

My Experience with Pair Programming can be summed up as follows. One group I worked with I was the J2EE guru (I was hired for my knowledge of J2EE). We adopted pair programming. I found that not only was I teaching and mentoring the novice J2EE developers, but after a few weeks (even more so after a few months) I was learning a lot from them. J2EE is big (EJB, Custom Tags, JSP, JMS and more). The knowledge of the team increased exponentially. This could not have been accomplished without Pair programming. I have never increased my skill level so much in such a short

programming. I have never increased my skill level so much in such a short amount of time.

The amount of experience and knowledge gained by pair programming with ten people for a year is equivalent to working for five years in a shop that does not use pair programming. (This includes domain knowledge transfer.)

Also, we started building a reusable framework. I don't think we could have made so much progress without pair programming. Code cannot be reusable if no-one knows how to use it. Pair programming grows in value the longer you do it. Not reinventing the wheel means less code in the code base to do the equivalent functionality. Less code is less code to maintain (and learn). The benefit of pair programming is not linear. It grows the more you use it.

What do you do with people who will not try pair programming? Fire them. Just kidding (sort of). I found through experience that the people who have the hardest time with pair programming usually become the biggest pair programming advocates. Here is the recipe. Set a metric that every team member do at least 10% pair programming (4 hours a week in the U.S. .... 3.2 hours in France). Slowly raise this to 50%. I have not had one person who has tried pair programming not prefer it. Believe the 96% people prefer it figure cited earlier. (Our team could only achieve 80% pair programming or so per week as in 32 hours a week. The rest of the time spent in meeting with customer (5%), planning(5%), designing in groups(5%) and working alone (5%).)

Here is a suggestion that is not going to go well with some. A tree grows best if you trim some branches from time to time. Not everyone on the team is going to work out. This gets amplified with pair programming. Unlike environments that do not use pair programming, environments that use pair programming are very sensitive to stinkers and slow pokes. Notice I am not saying novices. There are fast smart novices, and there are hard to work with experienced folk. We have had experience with getting rid of a few people and our project velocity increasing (the people were causing more harm than good). With pair programming, it is easier to identify these issues because everyone works with everyone else (switch partners often facilitates knowledge transfer), and the team knows who is not pulling their weight. In fact when we made adjustments to the team, no-one complained.

Pair programming works best with talented, professional folks. Unless you prefer working with talentless, unprofessional folks, pair programming will work like a charm.

See HowToPissOffYourPair

Question - if you have a set of PairProgrammers say P1,P2,... say up to P5 , how best to cross-train so if one or more members of each pair (randomly say P11 and P32) is off sick, on vacation, or leaves permanently the group can be re-arranged to continue efficiently PairProgramming? Ie should P11 cross train with p(i+1)2, P51 train with P12 by switching places once a week, month or whatever so there is a high probability of someone who knows each project to change pairs if needed? Even though there may be documentation, version control and the remaining pair member can explain to the new one etc wouldn't scheduled cross training reduce learning time and make transitions easier when pairs get shuffled?

Huh? PairPromiscuously.

Pair Programming can be considered as a shield from the negative environment impact: http://www.jroller.com/comments/deep/Weblog/shield_from_real_world_impact . -- RomanPorotnikov

---

Any one got examples of PairProgramming concept working well for CobolLanguage projects? Any site names and any special considerations to make it work? -- dl

---

A Personal Experience with Pair Programming

The most successful use of PairProgramming that I have personally experienced is on my current contract. We are dealing with a legacy database that is "unnormalized to the extreme" and cannot be changed for a variety of reasons. The programmer I am working with has little ObjectOriented experience, and is new to using an OO language, but he is a fast learner and an expert with respect to the data and domain.

He is learning OO techniques, Patterns, divide and conquer and the language issues etc while we are speeding ahead much faster than if I had to take more time to pre-learn everything about the data in the database, much of which is irrelevant to this project. We are experiencing a magnitude increase in productivity over working independently. In addition, our combined understanding of the system is leading to improvements in the design and the maintenance of this system will be greatly simplified as a result.

We got a good laugh over the "mind meld" comment and photo above, as this mirrors how we feel about this experience. I also believe that the personalities of the individuals involved can make or break the PairProgramming experience. So ... it is not for everyone, but when it works it really works well!

-- JudyGreen (Judy N. Green)

When done with the right person, PairProgramming feels like a turbo booster to your mind.

I first experienced Pair Programming many years ago, my partner and I were both Pair Programming virgins and were so blown away by the experience we got married the same year. We've Spouse Programmed for over a decade now and would strongly recommend it.

If there isn't a Pair Programmers Dating site already there should be.

Has anyone tried pair programming over VNC or something with a faster protocol (text or IDE based rather than graphical). I'm thinking about developing a network system in a text editor project so that programmers can pair up. Currently I think VNC is a bit slow so obviously a custom in-text-editor design would assist the speed.

See VirtualPairProgramming and CollaborativeEditing.

I'm getting the feeling that pair programming is holding back the adoption of eXtreme Programming. There's a general fear of pair programming. Developers often prefer to work alone, as I do. This makes it harder for me to implement test driven development, which I like and most agree. -- RandyCharlesMorin

Have you tried pair programming?

Yes and I wasn't thrilled with the results. But that's not my problem. It's an adoption problem. -- RandyCharlesMorin

---

Things like the PairOn would help the adoption of PairProgramming in many organizations.

At my school we tried implementing pair programming in a couple first year classes. For a bunch of possible reasons it didn't work out. The way we did it was to pair people up at the beginning of the semester and they would work together on all assignments. Well, by midterms we realized that about half the class knew what they were doing and the other half did not. Perhaps it was too early to start pair programming, perhaps it wasn't appropriate for course work. It does seem to work for real programmers though.

I'd think that the grade issues would make pairing difficult in an academic course - if your partner is truly behind the curve, your desire to pass the

course would be pressure to take over the development. In a work environment, there is no stigma about heavily coaching your partner, so you can make a difference in how they are. Also, IIRC PairProgramming calls for switching partners fairly often.

You could have made basic notes on ability and personality as part of the assignment marking procedure, then re-paired the students for the next assignment matching like with like. Then you get to focus more resource on helping the slow kids to learn slowly, while the fast ones go ahead with teaching themselves.

Another data point: for several years, I've assigned my freshman programming students to use pair programming, generally telling them to switch partners for each assignment (with 6-10 assignments in a semester, I figure the LawOfAverages allows me to assign reasonable grades). Many of them are reluctant to switch, often claiming that they've found the only student in the class whose weekly schedule is compatible. A previously-unmentioned benefit of pair programming in this environment is that I have half as many assignments to grade, so I can give more and/or quicker feedback. -- SBloch

---

It would probably be easier to sell if it were de-mystified. We used to call it WorkingTogether. Walk into any engineering shop and you'll find people gathered around white boards and workstations, collaborating on (not reviewing) design decisions. Before screens, it was common practice to desk-check each other's coding sheets. Even after screens, programming where it mattered (ISVs and OEMs) was still collaborative for two reasons: it produces better code and it eliminates the risk of losing the only person who knows how a bunch of the code works. As a manager, I want every line of code to be clearly understood by at least two people - preferably more. This is in and of itself such an important requirement that it renders all other arguments for or against collaboration irrelevant.

[Anecdote: When I was field systems analyst at SDS, anytime a significant piece of software was ready for release a bunch of us would be brought into HQ for "familiarization". That involved the programmers walking us through every line of code and key path diagrams, so that we not only knew what every line of code did, we also got to propose lots of improvements. We all went back to our offices with copies of the source code on mag tape (like a CD but bigger and lower capacity).

Someone could have nuked HQ and SDS could still have gathered a gang from the field to continue work on the product. Oh yeah - the code was in assembler. A lot of lines.]

Pair programming seems novel because for the past decade or two we've been tucking programmers away in cubicles and allowing them to avoid WorkingTogether. Big mistake.

There are various setups and configurations that can be used for PairProgramming. My current experiences are summarized here: http://weblogs.asp.net/jcogley/archive/2004/10/13/242117.aspx. -- JonathanCogley

---

Q: I don't like PairProgramming

A: There is no *I* in *PAIR*!!!

Q: ...there is, 3rd letter in..!

A: EVERYBODY ROTATE!

WardCunningham has eloquently described the joy of sharing a CocoaWorld programming session with a child:

- http://c2.com/~ward/cocoa/notes/pairs.html

Anonymous: saves on the cost of computers and desks (but not chairs)

How do you handle a situation in pair programming where the two programmers have opposite philosophies? For example if one is data oriented and the other is information oriented? They will generally make the opposite decisions at every turn.

See also BasketballMetaphor, PairLust, PairProgrammingInCpp, HeardOfDyads, FlowNumber, PairVine, MarkYourTerritory, IsPairProgrammingMandatory, XpWithoutPairProgramming, PairProgrammingPowerLunch, MenDownaHole, PairManaging

---

CategoryPairProgramming CategoryCollaboration

Last edit June 27, 2014, See github about remodeling.