Responses to “How Pair Programming Really Works”

We knew Stuart Wray’s article in the January/February 2010 issue of IEEE Software would lead to a stimulating discussion, so we posted it online and encouraged readers to add to the dialog (see http://computingnow.computer.org/wray). Here are short summaries of some of the entries. Feel free to continue adding your two cents.

—Hakan Erdogmus, Editor in Chief

Linda Rising writes:
It’s nice to see an author bring appropriate bits from the enormous amount of cognitive science research that applies to our industry. ... The big message in this article is so important, not just for pair programming, but for all practices, especially those that are “agile-flavored.” It’s not about the “other pair of eyes”—it’s about the other brain and the other set of life experiences.

Andrew Main writes:
I pair-programmed in the 1980s (feeling slightly guilty at the time; many thanks to Kent Beck for bringing it respectability). I heartily concur that it was not driver-navigator. It was two people debating higher-level architecture issues, or seeing clashes of intent and proposed coding, or noticing semantic inconsistencies—and of course spotting typos. The big wins were (a) the architecture debates—places where the detail of coding would confront us with much larger thoughts about the whole structure, and (b) the clashes of intent and coding, and semantic inconsistencies—places where we saved a lot of time by avoiding constructing code that had fundamental weaknesses.

Andy Raybould writes:
Even people who know better succumb to the “just do it” imperative, and fall into trying fixes rather than thinking about solutions. ... As for the expert effect, ... could there be a competitive aspect to discussing our problem with an expert, which prompts us to strive harder and perhaps more desperately? ... I am thinking particularly of those occasions when the expert doesn’t have to say anything to be effective.

Joshua Stern writes:
I find pair programming absurd. I suppose I’ve done it, on occasion, mostly in search of recalcitrant bugs, but for mainstream development? One keyboard, seriously? ... Remember, the benchmark for effectiveness has to be the productivity of two developers working separately. ... The process of communicating with a partner slows me down by 90 percent or more, and I do not see any offsetting increases in quality or any other benefits. A merger of skills? I hope the individuals involved have sufficient skills for their work, and if not, it sounds to me like the old theory that tying together two rocks will enable them to float.

Roy Morris writes:
I was involved in two projects using some form of pair programming. Project 1 used it for all aspects of development, from design to implementation to fixing bugs. The project failed because it was late and over budget. Project 2 used a modified form of pair programming. Design and implementation of complex components were done in pairs. All remaining (noncomplex) components were designed and implemented by a single engineer. Component integration

CANDIDATES MUST HAVE:

- Bachelor in Comp Science/Engg., Math or Business & 2 years experience in related technology group
- Some positions require Master’s Degree or equivalent with two years of experience OR Bachelors with five years experience

TECHNOLOGIES WANTED:

- JAVA, J2EE, JAVA Server, JSP, XML
- VC++, C++, C, MFC, COM/DCOM, OOD
- Systems Administrators, Unix/Windows/NT
- Win Runner, Lead Runner, TDL, Rational Suite
- VB.NET, ASP.NET, VB 6.0, MS SQL Server, Oracle
- GIS, ARC INFO, ARC GIS, ARC MAP, ARC INFO
- Siebel, Siebel Architecture, Siebel Analytics, EIM & EAI
- C++, Unix, PERL, CONCISE, Oracle, Multithreading, SQL, Protocol, TCP & IP, C#.NET
- C**B, C++, C#, DB2, JCL, VISUAL
- Database Administrator, Oracle, SQL Server, SQL Server, DB2, Power Builder, PERL, Oracle, Sybase SQL, Server
- Management Analyst - UNIX, UAD Sessions, Rational Rose

QUALIFIED CANDIDATES:

Please EMAIL CV to: hi@nuiws.com or FAX to: 504-828-3883 or MAIL to: Attn: HR Department 8401, Lake Worth Road, Suite 225 Wellington, FL 33467

www.nuiws.com

IEEE SOFTWARE  Published by the IEEE Computer Society 0740-7459/10/$26.00 © 2010 IEEE
was usually done in pairs, but not always. The issue of complexity was left to the assigned engineer(s) and lead. Fixing of bugs found in quality assurance was left to a single engineer to fix or enlist help if they were having issues identifying and/or fixing the problem. This project came in ahead of budget and schedule.

What did I learn? Code quality had nothing to do with whether it was written by a single engineer or a pair. Unit tests were the key to reducing total bugs found in QA. However, design quality was greatly increased in pair programming (as compared to other projects I have been involved in).

Would I do pair programming again? No.

**Andy Glew writes:**
I practiced “solo XP” for circa six years … then worked in small pair programming teams on several occasions: side by side, facing each other, and the classic single-keyboard, single-display model. Much to my surprise, single-PC pair programming worked pretty well. Frequent and constant interaction is the key to pair programming success. Frequent and constant code review.

A big advantage of pair programming is that good teams get stuck less often. It’s not just the potted-plant effect; often your partner solves the problem. But having a partner also sometimes forces you to be more pragmatic. Instead of thinking about the best possible way to solve a problem, you might be willing to compromise on a way that is good enough for now.

At Intel we used to create “dungeons” where people worked together intensely for weeks or months to solve problems—usually at the end of a project death-march, when pressure was high. Q: If war-rooms are a good way to get things done, why not work in them all the time? A: burnout. Pair programming is a way to produce burnout.

All good programmers are embarrassed when they write bad code just to be expedient. Sometimes, I admit, I have done so. I’m just less likely to when pairing.

We also tend to stay on target. Team members spend much more time working and coding, and much less time responding to email … so much so that successful pair programming teams must consciously reserve time for nonprogramming work.

It must be said: I find that eight hours of pair programming is more exhausting than 16 hours of solo work. You simply work harder while pair programming.

Would I pair-program again? Yes, in a heartbeat. I found it more fun. As a senior computer architect I was spending less time coding and more time attending meetings, often to coordinate the activities of junior coders. Pair programming allowed me to coordinate, but also to code more, rotating between partners. So, definitely, in pair programming versus attending meetings, coding wins.

But pair programming even wins over solo programming. I hate to admit it, but I am a social animal. I like teaching people. I enjoy interaction.

Let’s quantify this: I earn roughly $200K/year as a technical expert in my field … I would accept a pair programming job for $150K, possibly even $100K/year, even if it wasn’t in my main area of current expertise. In fact, pair programming would allow me to cross over to another field.

**Nico Coetzee writes:**
I work for a large banking group in South Africa. We took the open-plan office idea and made it even more open: typically, tables are all joined in the center of the floor area with no partitioning. Think of it as a round table where everybody can see each other. The outer walls are covered with magnetic white boards, encouraging developers to talk to each other. It’s easy in this set-up to quickly brainstorm a problem. During other “normal” coding periods, most developers have their iPods in their ears to eliminate distractions. We are at a comfortable middle ground, giving developers an opportunity to pair up only when required.

**William Miller writes:**
I have never experienced pair programming, but I can see how it could get the job done. I always think of the features that are needed, but often not until it’s too late to do much about it without starting over.

**Stuart Wray responds:**
You’re probably right that when we talk to people we regard as experts, we are more “keyed up”—we don’t want to embarrass ourselves. Even though we’re going to them with a problem of our own creation, we want them to have a good opinion of us. It makes sense that for a competent programmer, this extra pressure would improve his or her performance.

Regarding “just do it” slot-machine programming, and the benefits of stepping away from the machine, I agree that this often helps. … But exhorting people to “be good,” to stop back and think, seems to have very limited impact. I continue to be amazed at how students in my programming class each year end up trapped in a cycle of hopeful tinkering. Without stopping to understand their problem, they make a change to their code, “put a coin in the slot,” and hope for the best. … If we have the discipline to use the available tools in a more productive way (for example, by thorough unit testing), that’s better. Pairs can help with both, but we have to remember to agree ahead of time on how we are going to work.

I wouldn’t claim that pair programming is good for everyone, or for all kinds of development. Although pair programming experiments don’t usually show a doubling in speed, they generally do show some reduction in “wall-clock” time to finish compared to a solo programmer doing the same work. So, for some people, pair programming is a way to add programmers to a project and get an earlier delivery date. This is rather surprising, since it directly contradicts the conventional wisdom that adding programmers to a late project makes it later.

Outside of experiments, the true benchmark of effectiveness would be to compare the productivity of a team of pair programmers with a team of solo programmers. When you factor in the interpersonal communication and training necessary to keep a team functioning over a long period, you might find that the apparent inefficiencies of pair programming aren’t as severe as they first appear. ☺️

---

*Letters are edited for clarity. Include your full name, title, affiliation, and email address. We welcome your letters. Send them to dstrok@computer.org.*