Will Pair Programming Really Improve Your Project?

Articles

Posted by:

Posted on : 2007/11/19 3:56:26

This article is an excerpt from Chapter 6 of the book Extreme Programming Refactored: The Case Against XP [1], by Matt Stephens and Doug Rosenberg. The book provides an entertaining look at some of the flaws behind Extreme Programming (XP), whilst suggesting some alternative strategies and practical techniques to achieve XP's agile goals in a more rigorous way. For this article we concentrate on pair programming - and in particular the book Pair Programming Illuminated [2] by Laurie Williams and Robert Kessler.

Authors: Matt Stephens and Doug Rosenberg, Software Reality, www.softwarereality.com

Pair Programming Illuminated (we refer to it as PPI for the rest of this article), not surprisingly given its title, pitches the case for pair programming. However, it also discusses quite openly some of the problems typically encountered by pair programming teams. In this article, we focus on some of those problems and examine how they may affect an XP project as a whole.

Problems with Pairing Different Categories of Programmer

PPI divides programmers into different categories and then discusses the effects of the various combinations thereof. The programmer categories are novice, average, expert, introvert, and extrovert. The pairing combinations discussed in PPI, with a chapter dedicated to each, are as follows:

* Expert-expert
* Expert-average
* Expert-novice
* Novice-novice
* Extrovert-extrovert
* Extrovert-introvert
* Introvert-introvert

Because pairs are meant to rotate frequently, these various combinations will resurface often in a team of mixed abilities. Thus, in small teams (which is likely, given an XP project), it would be difficult to keep "problem pairs" apart.

"Go Make Me a Cup of Tea" Syndrome

What happens if you pair up a newbie programmer with an expert? This is described in PPI as "expert-novice pairing." The intention of such a pairing would be to "get the easier job done well, while training a novice programmer." The challenge of such a pairing is primarily that the expert must take on a tutoring role and must maintain extreme patience throughout. If the expert coder slips, then the result is a "watch while I type" session (sometimes called "go make me a cup of tea while I finish this program" syndrome [1]), in which the novice remains passive throughout and the expert is

effectively solo-coding. Despite this, there are distinct advantages to expert-novice pairing. In fact, it’s probably the one pairing combination that’s worth mandating, as long as the novice is willing and able to learn and the expert is prepared to give up a portion of her day to teach rather than code in full-flow. This combination is certainly better than novice-novice pairing, which even XP evangelist Ron Jeffries thinks is a bad idea [2].

Laurel and Hardy Take Up Pair Programming

The intent of a novice-novice pairing combination is described in PPI as follows:

"To produce production code in a relatively noncomplex area of the project, giving valuable experience to both programmers in the process." [2]

If you’re considering such a pairing, it’s important to ask yourself which part of your project is unimportant enough that you can afford to unleash two complete novices, unsupervised, on it. "Unsupervised" is actually the key. Two novices, unsupervised, would likely produce code that isn’t exactly production quality. Luckily, PPI has the answer:

"There must be a coach, instructor, or mentor available to answer questions and also to help guide the pair. . . .We feel very strongly about the need for a coach. If you are unwilling to assign the mentoring task to some expert, then you need to understand the limitations of the asset being produced by the pair." [2] In XP, this responsibility would fall into the lap of the person (or people) performing the coach role.

As with the other pairing combinations, pairs rotate so frequently that in a team of mixed abilities, the novice-novice pairing could happen quite often. Therefore, novice-novice pairing isn’t something that can easily be controlled: It just happens, almost by accident, several times a week. The coach must be fully aware of the fact that two novices are currently pairing at any time, and the coach must be available to guide them and correct their mistakes. In practice, to combat the proverbial blind leading the blind, there’s a risk that the coach may become fully occupied with mentoring one particular pair anytime two novices pair up.

Carrying Your Pair

Similar but less extreme problems occur with expert-average pairing. PPI describes three situations where the authors feel that expert-average pairing is a problem. The first is that the average programmer truly is average (i.e., the average programmer is likely to stay that way and will never really progress). The second is when the average programmer doesn’t interact enough with the expert. The third is when the average programmer doesn’t seem to "get it" and keeps asking the same question over and over:

"This can leave the expert frustrated and can reduce the ability of the pair to complete the task." [2]

And the Winner Is . . .

Aside from the longer-term learning benefits, it seems that the most beneficial form of pairing is with two programmers of roughly the same ability. It’s more likely that the pair will be on the same wavelength and will spend less time disagreeing over things that probably don’t matter that

much. Unfortunately, when you consider that 50% of all programmers are below average, it becomes obvious that mixed-ability pairing is likely to be the norm. This highlights the problem that teams of mixed abilities are almost unavoidable. Pair programming makes the issue unavoidable by forcing these people to code together on the same program.

In a non–pair-programming project, the problem is handled effectively through other more natural practices, such as team leading, code and design reviews, occasional (voluntary) pair programming, mentoring, design documents, and so on. With almost all of the problems described in this article, it’s up to the coach to catch and deal with them as promptly as possible. This places a lot of responsibility on the coach (almost as much as the on-site customer!)

Design Documents Reduce Reliance on Pair Programming

Design documents provide a record of design decisions. This makes them particularly helpful for novice programmers to explore the thinking behind the design, as described by the more experienced senior programmers. If the team is becoming lost in a sea of changed minds and refactorings, the design document often helps to remind the team members of why they originally decided to do something in a particular way. There’s usually a pretty good reason.

We discuss the role of documentation in software projects (and how it can lessen the need for pair programming) in Extreme Programming Refactored Chapter 7 [1].

And More Problems

Chapter 7 of PPI (titled "Problems, Problems") discusses several problems with pair programming. We briefly discuss some of these problems here. Although the authors of PPI do offer some practical advice to overcome or help prevent these problems, the proposed solutions either result in high maintenance or rely idealistically on the programmers being constantly aware of all the problems (with advice such as "Just proceed a bit more cautiously"). One problem is that of rushing. Because pairs rotate often, they might rush to finish a task before it’s time to separate. The advice given in Chapter 7 of PPI is as follows:

"If a task must roll over to another pairing session, the task must roll over to another pairing session! Slow down, and do it right together." [2]

The coach would need to be particularly vigilant to spot this problem recurring, because pairs rotate so often. If the problem happens a lot, it may be because the tasks are too big (another direct consequence—evidence of the circle of snakes unraveling. To counter this, the team may need to spend more time planning or designing, or change its process for estimating stories or tasks). Another problem, which we suspect would particularly manifest in teams that publicly laud themselves as "the best team on the face of the Earth," (such as the original XP team that worked on the infamous C3 project) is that of overconfidence:

"There may be a feeling that a pair can do no wrong. If you’re working together, you might convince yourself that whatever you do together must be right. Remain cautious and careful!" [2]

The problem of overconfidence would need to be watched for carefully by the coach, who should be

aware of this type of problem. She would then need to be able to watch out for the telltale signs and be prepared to act on them when she catches pairs reassuring each other into writing bad code. "Well, I suppose it will do for now—we can refactor it later!" is the typical start of a slippery slope. Another problem is that it’s human nature for people to want to be in control, at least of their immediate surroundings:

"New folks should specifically be paired with mentoring types, lest they feel unwelcome or frustrated in the hands of a partner who wants to make only personal progress. This mentor must also give up control and allow the less skilled team member to drive most of the time. When the mentor is directing most of the activity, it’s better for the trainee to be typing and not just listening. The student might not be assertive enough to ask for the keyboard." [2]

This is, of course, an idealistic approach. As we discover in Rich Camden’s "Voice of eXPerience" account [1], being in control of the keyboard is the preferred option for most people. Rich wasn’t offered the keyboard once in 5 months (that’s not to say that he didn’t get to type, but no one actually offered to relinquish control). If the other person doesn’t speak up, he’s not going to be offered the keyboard. As the previous quote suggests, this is particularly a problem with inexperienced programmers being allocated an experienced partner. Everybody likes to be the driver, to be in control.

Watch Out, There’s a Snake Under the Desk!

The problems we just described must all be watched for and quickly fixed before they lead to other problems. This is a lot of problems associated with one XP practice, all waiting to slip and catch the unwary coach, who must be especially vigilant.

Pair programming by itself can be a beneficial practice, but in an XP project its problems are much more acute because (as we discuss in Chapter 3 of Extreme Programming Refactored) so much else in XP relies so heavily on its correct and consistent execution throughout the project.

References

[1] Matt Stephens and Doug Rosenberg, "Extreme Programming Refactored: The Case Against XP", Berkeley, CA: Apress, 2003

[2] Laurie Williams and Robert Kessler, "Pair Programming Illuminated", New York, NY: Addison-Wesley, 2002 Originally published in the Winter 2003 issue of Methods & Tools