Nathan Quirk  
Article Review 2  
“Strengthening the Case for Pair Programming” – Laurie Williams, Robert Kessler, Ward Cunningham, Ron Jefferies

In “Strengthening the Case for Pair Programming,” the authors investigate the efficacy of the pair programming model, and bolster anecdotal claims that pair programming yields better, more consistent code for a relatively small increase in labor cost. Prior to this article, smaller isolated attempts had been made to assess the potential of the pair programming paradigm, including a study by Temple University Professor John Nosek. Larry Constantine (author of *Structured Design*), and the creators of the Smalltalk programming language and developers of the Extreme Programming model have all written in favor of this idea as well.

With pair programming two developers sit side-by-side. They work together to suggest ways to attack a problem, and then one implements the solution while his/her partner provides continuous error-checking and might suggest alternative solutions to the tasks at hand. Each partner is equally responsible for the outcome of the project, and they typically will trade roles every so often. Programmers need not work in tandem at all times. When time or budget constraints get in the way, team members might split up to knock out more mundane tasks alone.

The article then details an experiment carried out in 1999 at the University of Utah, conducted with software engineering students. These students were split up and distributed according to GPA with a control group of students who were assigned the same project to do individually. The programming pair teams were even given other assignments to complete so that their individual workload would be exactly the same as the individuals.

The authors describe a short period, usually not lasting more than a few days during which they had to acclimate to working in pairs. The pair’s first assignment, on average, was completed sooner, with fewer errors and better efficiency compared to the individual programmers. During this introductory period, the pair teams did use more man-hours by about 60%, but after teams “jelled” this number came down to a minimum of 15%. That meant that two programmers make producing good code so efficient that assigning two people to the task only resulted in adding 15% more man-hours to the project. Additionally, the teams turned in code for each of the four given projects that: 1.) Produced more consistent results 2.) Passed tests cases more often than the individuals’ 3.) Finished programs more punctually. The authors credit the pressure felt by team members to not let their partner down. In short, pair programming seems to be make team members accountable to one another.

The authors assert that even with a modest increase in man-hours (read: higher labor costs) this is more than offset. Pair programming results in a vastly reduced time to market and higher quality code. This means that software engineers can rapidly prototype things for prospective clients. Defects in the code are extremely costly to repair, but with two sets of eyes on the problem, major hurdles can be overcome easily.

Perhaps most surprisingly, according the Utah experiment, 90% said that they enjoyed pair programming more than working alone.