Pair Programming

BY RUDOLF MAXIMILIAN SCHREIER

One would think that programmers, loners by nature and profession, would have quite strong feelings against being looked over the shoulder (they did invent the monitor-fittable rear-view mirror, after all). So why is this still a useful practice?

To start things off: Yes, Pair Programming is exactly what it says on the tab. In front of every workstation, there are two monitors and/or keyboards, and four eyes follow every new line of code. Most teams are made up by one active and one passive member, who switch positions at certain intervals, usually about 30 minutes[1]. This is supposed to ensure less tiring in programmers, and also reduce the risk of Repetitive Strain Injury.

Now most people will instantly think that, in the same time, these two programmers could have worked on their own and produced nearly twice the amount of code, right? In a perfect world, yes. However, as the saying goes, four eyes see more than two, and especially if you have programmers with different backgrounds working towards a common goal, the synchronous discussion will often bring up questions and solutions that may have seemed impossible to one hacker alone. The technique of Pair Programming has also been shown to very quickly yet thoroughly introduce new employees to common architectural feats, programming conventions and principles. You might not be able to contradict Brooks's law[2] outright, but you can at least do your best.

Quite rightly, you might also say: «But these are programmers we're talking about here, this is never going to work!». And indeed, it has been noted that a large amount of the problems arising from (forced) Pair Programming stem from ego conflicts between programmers. Combinations such as Extrovert-Introvert or Average Programmer-Expert Programmer all bring with them unique scenarios for confrontation[3].

And yet, research on the time- and cost-efficiency of PP has shown incredible results: A 2001 study by the University of Utah found that PP could cut the percentage of buggy code in half, while the speed of code production went down only 15% in comparison to two solo programmers. Considering that debugging is a more time-consuming and costly process, this is a significant badge of success for PP.

Results

On the other hand, smaller projects don't seem to profit as much as complex ones do, as a 2007 study by Arisholm et. al. of the Norwegian Simula Research Lab[4] using 295 professional Java consultants showed[5]: Complex systems increased in correctness by 48% without significant slowdown, while simpler systems slowed down by 20%, without gaining correctness. The Arisholm et. al. study has, however, been rightfully criticized for a number of reasons: Comparison was between pairs of programmers working cooperatively and one solo programmer, as opposed to pairs vs. two programmers, and not one of the participants had any PP experience beforehand, and so many accuse the study results to by falsified by «warm-up slowdown»[6].

But no matter if one believes the statistics, one cannot help wonder about the feasibility of it all – well, at least, the author can't – and as such, let's have a look at the technical means to this goal.

Tool support

Now surely, most of you could probably handle actual side-by-side PP, but outside of a employment environment, when can you find a time and place for two people to work together locally? And so, developers have come up with the merriment that is collaborative real-time editing. These tools allow synchronous editing of documents either in standalone editors or integrated in your favorite IDE. Let's take a look at some of them, shall we:

- **Gobby (GPL)** is a standalone general-purpose editor available for Windows, Unix-likes as well as Mac OS X through X11.app. However, it only supports syntax highlighting in versions > 0.4.9.
- **Emacs**, everyone's favorite operating system that includes an editor, also offers synchronicity through the command «make-frame-on-display», but only through X window systems, so don't count on Windows support here.
- **Eclipse** can be equipped with the plugin called «DocShare», which uses communication over either XMPP providers such as Google Talk or Jabber, or a Skype connection. As Eclipse is available over many platforms, this is one of the more portable options, but has the disadvantage that every user must own a XMPP or Skype account.
- **Mozilla Bespin** (yes, as in the Cloud Town Bespin) is an ambitious project by Mozilla labs to create a high-performance browser-based synchronous code editing environment. Written in Javascript and also based on HTML5, Bespin is available on many major browsers save for Internet Explorer, and can deliver an unusually high performance.

Now all that remains is to wish you good luck, much fun, and happy hacking.

Links

[1] http://www.gamecareerguide.com/features/760/a_day_in_the_life_three_slices_of_.php
[2] http://en.wikipedia.org/wiki/Brooks%27s_law
[3] Will Pair Programming Really Improve Your Project? http://www.methodsandtools.com/archive/archive.php?id=10
[4] http://www.simula.no
[5] http://simula.no/research/engineering/publications/Arisholm.2006.2/simula_pdf_file
[6] http://catenary.wordpress.com/2007/03/12/pair-programming-evaluated/