RUP®/XP Guidelines: Pair Programming

Robert C. Martin
Object Mentor, Inc.

Rational Software White Paper

Rational®
the e-development company™

Table of Contents

Overview

Pairing, a brief description

Pair Programming is a technique whereby software on a project is written by pairs of programmers. Each pair works together at a single workstation. One member of the pair drives the workstation while the other looks on, carefully watching the code being produced. The driver is thinking tactically, concerned for the line of code he is currently writing. The observer is validating syntax and is thinking strategically about the whole program. They trade these roles frequently and the resulting code is written faster than if by a single person and has fewer defects. Moreover, the code is intimately known by at least two developers.

The case for pairing

Consider a typical code review session. A module that required eight hours for one person to develop is reviewed for one hour by eight people. The net result is that 16 person-hours are spent on the module. However, the reviewers cannot spend the needed time to become familiar with the code and their review is fairly shallow. A single developer is intimately familiar, but perhaps too familiar to find the bulk of the defects.

Contrast this with the practice of pair programming. If the module requires eight hours for the pair to develop, a total of 16 person-hours will be expended. However, in this case, *two* developers will have intimate knowledge of the code. The defects hidden from one developer will be visible to the other.

The case for pair programming is simple, but the repercussions are subtle and far reaching. Pair programming is simply a much more effective way to write and review code. With two people intimately familiar with a module, far fewer defects will be written into the code. The code will have a better structure and intimate knowledge of it will be in twice as many brains. If these were the only benefits they would be sufficient, but the act of pairing provides still more benefits.

Pairs are more courageous: What a single programmer might be afraid to try, a pair will have the courage to attempt and the skill to evaluate.

Pairing fosters teamwork: Since modules are not written by a single person the code becomes the property of the team, rather than of a particular developer.

Pairing fosters the spread of knowledge: The more developers who pair with each other, the more the knowledge of the system diffuses through the whole team. The result is a team whose members are familiar with all of the system rather than each member knowing only their one particular part.

Pairing promotes productivity: A person programming alone goes through bursts of energy followed by periods of relative inactivity. Pairs pace each other. When one gets tired, they swap roles. They manage to keep the intensity turned on for much longer than a single person can usually tolerate.

Pairing is fun: Working with another developer is educational, stimulating, and just plain fun. Pairing increases job satisfaction and overall morale.

The Practice

Pairing

Pairing begins when the developer responsible for a task asks someone else for help. The rule is: when asked, you must say yes. This does not mean you have to immediately stop what you are doing. Rather it means that you must negotiate a time when you can offer that help and another time when you can get help in return.

The pair partner does not assume responsibility for the task. That responsibility remains with the task owner. Nor does the pair partner commit to staying with the owner until the task is complete. The pair partner only commits to help.

One member of the pair becomes the driver, while the other looks on. The driver types in the code, runs the compiler, runs the unit tests, and so forth. The watcher examines each keystroke, each command, each test result, and offers help and suggestions. Both parties are engaged at all times.

Sometimes the driver will know best what to do, and the watcher will simply be following along. At other times, the watcher will dictate what to do to the driver. Sometimes the driver will get frustrated and will hand the keyboard to the watcher,

thereby switching roles. Other times, the watcher will ask for the keyboard and switch roles. This will happen many times in a pairing session.

Changing pairs

Pair partners are not long term. A typical pairing session will last about half a day. Either partner can opt out of the pair for any reason. When this happens, the owner of the task must find another pair partner. This may mean that it is time for the task owner to pay back a favor to someone who paired with him or her last week. On the other hand, perhaps he or she should ask someone with the right experience to help with a particularly sticky problem.

This changing of pairs causes knowledge of the system to diffuse throughout the entire development team. Within a short time, every member of the team will have spent time working on almost every part of the system. This drastically reduces the sensitivity of the project to turnover, and makes every programmer more confident in dealing with the whole system.

Collective ownership

Since everybody works on all the different modules in the system, nobody owns any particular module. This means that responsibility for the system is not divided up on a module-by-module basis. Rather, the entire team is collectively responsible for the entire system. Any member of the team may check out and change any module in the system for any reason. When a pair makes a change to module X and that change causes module Y's unit tests to fail, the pair repairs module Y.

Pacing and collaboration

Pair programming is a very intense form of communication. Verbal dialog is often spars, and an outside observer might have trouble making sense of it. As an observer, you may hear the pair uttering singular words like: "semicolon", or "close brace". Or you may simply hear less articulate grunts as the programmers agree or disagree with what is appearing on the screen. The two are so intimately engaged in the code that is appearing that much communication is non-verbal. Body language plays an important part. One pair-partner can tell when his counterpart is uncomfortable with the code, even though no words are spoken. A grimace, a sigh, a nervous fidget—all conspire to increase the bandwidth of communication between the partners.

Sometimes one partner will grab the mouse, while the other operates the keyboard. The mouse-holder controls the location within the module where work will take place. The keyboarder controls the content that is altered or added at that location. At other times, one partner will be typing, and the other partner will foresee a function call coming and will open the API documents to the right page just as the coder needs the spec.

When one partner gets tired, the other can take the lead, allowing his or her partner to rest by playing the observer role. Other times, both partners will be high energy and will swap the keyboard and mouse frequently.

In summary, there are few rules and fewer procedures. The only real constraint is that both parties have to remain engaged and communication between them must be intense. A pair in which one partner is typing and the other is looking out the window is not truly pairing.

What can the lone developer do?

You can't be pairing all the time. Some projects—that is, those that adopt the *eXtreme Programming* (XP) (see reference [1]) process—follow the rule that pairs must produce all production code. In that case, when you aren't pairing you can check your email, read up on a new technique or API, read through code that you are unfamiliar with, or talk with the stakeholders about the current iteration or future plans. Indeed, there is always something profitable that a developer can find to do in those few hours when a pair partner cannot be found.

Some projects are less strict about pairing. Some will allow single developers to write tests. Others allow single developers to write abstract classes or interfaces. Still others simply allow the developers to decide when it's best to pair. One thing is clear, however—studies have shown that defect rates drop dramatically when pairing is practiced.

Some people don't like pairing

Some people feel uncomfortable with the concept of pairing. In our experience, those that actually try it for a week or so find that their discomfort evaporates, and that they enjoy pairing and find it useful. Very few continue to dislike the practice. So, for most people, it's simply a matter of trying it and getting used to it. For those who make an honest attempt and still find they dislike the practice, the team will have to find something appropriate for them to do.

Furniture, Facilities, and Logistics

Monitor and keyboard placement

Furniture placement is critically important to successful pairing. One cannot pair well if the partners cannot sit next to each other and rapidly swap the keyboard. The rule is: you have to be able to hand the keyboard and mouse back and forth without changing seats.

The best arrangement is usually a nice long flat table. Place the monitor in the middle and place two chairs facing the monitor. Sit with the monitor between you. Make sure it's easy to slide the keyboard and mouse back and forth between you. Also make sure that while you have the keyboard, you are comfortable and sitting erect. Make sure the monitor is visible to both partners without having to swivel it.

Bullpen

To facilitate the changing of pair partners, it's often wise to work in a bullpen arrangement. Place several pair stations in a single room. Use wheeled chairs and linoleum or tile floors. Arrange the workstations so that the pairs face each other. The goal here is to increase communications. Sometimes the most important communications are those that are serendipitous. We want to increase the chances that such communications will take place.

Inside corners

Many cubicles nowadays have workstations placed in the inside corners. The developer sits facing a corner of the cubicle with a monitor in front of him or her. While this is convenient for individual work, it's nearly impossible to pair well in this environment. If you have cubicles with workstations in inside corners, then set up some pair stations elsewhere—perhaps in a conference room. Pairing at a conference room table using a laptop is often very effective.

Problems and Concerns

Pairing halves productivity

It stands to reason that two people working together on one task will consume twice as many person-hours as one person working on the same task. As reasonable as this is, it doesn't seem to be the case. Independent studies (see reference [2]) have shown that little, if any, productivity is lost by working in pairs. Those same studies show that pairing substantially decreases the defect rate, *and the code size*, while greatly increasing job enjoyment.

Disputes between pair partners

The owner of the task has final say in all design disputes, but the best way to settle a dispute is to try both ideas and choose the one that works best.

Specialists

Conventional wisdom suggests that developers who specialize in a particular area, such as databases or GUIs, should apply their efforts solely to those areas. In a pair-programming environment, however, these specialists become mentors to others who don't share their specialty. Developers can sign up for tasks outside of their specialty and then recruit help from the specialists. This way, knowledge of specialties tends to diffuse through the project team, greatly reducing the sensitivity of the project to turnover.

Noise

A pair will make noise as they program. A bullpen full of pairs will maintain a continual, low-level buzz. Some feel that this noise will be bothersome and distracting. This has not proven to be a significant problem. If you find the noise distracting, you can move to a conference room for a while.

Cowboys

Many teams find that they are the proud owners of one or two cowboy coders. These are the programmers who get the job done faster than anybody else, cannot work with others, and nobody else is allowed (or would be able, if allowed) to read

their code. The best thing to do with these developers is to move them off the project or into a role where they are not writing production code. Perhaps they can write short-lived tools or do some maniacal torture testing.

Physical impediments and impediments of style

Some folks use QWERTY keyboards. Others prefer DVORAK. Some need special keyboards, mice, displays, foot switches, and on, and on. Some cannot program without headphones and loud music. Others must surround themselves with empty Twinkie packages. Some folks like emacs. Others like VI. Still others want to work in WordPad.

Indeed, the impediments that can be named are innumerable. But each and every one of them can be resolved with a little thought and the ability to compromise. A team that lets such things impede them is a team that will simply not succeed in any endeavor they try.

How can the team plan for pairing?

We don't believe that tasks should be assigned to pairs. Rather each developer should be responsible for a set of tasks. We like the pairs to form informally. Each developer, pursuing his or her own responsibilities, asks other developers to briefly help. The owner of the task keeps that ownership and accountability. The pair partners are just helpers.

Each developer must take into account the amount of time he or she will be pairing when offering estimates for tasks.

Conclusion

Pair programming is a well-tested, well-accepted alternative to code reviews. More than that, it's a fundamentally different way to write software. The benefits go far beyond productivity and quality, and affect such things as the robustness and morale of the team.

References

[1] *eXtreme Programming eXplained*, Kent Beck, Addison Wesley, 2000.

[2] *Strengthening the Case for Pair Programming*, Laurie Williams, University of Utah, July/Aug 2000 IEEE Software.

Corporate Headquarters
18880 Homestead Road
Cupertino, CA 95014
Toll-free: 800-728-1212
Tel: 408-863-9900
Fax: 408-863-4120
E-mail: info@rational.com
Web: www.rational.com

For International Offices: www.rational.com/worldwide