

Chapter 9

Pair Rotation: Communication, Knowledge Management and Training

Let's say you have a ten-person software development team and you want to use pair programming. One way of implementing it would be to pair everyone and have five pairs that would work together. We believe a preferred way of implementing pair programming would be to have the pairs switch partners periodically. The advantages of rotating are that the programmers learn more about the whole product by pairing with different people, they team with the person who can help them the most on a particular task, and communication increases significantly. Also, you don't have to worry as much about pairs becoming dysfunctional and the need to switch around all the pairs because two particular people just can't get along. The two people who don't get along can probably stand to work together in half-day or full-day increments.

In this chapter, we'll describe a pair rotation strategy and how many companies determine who pairs with whom and for how long. Then, we'll talk about the knowledge management, training, and communication advantages of pair rotation.

Pairing with the Right Partner

When people hear about rotating pairs, they often have the vision that this creates lots of jack-of-all-trades- masters-of-none generalists and discourages specialization. This section gives a simplified example of how to maintain specialists, spread knowledge around the team, and still enjoy the other benefits of pair programming. The example may seem rigid and prescribed. In actuality, pair rotation often occurs very casually without any formal schedule because team members learn how to maximize the benefits of pairing by choosing the right partner for the right situation. Also note that in a training scenario, it is best if the new person rotates around the group a bit, pairing with patient, mentoring types within the team, until they are assigned an area of specialty.

Consider a simple drink vending machine program. This program logically divides into four pieces: the GUI, the 'customer capabilities' logic, the 'machine maintenance' logic, and the data structures to run the whole program. Based on each employee's previous experience and expertise, they are assigned one of these four pieces:

Team Member	Responsibility
Christopher	GUI
Brian	Customer Capabilities
Kimberly	Machine Maintenance
Chelsea	Data Structures

Based on what these individuals are working on during any given day, they have a logical partner in the team member who has the expertise and/or owns the functionality most closely related to the task at hand. For example, consider the following pairings.

Task	Task Owner	Partner
GUI for “Buy Drink”	Christopher	Brian
GUI for “Add Inventory”	Christopher	Kimberly
GUI for “Add Recipe”	Christopher	Kimberly
Input Coins/Return Coins	Brian	Christopher
Select Drink	Brian	Chelsea
Ingredient Data Structure	Chelsea	Kimberly
Recipe Data Structure	Chelsea	Kimberly
Add Ingredients	Kimberly	Chelsea
Customer Analysis	Kimberly	Christopher

As the GUI expert, Christopher is the owner of every GUI task. His partner is the person whose code the GUI is “touching.” By pairing, Christopher gains the continuous reviewer, brainstorming partner. However, his partner is also there to share expertise on the inner workings of the program logic that his GUI interacts with. His partner also gains great insight on the GUI it needs to interact with. In turn, Christopher will then partner with Brian and Kimberly when they work on their program logic for customer capabilities and machine maintenance. Brian and Kimberly will then own the task and Christopher will be the partner. All gain knowledge on how to best have their respective code interact while maintaining their area of specialty. All gain by having the expert in the area their code “touches” sitting right by their side. All broaden their project knowledge.

You will notice that Christopher (GUI) and Chelsea (data structures) never pair and that Kimberly (machine maintenance) and Brian (customer capabilities) never pair. This is because their code does not interact directly with each other, so they are not “logical” partners. If their code does interact in any way, they should pair with each other for those tasks. Alternately, Christopher/Chelsea and/or Kimberly/Brian may pair specifically for the purpose of broadening their project knowledge.

Partner Assigning Logistics

So how do these dynamic pairings get assigned? We will describe two very different ways; many variations occur. There is one common denominator, though. In order for pair rotation to occur, project responsibilities need to be broken down to a manageable task of less than a week. Especially in large projects, team members could be assigned functionality that might take 3 to 6 months to develop. With pair rotation, this 3 to 6 months worth of functionality would need to be broken down into smaller half-day to one-week chunks. Then, the owner can recruit partners for these smaller chunks.

Before describing some ways to assign partners, we will address a common area of resistance to dynamic pairing. Christopher says, “When I said it would take me three months to develop the GUI, I didn’t know I was going to have to spend some of my time helping Kimberly and Brian do their code. I don’t have enough time to help them and still get my own work done!” The only way to get over this resistance is to give it a try.

Others might also wish to develop strong bonds with just one or a few members of the team and not rotate around the whole team. As long as the “rotating team” is not larger than ten, trust and strong bonds can develop because you will be pairing with each other quite often.

The two ways that we’ve observed successful pair assignments are now discussed:

- **Short, daily meeting.** Short, daily meetings are a practice of XP and SCRUM, where the entire team stands up for a short 10-15 minute meeting (this keeps the meeting short, because no one can sit down until the meeting is over). The focus of attention goes around the room as everyone on the team briefly describes what they were able to accomplish the previous day, what problems they’ve encountered, and what they need to work on this particular day. Depending on what problems the person is facing or what the person hopes to accomplish that day, the appropriate team member in the room might say, “I’ll pair with you on that today” or “I had that problem once too. This morning, I can show you what I did to solve it.” After everyone in the room has spoken, hopefully everyone knows what they are doing that day and who their partner is. If not, the manager or team leader can assign the most appropriate pairings from those who are left. (If you recall, our initial scenario in Chapter One was an example of one of these meetings.)

Pair programming aside, these short daily, meetings are very beneficial for team communication. Within this short, 15 minute period, the team is synchronized and ready to go. These “Daily Meetings” are a published organizational pattern. This pattern states, “A short daily meeting is an efficient way of transmitting information to the entire team with the minimum communication overhead. This helps overcome some of the tunnel vision problems. (Coplien 2002)”

- **Just say yes.** Another method of assigning pairs is that a task owner asks the appropriate team member to pair with them for a task. The team member is not allowed to say no. The only question, then, is *when* they will do it, which is a scheduling issue. Whoever asks first goes first. However, if you give help, it is fair to expect help in return (Newkirk and Martin 2002).

Pair Rotation and Knowledge Management

Our knowledge has legs – it walks home everyday. – Leif Edvinsson¹

Good knowledge management techniques are important for employers and employees alike. Truly, it is not advantageous to have all knowledge in any area of a system known to only one. For the employer, this is very risky because the loss of one or a few individuals can be debilitating. And, employees like to take some time off without being bothered with technical questions or problems. By rotating pairs, knowledge is disseminated. Employers reduce risk and employees can relax when out of the office.

Much of the recent investment in knowledge management has focused on using the Internet and creating databases to allow professionals to share documents or compare data. Invaluable knowledge can also be distributed throughout an organization simply by humans sharing insights and ideas with other humans. Pair programming is a mechanism for encouraging this valuable exchange. Through the use of pair programming, organizations can create Communities of Practice (CoP), or a group of professionals that develop bonds between each other because they have jointly solved problems together (McDermott 2000).

¹ Brain of the Year '98, Director of Intellectual Capital, Skandia Insurance Co., quoted in (Basili et al, 2001)

Members of a CoP develop a common sense of purpose and a desire to share work-related knowledge and experience. CoP members gain knowledge from each other that extends far beyond “book learning.”

As programmers rotate among the group, they get the chance to get to know many on their team more personally. This familiarity serves to break down many communication barriers. Team members find each other much more approachable. They will struggle with questions or lack of information for less time before getting themselves out of their chair and going ask the right person a question – because they know that person quite well. CoP enables person-to-person sharing of tacit knowledge, ideas and insights that are not documented and are hard to articulate; sharing of tacit knowledge takes place daily in the normal course of the programmers’ day.

James Euchner, vice-president of Nynex’s research and development department, reports that CoPs allowed them to reduce the number of days to set up data services for customers from 17 to 3 days. (Stewart 1996) This type of efficiency gain explains why a pair of programmers can complete a programming task in approximately half the time as one solo programmer.

Alistair Cockburn shares this story from his interview files (Cockburn and Williams 2001):

“When I arrived, I saw a disheartening sight: Bill didn’t have a team; he had a random collection of six bright, talented individuals who didn’t work together. They didn’t sit near each other. They didn’t even like each other. Here is a scene from a weekly staff meeting:

‘Let’s talk about pair-programming. (benefits of pair-programming enumerated) <pause> Therefore, pair-programming is mandatory. All production code must be written with a partner present.’

<An awkward silence descends. Furtive eye glances are exchanged.>

...

Some of the first paired sessions went smoothly. Other sessions were awkward. Communication was serial and parsimonious. I handheld these guys by becoming a third wheel. I encouraged the developers to think out loud (what Ward Cunningham calls reflective articulation). This did the trick. They actually began to work together, not just take turns coding.

After about a week, I noticed a remarkable phenomenon. The developers were talking to each other. As people. You really would have to have been there at the beginning to appreciate this. Anyway, I noticed them having real conversations. And laughing. They actually began to enjoy and trust each other.

Within several weeks, they became a real team.”

It is through “becoming a real team” that pair programming encourages the creation of CoP.

The knowledge management capabilities of pair programming can be summarized by the reflection of Kurt, a manager at a large corporation in North Carolina via personal communication with Laurie. Kurt hired RoleModel Software to develop software for a strategic business application. RoleModel assigned Duff and Michael to be the primary developers. These programmers followed all the XP practices, including pair programming. After observing them and seeing their remarkable results, Kurt likened pair programming and pair rotation to bees pollinating an orchard. “The pollination of flowers is a ‘lucky accident’ that happens when bees and other animals accidentally brush against the stomas leaving pollen

grains behind.²” With pair programming, the nuggets of knowledge left behind with pairing partners are indeed “lucky accidents” of the technique. In addition, both partners can then propagate the nugget throughout the team.

When Duff and Michael started working for Kurt, Michael had web-programming experience and Duff did not. Duff had much more experience than Michael in building business applications. As a pair, they very quickly developed a web-application with virtually no ramp-up time. Duff learned web programming through Michael and was productive without much delay. Michael leaned on Duff to identify and deliver the most critical business value. Kurt also reports that from time to time either Duff or Michael was not available. RoleModel substituted another programmer, who was part of its community, in their place. The surrogate programmer paired and again was essentially productive immediately. Pair rotation creates a shared mindset and a symbiotic culture that simply cannot be duplicated when each person works alone on their own task.

Pair Rotation and Training

Not only do you have experience walking out the door, you have inexperience walking in the door.

--

Scott Eliot³

Adding staff to a project consumes valuable resources of existing team members. Traditionally, new folks are indoctrinated into a team by spending time with a senior person on the team. Perhaps, the person spends a few days giving them an introduction to the system, showing them where various things are – everything from the cafeteria to the manuals. Then, they are often left on their own to perform some task, albeit not an overly complex one, to get started. During this time, he or she tries their best to figure out what’s going on and how things work. Ultimately if they can’t figure something out, no matter how hard they try, they will get up and interrupt someone to ask for help. Eventually, team members come to be far less reliant on others. However, this interruption-based approach can be a “sink or swim” form of training for the newbie and burdensome to those who are interrupted.

Alternately, we have found pairing to be a very effective and efficient training technique. When a new person starts with the team, he or she pairs with an existing team member. When the person first arrives, it is best if the partner is a patient, mentoring senior team member, such as the team coach or leader. The mentor must be willing to communicate and to give up some of their own productivity to bring the new person on board. Within a few days to a week, the new person can start to rotate around the team to learn the various aspects of the system and to get to know the team members themselves. This also distributes the burden of training throughout the team. However, each person who works with the newbie must realize that he or she will have some form of reduction in their own productivity. Each must spend time to slow down and explain what he or she is doing and how the system works, or the training-via-pairing session will be useless and will be very, very frustrating for the new person.

Laurie’s graduate student, Anuja Shukla, did some research about the effect of pair programming on training time. She surveyed software professionals on the differences in training software developers with and without pair programming. She received responses from 24 software professionals. Utilizing a model developed by Richard Stutzke (Stutzke 1994), the survey centered around the differences in

² From <http://www.thebeeworks.com/pollinators/index.html>

³ Director of Knowledge Management Product Groups, Lotus (KMWorld 2001), quoted in (Basili et. al. 2001)

assimilation time (which was defined as the time spent until they become “independently” productive to own their own task without relying heavily on other team members) and mentoring time (the percentage of the mentor’s day that is dedicated to teaching the mentee). Using Stutzke’s model, the effort expended on training is given by the following formula:

$$\text{Training Effort} = \text{Assimilation Time} * (1 + \text{Mentoring Time})$$

Based on the survey, she found the results shown in the table below, which indicate that training effort can be reduced by a factor of two through the use of pair programming and pair rotation!

	With Pair Programming	Without Pair Programming
Assimilation Time (work days)	13	28
Mentoring Time (%)	25	32
Training Effort (work days)	17	37

Alternatively, Alistair Cockburn (Cockburn 1998) recommends a Day Care arrangement for training where the development team is split into a “progress team” and a “training team.” The progress team is responsible for 85-95% of the system and the training team is expected to deliver only 5-15% of the system. People are transferred from the training team to the progress team as they are ready. Experts are selected from the progress team to aid in mentoring the training team. These experts must have the personality, inclination, and background to successfully train novices. Using this model, the mentoring experts can take turns rotating around the training team, pairing with each in turn. Additionally, members of the training team that are not completely new can pair with each other with better results than if each struggled alone.

Reprisal: Pair Rotation

If each person in the team paired with only one other person on the team, many benefits would be realized – quality gains, communication gains, teamwork gains, trust building, etc. However, even more can be gained by dynamically assigning the partners based on the task at hand. Two of these additional gains relate to knowledge management and to training. Pairing is an excellent knowledge management strategy because it breaks down communication barriers between teammates. They get to know each other and each seems more approachable when a question arises. Also, by pairing with many different people, developers learn more about many aspects of the system and more about the programming and tool techniques of many different people. Pair rotation also reduces the training time to assimilate a new team member and distributes the burden of training throughout the team.

We highly recommend you work out the logistics of pair rotation to enjoy these added benefits.

References:

- Basili, V., Costa, P., Lindvall, M., Mendonca, M., Seaman, C., Tesoriero, R., and Zelkowitz, M. (2001), *An Experience Management System for a Software Engineering Research Organization*, NASA Software Engineering Workshop.
- Beck, K. (2000). *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, Massachusetts.

- Cockburn, A. (1998), *Surviving Object-Oriented Projects: A Manager's Guide*, Addison-Wesley, Reading, Massachusetts.
- Cockburn, A., and Williams, L. (2001). "The Costs and Benefits of Pair Programming." *Extreme Programming Examined*, G. Succi and M. Marchesi, eds., Addison Wesley, Boston, MA, pp. 223-248.
- Copien, J. O. (2002), *Daily Meeting*, <http://i44pc48.info.uni-karlsruhe.de/cgi-bin/OrgPatterns?DailyMeeting>
- McDermott, R (2000), "Knowing in Community: 10 Critical Success Factors in Building Communities of Practice," <http://www.co-i-l.com/coil/knowledge-garden/cop/knowing.shtml>.
- Newkirk, J. and Martin, R. C., (2002) *Extreme Programming in Practice*, Addison-Wesley, Reading, Massachusetts.
- Stewart, T. (1996), "The Invisible Keys to Success," in *Fortune*, vol. 8/6.
- Stutzke, R. (1994), *A Mathematical Expression of Brooks' Law*, Ninth International Forum on COCOMO and Cost Modeling, Los Angeles, CA.