

# Paired Programming & Personality Traits

**Andrew J. Dick**  
Red Hook Group  
#708 33 Wood Street  
Toronto  
Ontario, Canada, M4Y 2P8  
+011 416 877 2249  
andrew@redhookgroup.com

**Bryan Zarnett**  
Red Hook Group  
#708 33 Wood Street  
Toronto  
Ontario, Canada, M4Y 2P8  
+011 416 877 2249  
b@redhookgroup.com

## ABSTRACT

Paired programming is an essential element of extreme programming – a methodology comprised of applying best practices to software development. Not all developers are suited for paired development however, and care should be taken when building a team for extreme programming. The team members should be selected with personality traits that are beneficial to paired programming. These personality traits can be determined through various interview techniques and the corresponding behavioral responses of the candidates.

## Keywords

Extreme programming, paired programming, personality traits, XP

## 1 INTRODUCTION

Extreme programming (XP) is a developmental methodology that enables rapid architectural development through the application of a collection of best practices. XP is dependant upon the combined use of these best practices because each practice provides a counterbalance to any detrimental characteristics of using a single best practice by itself. Hence, it is important to ensure that each best practice is utilized fully.

One of these best practices is paired programming in which all production code is written by two developers sitting at one machine [1]. Paired programming however, is often the single practice that draws the ire of most XP critics [2]. Commonly the criticism is leveled that two developers working together cannot equal the productivity of the same two developers working in parallel. However, several studies of paired programming have demonstrated its merits [2][3]. Significantly, the idea that two developers working together on the same task will double the development cost is erroneous. In fact, it has been statistically shown that paired programming costs approximately 15% more time than traditional programming [2]. When one factors in the benefits of a simpler design and that errors are detected and corrected earlier in the developmental cycle when they are orders of magnitude less costly, pair programming can be viewed as a significantly less expensive method of developing software [1][2][3].

Effective paired programming is difficult to achieve and requires a careful cultivation of personalities within the development team. This paper will briefly discuss the importance of paired programming within the XP framework. The authors will then discuss their experiences with paired programming in relation to the observed personalities of the developers involved. Finally, the paper will discuss those characteristics deemed beneficial to the successful pairing of developers and as a final point will explore methods of selecting developers that possess these characteristics.

## 2 PAIRED PROGRAMMING ELEMENTS

Paired programming requires all production code to be written by two developers at a single machine. Each developer has a role within this team: the first is responsible for the typing of code (the driver); the second is responsible for strategizing and reviewing the problem currently being worked on (the navigator). These roles can change dynamically as the keyboard is passed back and forth. Moreover, the combinations of partners themselves can change on a daily basis depending on the current task. This dynamic interaction provides an ideal environment for inter-developer mentoring within the team [3].

XP critics often claim that paired programming would be too slow or that the developers would be in constant disagreement. These detrimental effects of paired programming are counterbalanced by other XP best practices such as:

1. The use of a common metaphor to describe the problem, which reduces misunderstandings related to the problem domain.
2. A simple design that reduces the complexity that causes confusion when the design requires modification.
3. Unit tests which provide a clear understanding of each task's required function and subsequently allow the developers to proceed with bold changes to the design without impacting the behaviour.

4. Coding standards that eliminate problems that arise from disparate coding styles of different developers.

In the same manner, paired programming provides a counterbalance to the adverse effects of several other XP best practices:

1. Refactoring (the incremental improvement of a component's existing design without changing the external behaviour [4]) can be performed more rapidly due to the ongoing code reviews of the navigator.
2. A simple design can be produced more effectively because of the ongoing interaction between the two developers.
3. The concept of collective code ownership is enhanced because the watchful eye of the navigator minimizes code breakage that might occur when a developer must modify an unfamiliar software module.
4. Unit testing is accomplished faster and more thoroughly due to the vigilance of the developers on each other to fully define and implement the requisite unit tests for a task.

Paired programming is clearly an integral part of the XP methodology and consequently paired programming has a significant role in the proper application of the various best practices that comprise XP.

### 3 PAIRED PROGRAMMING OBSERVATIONS

The authors had the experience of developing a financial application with a development team built expressly for exploring the XP methodology. The authors had previously experienced the merits of paired programming, both with each other and with other developers. The authors had successfully practiced pair programming in a traditional development environment before learning about XP. Subsequently, the concept of pair programming was easy to accept as one of the basic tenets of XP, having experienced the benefits that it provides, including better designs, fewer errors and a more enjoyable working environment.

The authors interviewed each candidate using a paired approach and selected the team based on aptitude and attitude as opposed to a list of technical experiences. Although some groups advocate allowing the entire team to interview prospective candidates, the authors felt that the team's lack of XP experience would prevent them from contributing to the evaluation in a meaningful way. The final team consisted of two senior developers (greater than two years of development experience) and four junior developers (less than one year of development experience). In addition, the authors performed the XP roles of coach and tracker [1,5]. The development environment consisted

of standard L-shaped desks with sufficient room for the developers to sit side by side. Although not an ideal layout compared to tables, both authors had successfully paired in the same environment.

Each developer signed up for a set of tasks for an iteration based on their personal velocity from the previous three-week iteration. The developers estimated their initial iteration velocity based on their respective experience levels. The development team was encouraged to switch partners as tasks were completed to increase inter-partner mentoring. Each day began with a quick stand-up meeting to review the team's status, select pairings and discuss any unresolved problems.

The first problem noted was that the dynamic interchange of roles (driver and navigator) was not taking place. One developer would invariably drive and the second developer's attention would drift away. This behaviour continued in spite of frequent intervention by the coach (team lead). Interestingly this behaviour even occurred with some of the junior-senior pairings as well as the junior-junior pairings. The lack of an attentive navigator allowed the pair to divert from the desired path of developing a simple design that met the task's requirements.

The second problem was that the breakdown of each pair's interaction had an adverse effect on the desired mentoring within the team that is necessary to properly utilize the XP practices. The lack of knowledge transfer subsequently meant that the team's developmental speed (velocity) was not increasing at the expected rate. To counteract this problem, paired programming was temporarily eliminated after the fourth iteration so that each developer was responsible for his or her own development tasks for the current iteration. The result of this directive was an immediate increase in inter-developer communication and the resultant knowledge transfer.

The developers finished the remaining two iterations of the project's first phase working on their own tasks. A significant portion of their time however, was spent discussing design directions and debugging problems in pairs. Unfortunately, the client did not finance phase two of the project for business reasons, which prevented the opportunity of observing the team's subsequent paired programming performance.

One possible explanation for the noted observations is that when two junior developers are paired, each assumes the other developer understands the aspects of the current task and is subsequently apprehensive about asking questions due to the fear of appearing ignorant. The navigator is required to make continual course corrections for the best design to evolve [3]. Consequently, the design appeared to continually drift from the expected path. When the developers were separated into the traditional single

developer environment, they were each forced to face the limits of their knowledge. Hence, each developer had to ask questions to ensure their comprehension of the task as opposed to assuming their partner understood the problem. Evidence of this hypothesis was the observed surge in communication between the developers when paired programming was discontinued.

The authors explored the possible reasons why paired programming worked for them but not for the development team. The conclusion was that the authors shared personality traits that were lacking to various degrees in the development team. These traits are examined in the following section.

#### 4 PERSONALITY TRAITS

The noted observation that some developers paired more effectively led to the conclusion that certain personality traits are beneficial for paired programming. This section will discuss these traits and their impact on the dynamics of paired programming.

##### Communication

The most obvious personality trait that is essential for success in paired programming is *communication*. Communication is clearly valuable in any development environment, but in a paired programming environment, the ability to communicate is crucial. The pair must be able to communicate effectively in order to properly analyze the merits of different design directions, discuss strategy for testing, and correct errors that are caught by the ever-watchful eye of the navigator. In essence, a lack of communication between developers will diminish the potential for the pair to work in harmony. A lack of communication was observed in our project between the driver and navigator resulting in substandard design.

##### Comfortable

The developer must be *comfortable* working directly with other people. Pairs that are not comfortable with one another will be reluctant to offer suggestions due to the possibility (real or imagined) of being ridiculed. The fear of appearing stupid decreases the number of bold proposals and ideas that are an essential element of XP. Conversely, pairs that are comfortable with each other will offer intriguing suggestions and interesting strategies with the knowledge that their counterpart feels comfortable doing the same [3]. Developers may also be uncomfortable working with individuals that have a different standard of work ethic and professional etiquette (e.g. personal hygiene).

##### Confidence

The developers must be *confident* in their abilities as well as their competence as a team. XP requires the developers to manipulate both design and code throughout the production code base. The pair must be confident in their abilities to successfully add new functionality and

conversely to judge where existing unused functionality can safely be removed. Pairs that lack confidence will maneuver around dead code and problem areas rather than to continually simplify the design by removing the dead code and resolving any problems encountered. Lack of confidence was illustrated by the pairs of junior developers through their reluctance to refactor the production code base.

##### Compromise

The ability to *compromise* completes the quartet of paired programming personality traits. Developers that are too confident often lack the ability to compromise and become argumentative when paired. The primary purpose of pairing is to work towards the best design possible, regardless of from where or from whom the design originated. Good development pairs can discuss suggestions without bias concerning its origin and deliberate solely on the merits of the suggestion itself. From our observations, the senior developers were unwilling to compromise their design ideals and subsequently ignored their junior partner, which reduced mentoring.

A team comprised of developers that possess these four personality traits is much more likely to fulfill the potential that XP offers. Although several other traits are invaluable to XP in general (such as creativity or attention to detail), the four listed personality traits represent those necessary to pair program successfully.

#### 5 INTERVIEW TECHNIQUES

The four desirable personality traits – communication, comfortableness within a team, confidence in one's self and the ability to compromise – can usually be gauged during the interview process. The authors often found that developers that would have been suitable for a traditional programming environment were not necessarily desirable for the XP development environment. For example, introverted developers can flourish in a traditional development environment where interpersonal communication is not mandatory. Similarly, an extremely uncompromising developer that is technically competent can thrive in a development environment where they have absolute control over an individual software module.

The authors reexamined their interview techniques to ascertain if the desired personality traits could be assessed in prospective development candidates. The remainder of this section discusses approaches that may be used to gauge a candidate's suitability for paired programming.

##### Communication

Determining a candidate's ability to communicate during an interview is not difficult. A key indicator of a candidate's capacity for communication is their willingness to elaborate on their interview answers beyond simple sentences. Although candidates can be expected to be

nervous during an interview, a candidate that provides excessive information should be avoided to the same extent as the candidate who provides too little information [2]. An ideal candidate should possess the ability to explain each answer succinctly.

### **Comfortable**

It is possible to gauge a prospective candidate's ability to be comfortable working within a pair by providing them with the opportunity to discuss their answers. An extrovert candidate will take advantage of the chance to discuss their answers in depth. Admittedly, it usually takes time for two people to become comfortable working together [3], however, a candidate that is personable during an interview situation will predictably act in a similar manner within the team environment. A candidate that cannot open up sufficiently during the interview is unlikely to succeed in becoming comfortable with the number of partners they will experience within the dynamic pairing of extreme programming.

### **Confidence**

A candidate's confidence can be determined through questions that require analytical answers and problem solving. Asking a candidate to solve a hypothetical non-technical problem provides the opportunity for the candidate to demonstrate their confidence in breaking down a problem, resolving any ambiguities through discussion and explaining their proposed solution (i.e. locating the most economical path between two arbitrary points in a weighted, directed graph). Candidates that lack confidence will usually be unable to adequately answer questions of this nature; conversely, candidates that possess too much confidence will often question the need for performing the exercise at all.

### **Compromise**

A candidate's willingness to compromise can be difficult to ascertain during an interview where they will be expected to be presenting their best behaviour. However, the discussion of the principle of adherence to a common coding convention can often effectively illustrate this trait. After explaining the merits of common coding conventions, (e.g. allowing easy comprehension of all production code regardless of the author) some candidates will continue to argue that their coding style is impeccable and demonstrate a strong resistance to conformance. Regardless of an argumentative candidate's technical skills (which are often high), this candidate should be avoided for paired programming situations due to the predictable conflicts that will arise over trivial issues. Conversely, candidates that express an understanding of the merits of a common code convention and the benefits of compromise are obviously highly desirable.

Perhaps the best way to gauge a candidate's potential to pair program is to integrate them into the team for half a day or more. Although the cost in time might appear

prohibitive at first, the resulting feedback could far outweigh the ramifications of a poor hiring selection. This method assumes of course that a development team that can pair program already exists. Alternatively, the entire development team may exist already, but paired programming is only being introduced. The solution to this problem is briefly discussed in Kent Beck's second book *Planning eXtreme programming*. Essentially, a small core of pair programmers must be cultivated and then slowly enlarged. The interview techniques described above can be applied to the process of selecting which team members should form the initial core of pair programmers.

## **6 SUMMARY**

Extreme programming is comprised of best practices that work well together but are ineffective on their own. Consequently, for extreme programming to be effective, it is imperative that all the elemental practices are applied properly. One of these practices is paired programming which involves two programmers developing all production code on one machine. For two developers to pair program effectively, several personality traits are beneficial: effective communication, comfortableness working with one another, confidence in one's abilities and the ability to compromise. These personality traits allow the pair to collaborate effectively to realize the simplest design that fulfils each task's requirements.

Interviews with prospective candidates can often be geared towards determining whether the candidate has the aptitude for paired programming. Building a development team with the necessary personality traits that are beneficial to pair programming will result in greater success with extreme programming than a team built based on technical skills alone.

## **7 INFORMATION AND QUESTIONS**

For more information, contact [info@redhookgroup.com](mailto:info@redhookgroup.com) or visit <http://www.redhookgroup.com>.

## **REFERENCES**

1. Beck, K. *Extreme programming explained: embrace change*. Reading, Mass.: Addison-Wesley, 2000.
2. Williams, L., *et al.*, "Strengthening the Case for Pair Programming." In *IEEE Software*. 2000.
3. Cockburn, A. and L. Williams. "The Costs and Benefits of Pair Programming." In *Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering*. Italy, 2000.
4. Fowler, M. *Refactoring: improving the design of existing code*. Reading, Mass.: Addison-Wesley, 1999.
5. Beck, K. and M. Fowler. *Planning eXtreme programming*. Reading, Mass.: Addison-Wesley, 2001.