Should NASA Embrace Agile Processes?

Jefferey Smith, Tim Menzies
Lane Department of Computer Science
West Virginia University
PO Box 6109, Morgantown
WV, 26506-6109, USA;
jefferey@jeffereysmith.com, tim@menzies.com

Abstract

As a growing number of NASA software projects tend to stray from the conventional waterfall method of software development, alternate development paradigms need to be explored and assessed. This paper examines one such paradigm, Agile Processes; and more specifically, the economics of one its key practices, pair programming. It will be shown that this core practice is only demonstrably useful in a very small number of cases.

1 Introduction

A growing trend among NASA software projects is the tendency for software development to move away from the traditional software development paradigms. Previously, NASA software has been developed by large software contractor organizations with mature and traditional process models. While this continues to be NASA's main mode of software development, there are an increasing number of cases where an alternative approach is taken. For example, the telescope controller for a current near-earth satellite mission was developed by a university-based consortium that used numerous heritage products and a loose consortium of sub-contractors. This team produced useful code, but not a fully developed traditional requirements document.

The economic motivation for such non-traditional development is clear: more science can be conducted for less dollars. However, the safety implications such as the potential for loss-of-mission is unclear. In situations such as these, portions of code are being developed *before* the requirements are written. This then leads to requirements being written based upon the code; instead of the other way around. These projects are clearly no longer following the traditional waterfall model of software development:

```
Requirements ->
        Design ->
            Coding ->
                Testing ->
                    Maintenance
```

In this light, alternate software development paradigms should be explored as possible replacements for the waterfall method.

One such alternate paradigm is *Agile Process* (AP) development. AP is "iterative, incremental, self-organizing, and emergence" [5]. According to the Agile Manifesto, AP is based on the idea of "uncovering better ways of developing software by doing it and helping others do it [3]." In his article, "Get Ready for Agile Methods, With Care," Barry Boehm states that the primary difference between AP and conventional methods is that "[AP] methods derive much of their agility by relying on the tacit knowledge embodied in the team, rather than writing the knowledge down in plans [2]." He goes on to discuss that while AP methods risk making mistakes that could have been found by planning practices used by conventional methods, conventional methods risk "that rapid change will make the plans obsolete or very expensive to keep up to date [2]." While AP may not be appropriate in every situation, there appears to exist cases in which AP may be advantageous to conventional methods.

Before beginning, it is important to stress the limits of our study. This work will comment specifically on an agile process practice called "pair programming". We will study pair programming because it is a core part of an agile process called "extreme programming", which is the most widely cited agile process in the literature. It will be shown that this core practice is only demonstrably useful in a very small number of cases. However, if another agile process

[0]Submitted to the 27th Annual IEEE/NASA Software Engineering Workshop, Greenbelt, MD, USA, December 5-6, 2002, Greenbelt Marriott. http://sel.gsfc.nasa.gov/website/27ieee.htm.

did not use pair programming, then the negative conclusions of this paper would not apply to that agile process.

2 Agile Processes / Extreme Programming

AP is an approach that has gained popularity in recent times. AP is a collection of software design practices and techniques that diverge from the heavily structured methodologies in favor of a less structured, more adaptive approach. According to its manifesto [3], AP values...

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

In addition to these values, AP is based upon a set of twelve principals:

1. The highest priority is to satisfy the customer through early and continuous delivery of valuable software;
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage;
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale;
4. Business people and developers must work together daily throughout the project;
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done;
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation;
7. Working software is the primary measure of progress;
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely;
9. Continuous attention to technical excellence and good design enhances agility;
10. Simplicity–the art of maximizing the amount of work not done–is essential;
11. The best architectures, requirements, and designs emerge from self-organizing teams;
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

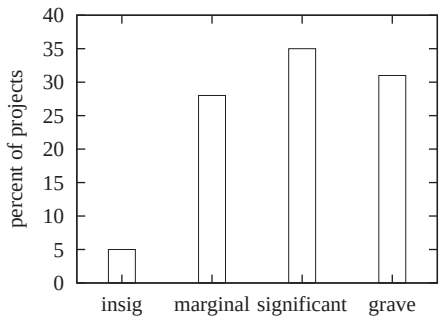

Figure 1. Ratio of different project error severities seen in 91 NASA IV&V projects 1996-2002.

There are many positive aspects of AP which might be attractive to the NASA organization:

- Continual interaction with the customer helps the project to become more tailored to the customer's needs
- Frequent releases allow the customer to see progress and working code more often
- Pair programming and continual testing leads to fewer errors

Despite these benefits, however, there are some aspects of AP that render it inappropriate for certain types of projects. The best example of this is its approach towards documentation. While some documentation is produced, AP does not include the full documentation provided by other methods. This is perfectly acceptable in many cases, but in the case of safety critical systems, this full documentation, like that which is produced as part of the waterfall method, is required.

This doesn't meant that AP should be abandoned. It simply means that AP is not an acceptable choice in certain situations. Figure 1 shows that out of 91 NASA IV&V projects, more than one third of these projects were not of high criticality (grave or significant). For projects such as these, AP should still be explored for possible use.

A critique of AP is complicated by the broad and diverse nature of the AP movement. However, one of the most popular AP approaches, Kent Beck's *extreme programming*, has been thoroughly specified; and therefore, it is easier to study [1]. According to Beck, extreme programming has the twelve key practices shown in Figure 2. The precise definition of all these practices are beyond scope of this paper. However, note the key role of one of the practices: *pair programming* (PP). PP is the concept of two developers working together at a single machine, designing and writing code cooperatively. Figure 2 shows the connection between XP

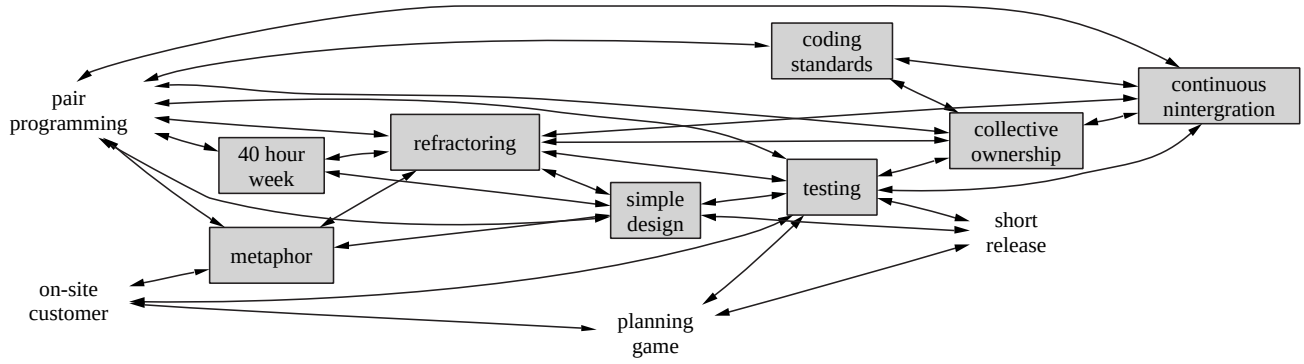

Figure 2. The connection between extreme programming practices. From [1]. Practices directly effected by PP are shaded.

programming practices; the shaded practices have a direct connection to PP. The centrality of PP to XP is evident from this diagram. If PP is removed, it would directly affect all of the shaded practices.

AP proponents claim that by working in pairs, developers produce code at a faster rate, and with fewer errors. But the idea of developer pairs leads to two scenarios, each with their own issues:

- **Pool:** If additional developers are added to a project from a pool of developers to create the developer pairs, then does the time/error advantage outweigh the additional developer costs?
- **NoPool:** If additional developers are not added, and instead, the current developers are divided into groups of two, then does the time/error advantage outweigh the additional time needed to write code that results from the fewer number of tasks that can be worked on at one time?

To answer these questions, the economic effects of PP on a software project need to be examined, and compared to the outcome of conventional methods.

3 Müller/Padberg Study

A study of this nature has been conducted previously. In their paper, "Extreme Programming from an Engineering Economics Viewpoint," Müller and Padberg compare the economics of PP with those of conventional methods [4]. This section examines their work. The next section discusses our concerns with their methods, and the procedure we followed as a result.

3.1 Equations

In their paper, Müller and Padberg present an economic model of software projects for both PP and conventional methods. Their model is based on the Net Present Value (NPV) of the software project, which is calculated and compared for both programming methods.

The NPV of the projects is found using Equation 1.

$$NPV = \frac{AssetValue}{(1 + DiscountRate)^{\frac{DevTime}{12}}} \quad (1)$$

The $AssetValue$ and $DiscountRate$ are the same for both projects. The $DiscountRate$ is used to simulate time to market: a lower discount rate represents a longer time to market, and a higher discount rate represents a faster time to market. In NASA's case, a fast time to market might be the equivalent to a rapidly approaching launch date.

The equation for the development time, $DevTime$, differs for the two development methods. Since the PP method results in fewer errors, an allowance must be made in the development time for the conventional method to allow for the elimination of an equivalent number of errors. This additional time for quality assurance is the $QATime$. The $DevTime$ equation for the conventional method is:

$$DevTime = QATime + \frac{ProductSize}{DeveloperProductivity * NumberOfDevelopers} \quad (2)$$

And the $DevTime$ equation for the PP method is:

$$DevTime = \frac{1}{DeveloperProductivity} * \frac{1}{NumberOfPairs} * \frac{ProductSize}{100\% + PairSpeedAdvantage} \quad (3)$$

where

Parameter	
Developer Productivity	350 LOC/month
Developer Monthly Hours	135 hours/month
Defects per KLOC	100
Defects Eliminated by Conventional Processes	70%
Product Size	16,800 LOC
Developer Salary	$50,000
Project Lead Salary	$60,000
Asset Value	$1,000,000

Table 1. Fixed parameters used by Müller and Padberg

$$NumberOfPairs = \frac{NumberOfDevelopers}{2} \quad (4)$$

The $QATime$ require three calculations. First, the number of defects left in a typical project must be calculated using Equation 5.

$$DefectsLeft = ProductSize * \frac{DPKLOC}{1000} * (1 - DNE) \quad (5)$$

$DPKLOC$ is the number of defects per thousand lines of code, and DNE is the percent of defects not eliminated by conventional techniques (1 - Defects Eliminated by Conventional Processes). Once $DefectsLeft$ has been calculated, the next step is to find the $DefectDifference$. This is simply the $DefectsLeft$ multiplied by the $PairDefectAdvantage$, which is the percent of defects PP eliminates that conventional programming does not.

$$DefectDifference = DefectsLeft * PairDefectAdvantage \quad (6)$$

The final step is to calculate the time required for conventional developers to remove these extra defects. This is known as the $QATime$, and is found using Equation 7.

$$QATime = DefectDifference * \frac{DefectRemovalTime}{DeveloperMonthlyHours} * \frac{1}{NumberOfDevelopers} \quad (7)$$

The final calculation that is required for the project model, which is the same for both the conventional and the PP method, is the development cost, $DevCost$. $DevCost$ is found using Equation 8.

$$DevCost = \frac{DevTime}{12} * (NumberOfDevelopers * DeveloperSalary + ProjectLeadSalary) \quad (8)$$

Parameter						
PairSpeedAdvantage	10%	20%	30%	40%	50%	
PairDefectAdvantage	5%	10%	15%	20%	25%	30%
DefectRemovalTime	5h	10h	15h			
DiscountRate	0%	25%	50%	75%	100%	

Table 2. Parameters systematically varied by Müller and Padberg

3.2 Method (Müller/Padberg)

For their study, Müller and Padberg used a set of fixed parameters, see Table 1, and four parameters, that they considered to be key features, for which they systematically varied their values (Table 2).

They used these values to calculate the NPV for the conventional method and the PP method, and they did so for both in two situations:

- **NoPool:** The number of developers was fixed. In this situation, when the conventional method is using n developers, the PP method is using $\frac{n}{2}$ pairs.
- **Pool:** The number of developers is not fixed, as if there existed pool of developers to create pairs with. In this situation, when the conventional method is using n developers, the PP method is using n pairs, or $2n$ developers.

3.3 Results (Müller/Padberg)

First, Müller and Padberg found that PP is advantageous when the number of pairs is equivalent to the number of developers, as described in situation 2 above. In other words, n developers are more efficient than n/2 pairs. Therefore, as long as there exists additional tasks that can be assigned to individual developers, PP is not the best solution.

However, if the project cannot be subdivided beyond n tasks, and you have access to 2n developers, then PP has the potential to be beneficial in certain cases. Müller and Padberg found that PP is advantageous in this situation if three criteria are met:

1. the project is of small to medium size ($ProductSize$ is not large)
2. the project is of high quality ($AssetValue$ is high)
3. the need for a rapid time to market is present ($DiscountRate$ is high)

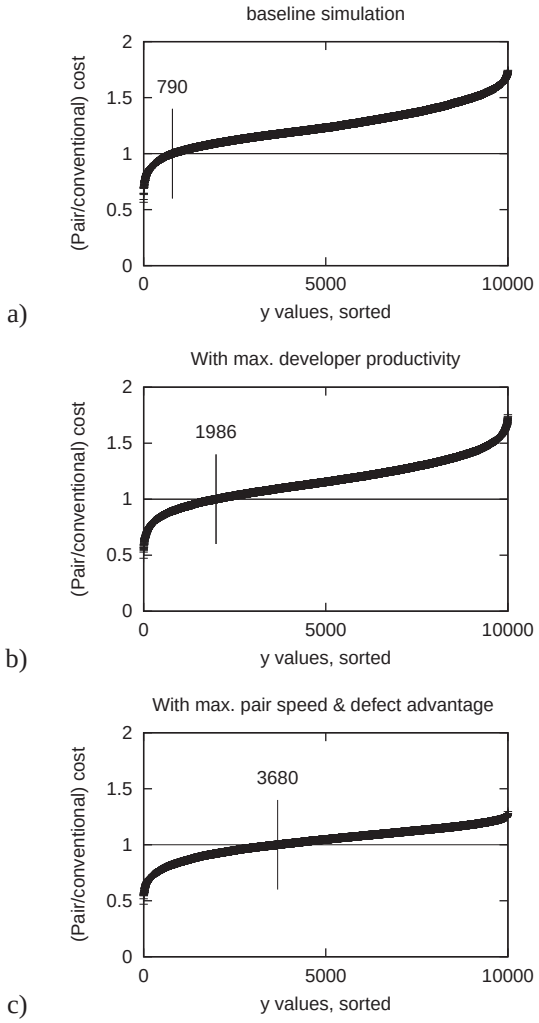

Figure 3. Raw data plots of a) completely random cases, b) cases with $DeveloperProductivity$ set to maximum (**T1**), and c) cases with $PairSpeedAdvantage$ and $PairDefectAdvantage$ set to maximum (**T2**). The vertical line indicates the point where PP is no longer advantageous

4 Smith/Menzies Study

After examining the work done by Müller and Padberg, we felt that a wider range of attributes could be explored. By only varying four parameters, a large number of the model's attributes were not being examined as possible factors in determining the advantages of PP.

4.1 Method (Smith/Menzies)

To correct this, the Müller and Padberg model was re-implemented, and random values within appropriate ranges (see Table 3) were generated for a larger set of attributes.

Parameter	Min	Max
PairSpeedAdvantage	10%	50%
PairDefectAdvantage	5%	30%
DefectRemovalTime (hours)	5	15
DiscountRate	0%	100%
ProductSize (LOC)	10000	250000
DeveloperSalary ($)	45000	65000
ProjectLeaderSalary ($)	60000	90000
AssetValue ($)	200000	2000000
DeveloperProductivity (LOC/month)	100	500
NumberofDevelopers	4	20
DefectsPerKLOC	4	100
DefectsNotEliminated	10%	80%

Table 3. Parameters and ranges used by Smith and Menzies

These attributes were then input into the model and, using Equation 9, the total cost was found for both AP and conventional development methods.

$$TotalCost = [(1 - \frac{1}{(1 + DiscountRate)^{\frac{DevTime}{12}}}) * AssetValue] + DevCost \quad (9)$$

Equation 9 is a modified version of Equation 1. This modified equation was used so that the resulting values would always be positive. This then allowed us to use the following ratio without worrying about the effects of negative numbers:

$$\frac{TotalCostofPP}{TotalCostofConventional} \quad (10)$$

This ratio was our means of comparison between the two methods.

The random generation and calculations were repeated for 10,000 cases, the results of each being recorded. These 10,000 cases were then used with a treatment learner to find the attributes that led to an advantage for PP.

Note that this approach generates very large data files. The *TAR2 treatment learner* is a tool for performing automatic sensitivity analysis of large logs looking for constraints to parameter ranges that can most *improve* or *degrade* the performance of a system [6]. *TAR2* performs an exhaustive search over the data, and is capable of discovering treatments not easily found by hand. In this study, *improve* or *degrade* was measured according to Equation 10; i.e. the impact on the ratio of PP's cost to the cost of conventional methods.

The treatment learner was applied to the test cases in two separate studies:

- **S1:** no possible treatments were ignored. That is, no attributes were ignored as possible factors that could be changed to influence the performance of PP.

- **S2:** select attributes that the authors considered to be unchangeable were ignored as possible treatments. These attributes were $PairSpeedAdvantage$, $PairDefectAdvantage$, and $DefectRemovalTime$.

Also, each of these studies were examined in the two situations, **NoPool** and **Pool**, described in section 3.2, for a total set of four studies: **S1/Pool**, **S1/NoPool**, **S2/Pool**, and **S2/NoPool**

In addition to these completely random tests, two additional tests were conducted with specific values set for various attributes. These were done to examine the effects of specific attributes about which the authors wanted to know more. These tests were as follows:

- **T1:** $DeveloperProductivity$ was set to the maximum value, 500 LOC/month, to see the distribution when developers are being the most productive.
- **T2:** $PairSpeedAdvantage$ and $PairDefectAdvantage$ were set to their maximum values, 50% and 30% respectively, to see the distribution when pairs are operating at their greatest rate of advantage over individuals.

4.2 Results (Smith/Menzies)

4.2.1 Summary (T1 and T2)

A summary of our results is shown in Figure 3. In those plots, pair programming is at an advantage over conventional approaches when $\frac{PP}{Conventional} > 1$.

Plot *a* in Figure 3 shows the output of our model from 10,000 runs across the distributions shown in Table 3. Note that in only 8% of the runs is there an advantage to PP.

Plot *b* in Figure 3 shows the output from **T1**. When $DeveloperProductivity$ was set to the maximum value, PP was advantageous in approximately 20% of the cases.

Plot *c* in Figure 3 shows the output from **T2**. From this plot it can be seen that even when the two attributes that have the most effect on the advantage of PP over conventional methods are at their highest values, still only about 37% of the cases resulted in an advantage for PP. This is the greatest advantage we found, even after using *TAR2* to exhaustively search all the attribute ranges.

4.2.2 Details

In both **S1/Pool** and **S1/NoPool** it was found that increasing $PairSpeedAdvantage$ and $PairDefectAdvantage$ was the way to most improve PP's advantage over conventional methods. This is not surprising since $PairSpeedAdvantage$ and $PairDefectAdvantage$ represent advantages that PP has over conventional methods. Another attribute property that showed some influence in increasing the advantage of PP in **S1/NoPool** was a high $DefectRemovalTime$. This too makes sense because conventional methods are prone to more defects, and therefore, a high $DefectRemovalTime$ would result in conventional developers working significantly more than developer pairs.

Considering that it may not be possible to simply change $PairSpeedAdvantage$, $PairDefectAdvantage$, and $DefectRemovalTime$ as needed, studies **S2/Pool** and **S2/NoPool** ignored these attributes as possible treatments, and looked for additional features that could lead to an advantage for PP. In **S2/Pool** it was found that there were three things things that significantly contributed to an advantage of PP over conventional methods:

1. A high $DiscountRate$ (> 0.42)
2. A small $ProductSize$ (10,000 to 60,000 LOC)
3. A high $AssetValue$ ($\approx$ \$1.6M to \$2.0M)

These are the same three factors that were found by Müller and Padberg when a pool of developers was present.

In **S2/NoPool**, no significant treatments were found. Therefore, when a pool of developers is not present, or when additional tasks are present as described by Müller and Padberg in section 3.3, PP does not have an advantage over conventional methods.

5 Conclusions

We offer two conclusions. Software development approaches such as AP/XP that do not precisely pre-specify the requirements are not indicated for NASA projects with significant or grave risks. Based on the IV&V experience 1996-2002 (Figure 1), this excludes around 64% of NASA software projects from using AP/XP.

For the remaining projects, our model has only endorsed AP/XP over conventional approaches in a relatively small and specialized set of cases; i.e. when:

- the project is relatively small
- an abundance of developers exists
- a rapid development time is needed

This is not to say that NASA should avoid AP/XP. However, this study concludes that a convincing case for AP/XP cannot be based on the factors modelled in this paper; i.e.

- increased productivity of pairs over conventional approaches
- productivity vs. cost
- lower error rates

Proponents of AP/XP must therefore base their case on other factors not modelled in this paper, such as (e.g.) increased performance in rapid changing environments, or decreased cost due to conventional requirements reworking to accommodate changes.

Finally, it is important to reiterate the limitations of our study. We focus primarily on the AP practice of pair programming. If an AP method does not include pair programming as part of its process, then the negative results of our study are not applicable.

Acknowledgements

This research was conducted at West Virginia University under NASA contract NCC2-0979. The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

[1] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison Wesley, 2000.

[2] B. Boehm. Get ready for agile methods, with care. *Computer*, 35(1):295–321, January 2002.

[3] M. B. et. al. Manifesto for agile software development, 2001. Available from `http://www.agilemanifesto.org/`.

[4] M. M. Mller and F. Padberg. Extreme programming from an engineering economics viewpoint. In *Proceedings of the Fourth International Workshop on Economics-Driven Software Engineering Research (EDSER)*, 2002.

[5] K. Schwaber, 2002. Quote from the First eWorkshop on Agile Methods. Available from `http://fc-md.umd.edu/projects/Agile/Summary/Summary1.htm`.

[6] T.Menzies and Y. Hu. The tar2 treatment learner, 2002. Available from `http://www.ece.ubc.ca/twiki/pub/Softeng/TreatmentLearner/intro.pdf`.