



Agile Software Development of Embedded Systems

Version : 1.0
Date : 2005.03.15
Pages : 6

Authors

Jari Vanhanen

Hantro

Status

Final

Confidentiality

Public

Agile Deliverable D.5.2.2

Experiences of using pair programming in the Eddy project

Abstract

This document reports about the usage of pair programming in the Eddy project during fall 2004 at Nokia Technology Platforms in Helsinki. The report is based on the metrics gathered by the project team during the project, and on a team interview and a survey done by the author of this report in the middle of the project on 17.11.2004.



ITEA

INFORMATION TECHNOLOGY
FOR EUROPEAN ADVANCEMENT

Content

1	Introduction	3
2	Results from the team interview.....	3
2.1	Doing pair programming	3
2.2	Effects of pair programming	3
3	Results from the inquiry.....	4
4	Metrics collected during the project	5
5	Summary.....	6

1 INTRODUCTION

This document reports about the usage of pair programming in the Eddy project during fall 2004 at Nokia Technology Platforms in Helsinki. The report is based on the metrics gathered by the project team during the project, and on a team interview and a survey done by the author of this report in the middle of the project on 17.11.2004.

The project team consisted of four developers, two master's thesis workers from HUT, and two developers from a sub contractor. They had 4 -10 years of programming experience and 1.5 - 4.0 years of experience with the technologies used in the project. Only one of the developers had used pair programming before the Eddy project and even he only about 1 month.

2 RESULTS FROM THE TEAM INTERVIEW

The chapter presents what the developers told about the details on how they used pair programming and their feelings about the effects of using pair programming.

2.1 Doing pair programming

Pair programming was used quite a lot. One reason at least partially forcing to the use of pair programming was that there were only three workstations for the four developers available. When all developers were present pair programming was used almost always and for all kinds of tasks.

Pair programming was considered especially useful for more complicated tasks. When doing simple "copy-paste" coding the navigator lost his interest in the pair work quite soon.

Pairs were formed, e.g., by casting lots. However, in the first iterations the same two people continued to pair the next day if the same work continued. Starting from the third iteration more emphasis was put on switching pairs each morning. This kind of regular swithing worked very well and it simplified forming the pairs, because in a small team the other way, i.e. swithing pairs only after a task ends requires the faster pair to wait until the other pair finishes their task.

In a typical pair programming session the partners first made some design together and then started programming. There were different opinions on the amount of design required before coding and this caused sometimes disagreement between the partners. When programming, the keyboard was not switched very often, only two to three times a day. In the beginning of the project switching was more frequent and the keyboard was even wreched from the other person's hands. Later switching typically happened when the other person took a break and left the workstation for a while. Communication during the pair programming sessions was continuous and there were no silent moments. If the other partner was already familiar with the problem, he wrote the code and simultaneously explained what he did. It felt impossible to act as the navigator if one did not know what the code did.

Test driven development, which extreme programming recommends to be used with pair programming, was applied using JUnit for almost all programming. However, different developers had different opinions on the required amount of the tests.

Pair programming required some learning from the developers. The first development tasks (spikes), which took about 8 hours per person, involved learning to do pair programming. The use of pair programming got a little bit looser later in the project. For example, developers did not necessarily sat at the same workstation all the time when programming. Especially when very simple things were programmed, the partner sometimes went to do other work or took a break.

2.2 Effects of pair programming

Pair programming probably did not greatly affect the amount of defects in the code, but the design might become more understandable. This was seen so that the next person(s) working with a certain piece of code understood better what the previous pair had done. When working with a pair, the navigator forced

the driver to stop writing when he no more could understand the code. Thus all code became written so that at least one other person (the navigator) was able to understand it. Only quite seldom the navigator found that the driver created a defect. The tests were more efficient in finding defects.

For most of the tasks the total effort when doing pair programming was considered higher than if the task were done alone. However, for complex tasks the use of pair programming might have had lower total effort. The amount of these complex tasks was quite small, but they were big tasks requiring about 50% of the total project effort.

Pair programming had a positive effect on knowledge transfer. The team studied the knowledge transfer themselves by evaluating their understanding of each component after each iteration. After the third iteration, where pairs were rotated more frequently, the minimum level of understanding increased. This happened even though the amount of code increased and the tasks were considerably more complex. All developers acquired quite a good understanding of the complex topics related to the domain.

The team spirit was very good, but the co-location was probably the main contributor for this. Pair programming probably accelerated the formation of good team spirit in the beginning of the project.

When doing pair programming, writing unit tests was probably more disciplined than when writing code alone. Additionally it probably affected the discipline in following the coding standard.

If the team could choose whether to use pair programming in the next project or not, they would prefer an environment where both pair programming and solo programming could be used. Then they could use pair programming for those tasks they consider it useful.

3 RESULTS FROM THE INQUIRY

The developers were asked to prioritize different alternatives for how pair programming should be used in a project like the one they were doing. The answers are shown in Table 1. Quite naturally, the developers preferred being able to decide themselves when to use pair programming. They also found important that management supports the use of pair programming. Nobody considered that pair programming should not be used at all.

Table 1. Preferences on how pair programming should be used. Scale: 1=best alternative, 4=worst alternative.

How PP should be used in this kind of a project?	A	B	C	D
Force the usage of PP for most programming.	3	2	3	3
Allow developers decide when to use PP, actively promote it by the management.	1	1	1	2
Allow developers decide when to use PP, no promotion by the management.	2	3	2	1
PP should be used only very little or not at all.	4	4	4	4

Table 2 shows the developers' opinions on several claims related to pair programming as such and in comparison to solo programming. The attitudes towards pair programming before trying it varied from slightly negative to quite positive. All developers strongly agreed that pair programming increased their understanding about the developed software more than solo programming. Two of the developers considered that pair programming also helped understanding the development tools better, whereas the other two did not find a difference in this aspect. All developers agreed that there were less bugs in the code when using pair programming, which contradicts to what they said in the interview. Considering the improvement in the understandability of code, the opinions varied from no effect to strong agreement. The opinions on the effects to the discipline of following the development process varied the most from quite strong agreement to strong disagreement. All developers will promote the use of pair programming in their future projects, and all liked doing pair programming, some even more than solo programming.

Table 2. The developers' opinion on pair programming. Scale: 7=Yes, I totally agree, 4=Neutral, 1=No, just the opposite.

Claim	A	B	C	D
Before this project I had a positive attitude towards PP.	6	5	4	3
PP increased my understanding in the developed software more than SP.	7	6	7	7
PP increased my understanding of the used development tools more than SP.	6	4	7	4
PP resulted in a higher number of bugs in the code than SP.	3	2	1	2
PP resulted in more comprehensible code than SP.	5	4	7	6
PP decreased discipline in following our development process.	5	2	7	3
I will promote the use of PP in my future projects.	5	6	7	5
I liked doing SP.	6	3	7	6
I liked doing PP.	7	7	7	6

4 METRICS COLLECTED DURING THE PROJECT

The data collected by the team itself during the project is presented in Table 3. The data characterizes the size of the project and the amount of pair work during the project. Pair work was used quite a lot in all iterations. Studying the effects of using vs. not using pair programming to the productivity and quality is impossible, because there are no differences in the amount of pair programming used and many other factors certainly also affected productivity. Only when the team size shrank to two, its use was much lower, but this changed the overall situation so much that these iterations are not comparable to others.

Without data from otherwise similar projects where pair programming was not used, it is also dangerous to evaluate the effects of pair programming in general. However, the data characterizes what kind of quality and productivity was achieved in this kind of a project when pair work was used quite a lot. This data can be used as a basis for comparison, if similar projects are executed.

Table 3. Project metrics.

	Pre	I1	I2	I3	I4	I5	I6	Total
Number of developers	4	4	4	4	4	2	2	
Working days	4	9	10	10	11	10	17	71
Total effort	129h	233h	235h	264h	274h	147h	222h	1505h
Programming work								
Work hours with a pair	32h	91h	79h	109h	85h	12h	11h	419h
Pair working %	100%	91%	63%	79%	79%	67%	18%	72%
Non-programming work								
Work hours with a pair	-	135h	117h	159h	140h	42h	37h	628h
Pair working %	-	82%	58%	73%	57%	28%	16%	52%
Productivity								
LOC (increase)	517	2198	1411	1859	2290	248	700	8706
Productivity (LOC/all hours)	4.0	9.4	6.0	7.1	8.4	1.7	3.2	5.8
Productivity (LOC/programming hours)	16.2	21.5	11.2	13.4	21.3	13.8	11.6	14.9
Defects								
Post release defects		1	4	3	4	3	2	17
Post release defects/KLOC		0.5	1.1	0.6	0.5	0.4	0.2	0.2

5 SUMMARY

Developers seemed to consider pair programming a good practice for the project. Even though the use of pair programming was partially unavoidable due to the limited amount of workstations, the developers would have probably even voluntarily used it quite a lot. They all reported enjoyment of doing pair programming. The only complaint was on using it for simple tasks.

Pair programming might have had a positive effect on the code quality. The developers believed in it themselves and the defects metrics actually were very low for this type of a system. The developers also confidently and unanimously reported on positive effects to code comprehensibility and knowledge transfer.