FREE UNIVERSITY OF BOZEN BOLZANO

# Analyzing Pair Programming with respect to Tool Usage and Developer's Expertise

Bachelor Thesis

Supervisor: Professor Giancarlo Succi
Author: Jelena Vlasenko

Bolzano 2010

## Contents

Contents....2

Abstract....4

Zusammenfassung....5

Riassunto....6

Acknowledgement....7

1. Introduction....8

  1.1. Problem Statement....8

  1.2. Aims and Objectives....9

  1.3. Research questions....9

  1.4. Methodology....9

2. Background and Related Work....11

  2.1. Agile Methodology in Software Development....11

  2.2. Extreme Programming....12

  2.3. Pair Programming....13

  2.4. Research on Pair Programming....13

  2.5. Research on tool usage....15

  2.6 Contribution of this study to other research works on Pair Programming and Tool Usage....16

3. Research Methodology....18

  3.1. The Goal-Question-Metric....18

  3.2. Application of the Goal-Question-Metric to the research....18

4. Experimental Settings....21

4.1. Structure of this study ........ 21

4.2. Data ........ 21

4.2.1 The developers ........ 21

4.2.2 Data collection ........ 22

5. Results ........ 23

5.1. Time distribution and usage of tools ........ 23

5.2. Transitions and cycles in usage of tools ........ 25

6. Discussion ........ 30

7. Conclusions and future work ........ 33

Bibliography ........ 34

Appendix A ........ 37

Appendix B ........ 45

## Abstract

Pair Programming is a software development technique when two developers use one computer to work on the same task. They use one monitor and one keyboard. The developer who is typing in a program code is called the driver and the second developer, who is observing and brainstorming the work of the first developer, is called the navigator. The driver writes the code, finds fast solutions, and implements algorithms. The navigator observes the work of driver, thinks at more complex solutions, and finds logical errors. The developers should change their roles at least every 30 minutes. In this study there has been analyzed the work of an IT team of a large Italian manufacturing company, which prefers to remain anonymous. The team is composed of 19 developers – 15 already working in the company and 4 newly hired. The study uses a non-invasive measurement collection obtained from PROM. Specifically there has been investigated the usage of tools by developers when working alone and in pairs, taking also into consideration their working experience in the company. The results of this study indicate that the developers working in pairs devote significantly more time to programming activities than the developers working alone.

## Zusammenfassung

Pair Programming ist eine Technik der Software-Entwicklung, wenn zwei Entwickler einen Computer benutzen, um auf der gleichen Aufgabe zu arbeiten. Sie benutzen einen Monitor und eine Tastatur. Der erste Entwickler, der einen Programmcode eingeben wird, nennt man der Fahrer und der zweite Entwickler, der die Arbeit des erstes Entwicklers beobachtet und begeistert, nennt man der Navigator. Der Fehrer schreibt den Code, sucht nach schnelle Lösungen und implementiert Algorithmen. Der Navigator beobachtet die Arbeit des Fahrers, denkt bei komplexeren Lösungen und findet logische Fehler. Die Entwickler sollen ihre Rollen mindestens alle 30 Minuten umtauschen. In dieser Studie wurde die Arbeit eines IT-Team eines großen italienischen Herstellerunternehmen analysiert, die anonym bleiben bevorzugt. Das Team besteht aus 19 Entwicklern - 15 Entwickler, die bereits im Unternehmen arbeiten, und 4 Entwickler, die vor kurzem eingestellt den Unternehmen beigetreten haben. Die Studie benutzt eine Sammlung nicht-invasive Messungen, die aus PROM erhalten sind. Konkret gibt es seit dem Einsatz von Werkzeugen von Entwicklern bei der Arbeit allein und in Paaren untersucht, wobei auch berücksichtigt ihre Berufserfahrung in die Firma. Die Ergebnisse dieser Studie zeigen, dass die Entwickler, die in Paaren arbeiten, widmen mehr Zeit der Programmierungtätigkeiten, als Entwickler, die allein arbeiten.

## Riassunto

Pair Programming è una tecnica di sviluppo software quando due sviluppatori utilizzano solo un computer per lavorare sullo stesso compito. Loro usano un monitor e una tastiera. Lo sviluppatore che digita il codice di programma si chiama l'autista e il secondo sviluppatore, che sta valutando e ispirando il lavoro del primo sviluppatore, si chiama il navigatore. L'autista scrive il codice, trova i soluzioni veloci, e implementa gli algoritmi. Il navigatore osserva il lavoro di autista, pensa a soluzioni più complesse, e trova gli errori di logica. Gli sviluppatori dovrebbero cambiare i loro ruoli, almeno ogni 30 minuti. In questo studio è stato analizzato il lavoro della squadra IT di una grande azienda italiana, che preferisce rimanere anonima. Il team è stato composto da 19 sviluppatori - 15 di loro che già lavorano in azienda da tanto tempo e 4 neo-assunti. Lo studio utilizza una colezzione di misura non-ivasiva che si ottiene da PROM. In particolare è stato studiato l'utilizzo dei strumenti da parte degli sviluppatori quando si lavora da solo e in coppia, prendendo in considerazione anche la loro esperienza di lavoro in azienda. I risultati di questo studio indicano che gli sviluppatori che lavorano in coppia dedicano molto più tempo alla programmazione delle attività di sviluppatori che lavorano da soli.

## Acknowledgement

I am sincerely thankful to my supervisor, Professor Giancarlo Succi, who supported me throughout this work with his patience and knowledge. This thesis would have not been possible without his guidance.

I am extremely grateful to Ilenia Fronza for her assistance and precious expertise. She was always ready to answer the innumerable questions I had in the beginning of this study and helped me to develop an understanding of the subject.

Finally, I cannot forget the moral support of my parents. I am grateful to my friends Artem Konev and Viktor Skoks for their patience and readiness to help during the whole period of my studies at Free University of Bozen Bolzano.

# 1. Introduction

The software development process is a set of phases needed to produce a working software product that satisfies the requirements of a customer.

Since the early 60s there have been several proposals of development processes. These different processes differentiated one another for the different phases, and for the time distribution, and the mutual dependencies among these phases. Typical phases that have been proposed include: planning, implementation, testing, documentation, deployment, and maintenance.

Several software development models have been developed in the last 40 years to organize the development process. Most known of them are: waterfall model, spiral mode, iterative and incremental development, and agile development [Begel and Nagappan, 2007]. Each of them has advantages and disadvantages. A good recommendation is to select for each project the most suitable model or even combinations of models and adopt them for the current project.

Agile methods suggest a disciplined project management process. Software product is deployed with iterative improvements and additions of new customer requirements. During the planning phase it will be enough to gather only the most important and fundamental requirements. Further the missing ones will be gathered and implemented during the next iterations. Though, developers should be able to present a correctly working software product at the end of each iteration. Agile development is very flexible to customer wishes regarding the project and changing requirements. Moreover, it is possible to have the working software product with minimal functionality at the end of each iteration that can be just about 2 weeks long. That is why nowadays agile methods attract a lot of attention of project managers.

One of the techniques of agile methods that has gained popularity both in academic and industrial environment is Pair Programming [Begel and Nagappan, 2008]. Pair Programming is one of the practices of Software Development where 2 programmers work on the same task at one computer using 1 monitor and 1 keyboard. Many advantages of Pair Programming have been identified: [Cockburn and Williams, 2001, Succi et al., 2002, Heiberg et al., 2003, Hulkko and Abrahamson, 2005, Lui and Chan, 2006, Braught at al., 2008, Vanhanen and Korpi, 2007] when working in pairs defects are detected when they are typed in, code becomes shorter, pairs solve problems and complete tasks faster than individuals, developers constantly exchange their knowledge and developers are more satisfied with their work.

## 1.1. Problem Statement

Many research works have been conducted on Pair Programming. Several ones were carried out in an educational environment. In these cases participants were computer science students. Only few of the experiments were carried out in an industrial environment. The period of time during which the experiments were held, was no longer than few days. In some researches there has been analyzed performance of students during a whole studying semester. The results obtained from the experiments held in the educational environment may not be useful for industry. In addition, results obtained from different empirical studies contain contradictions. In this study there has been analyzed work of software developers of a large Italian manufacturing company during a period of 10 months. There has been investigated how Pair Programming affects the patterns of usage of tools. This work could be useful for further analysis of Pair Programming in industrial software development teams.

## 1.2. Aims and Objectives

The aim of this study is to analyze the effects of Pair Programming on usage of tools during developers' daily work.

To achieve the aim there has been evaluated how the developers use tools to perform their daily tasks when they work alone and when in pairs, taking into consideration their working experience in the company.

## 1.3. Research questions

This study tries to find answers to the following questions:

- Is there a difference in tool usage when programmers work alone and when in pairs?
- Is there a difference in tool usage when the developers have different working experience?

## 1.4. Methodology

This work is an observation study in which there has been investigated an impact on tool usage when developers work alone and when in pairs. The data for this study has been collected from an industrial team of software developers. The team uses spontaneous Pair Programming (i. e., when the developers find it appropriate) during the observation time space. It gives a sufficient evidence to answer the research questions.

The research work is divided in 5 phases:

- Familiarization,
- Planning,
- Data collection,
- Data processing,
- Data analysis.

In the first phase there has been provided familiarization to existing researches on Pair Programming and tool usage, techniques of data collection and data storage environment. In the second phase the detailed structure of the research had been planned, defined the research questions and measures to answer them. In the third phase the data needed for the research has been extracted. In the fourth phase the data has been structured and processed for the further analysis. In the final fifth phase data analysis has been performed and the obtained results have been evaluated.

The importance of this research is as follows:

- In majority of studies, participants usually are either students or professional software developers working on the assignments prepared specially for an experiment. In this study the data obtained from experienced software developers working in their environment has been analyzed,
- Developers decide themselves when it is more effective to use Pair Programming than solo programming,
- The time space during which the data was collected is quite large – 10 months, so that it possible to obtain a large data sample,
- There have been used plots, graphs and tables for data visualization.

# 2. Background and Related Work

## 2.1. Agile Methodology in Software Development

Nowadays project managers express a growing interest in the agile methods. The agile methods are helpful for the teams of developers who apply practically iterative software development approach in their daily work. In this approach software development process is divided into short release phases known as sprints.

The main idea of the agile development methodology is that software developers should not fear or avoid changes in the project during its development cycle. Oppositely, they should accept new requirements and be able to change a development plan according to the new requirements with minimal losses to the project. This can be achieved applying regular modulations of work, known iterations or sprints. Teams of developers should be able to present at the end of each iteration a shippable software product with new functionalities added during the sprint. Thus, the developers focus on short work cycles and on the functionalities of the product they have to deliver. If there is applied the waterfall software development methodology the developers will have only one chance to understand each aspect of the project correctly. In the end this can lead to delivering the project that is not corresponding to customers' wishes. According to the agile methodology, every aspect of development should be revisited constantly throughout the project's lifecycle. A team of developers has to stop and re-evaluates the direction of a project in the end of each iteration to be sure that the project they develop is still the desired one. Thus, there is always a possibility go back with minimal losses to the project if the team has been moving in a wrong direction. Figure 1 represents the idea of the agile approach.

The agile approach significantly reduces both development costs and time to deliver the project. Since the work cycles should be limited to 2 weeks it gives the customers the insurance that the project under the development is really the product they want. Moreover, after each iteration the developers should deliver a correctly working piece of software. Thus, the customers can start using and test the product before the whole development cycle is finished.

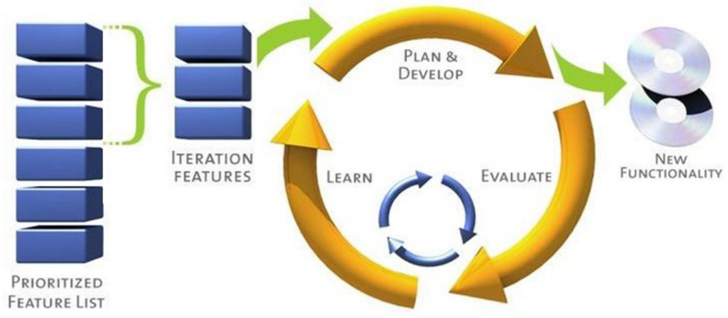


Figure 1: "Agile approach to Software Development." Source: http://leadinganswers.typepad.com

The agile methods are aimed to meet customers' visions about a project. When developing the project more value should be given to individuals and communication between them than to processes and tools they use. A working software product should have higher priority than extensive documentation. Having a customer on site should be more advantageous for the project than having periodical negotiations.

When applying the agile methods to the software development, customers' satisfaction can be achieved by having frequent product releases. Working software product is a way to evaluate progress of the project. To achieve better results the developers should constantly communicate with the customers to have a better understanding about the project and to be sure that they move in a right direction. Moreover, it is important to provide the developers with good working environment. Teams of developers should be located in such way that constant face-to-face communication could be possible when it is needed.

There have been introduced many agile software development methods. Project managers should choose the most suitable one or combination of different ones for every project. Some of these methods are: Agile Modelling, Agile Unified Process, Essential Unified Process, Feature Driven Development, Extreme Programming (XP), and Scrum. The last two methods raise an increasing interest from software developers and project managers.

There are many agile software development practices which are applied by different methods. Some of them are: Test Driven Development, Behaviour Driven Development, Code Refactoring, Continuous Integration, and Pair Programming. The last practice is widely applied by Extreme Programming.

## 2.2. Extreme Programming

In traditional system development methods, like the waterfall model, software development process consists of several phases. Each phase should be finished before the next one can be started. Thus requirements for the system are fixed at the beginning of the project. Extreme Programming (XP) is one of agile software development methods which is focused on software quality improvement. As one of the methods of the agile software development, it also applies frequent product releases in short development cycles. It is important for productivity improvement and for adopting new customer requirements throughout development cycle.

Some of main principles of extreme programming are the following: Pair Programming, constant code reviews, unit tests, Test Driven Development, an organized project management structure, code's simplicity, ability to react quickly to changes in the customer's requirements, having representatives of the customer on site and good team work.

## 2.3. Pair Programming

Pair Programming is one of the agile practices when two developers work at one computer on the same task. The first developers types in program code and the second developer observes the work of the first one. The first person is called the driver and the second one is called the navigator. It is advised that the driver and the navigator exchange their roles every 30 minutes or even less.

The role of the navigator is more strategic than the role of the driver. He should be more concentrated on solving complex problems, identifying logical errors and constantly reviewing the program code. Thus, the driver can be more focused on the "tactical" aspects of completing the current task, relying on the navigator as on his guide.

It is claimed that when working in pairs the developers produce shorter programs with better designs and fewer bugs than when working alone. Two developers working as a pair usually complete work faster than one developer when they are assigned to the same task. The developers working in pairs solve at first sight impossible problems quickly and in an elegant way. Regarding the total time the developers spend to complete a task it is has always been questionable why there is a need to assign two developers to it if this task can be completed by only one. A project manager should take into consideration that Pair Programming can reduce the time needed for bug fixing but it can increase the cost of coding. Thus, it is really important to prioritize tasks where Pair Programming will be beneficial. The relative weight of these factors can vary from a project to a project and from a task to a task. When working on the tasks that are complex or not yet understood or known, Pair Programming could be a good solution. On the other hand, working in pairs on simple tasks will lead to excessive and needless costs.

Pair Programming could be also useful for knowledge transfers since both programmers exchange constantly their experience when they work as a pair. They share knowledge of the specifics of the system they are developing and they learn programming techniques from each other. It can be really useful when introducing newly hired developers to the team – they learn quickly the specifics of the project they should work on. When programmers exchange their partners randomly they reduce possible risks if one of the developers leaves the team.

Another advantage of Pair Programming is improved time management. Developers are better disciplined when they work in pairs than when they work alone. The developers working in pairs are less likely skip writing unit tests and are more careful about the code quality since there is always the observer who will notice it. Also it is true to say that when working in pairs the developers spend less time on private browsing and writing e-mail than when working alone. The developers working in pairs are more confident about the quality of their code since they perform constant code reviews.

## 2.4. Research on Pair Programming

In the last years there has been conducted a large number of researches on Pair Programming. Most of them were focused on costs and benefits [Cockburn and Williams, 2001] of this technique. Table

1 summarizes some of these researches identifying subjects, goals, and obtained results. In the Appendix A there can be found a more extensive overview on research works conducted on this topic.

Some of the studies were aimed to determine if Pair Programming had a positive effect on developer's working skills [Vanhanen and Korpi, 2007, Braught et al., 2008] and on code's quality [Williams et al., 2000, Heiberg et al., 2003, Vanhanen and Korpi, 2007, Begel and Nagappan, 2008]. Most experiments were held with students. Validity of these experiments is questionable and it cannot be generalized to teams of industrial developers. Only few ones were held with professional developers [Lui and Chan, 2003, Hulkko and Abrahamsson, 2005, Canfora et al., 2006, Vanhanen and Korpi, 2007, Chong and Hurlbutt, 2007, Arisholm and Sjoberg, 2007]. It has been identified in [Lui and Chan, 2003] that pairs outperform individuals only when tasks are challenging and new to them. In [Hulkko and Abrahamsson, 2005] there have been analyzed four software development projects. It has been claimed that Pair Programming does not provide an extensive quality benefits and does not result in consistently superior productivity when comparing to solo programming. In [Canfora et al., 2006] it has been claimed that Pair Programming decreases productivity but on the other hand increases code quality. In [Arisholm and Sjoberg, 2007] authors conducted an experiment with the largest number of professional developers than in other experiments on Pair Programming. It appears that junior pair programmers have achieved a significant increase in correctness comparing to the individuals and have achieved approximately the same degree of correctness as senior programmers working alone.

| **Authors** | **Subjects** | **Goal of the experiment** | **Results** |
|---|---|---|---|
| "Pair Programming: What's in it for Me? ," Andrew Begel, Nachiap-pan Nagappan (2008) | 487 surveys | To continue the preced-ent study (Begel et al., 2007). | PP allows the introduction of fewer bugs, spreading code understanding and over-all higher quality of the produced code. Disad-vantages of PP are cost-ef-ficiency , work time scheduling difficulties and personality conflicts. |
| "Evaluating Pair Pro-gramming with Respect to System Complexity and Programmer Ex-pertise," Erik Arisholm, Hans Gallis, Tore Dyba, and Dag I. K. Sjoberg (2007) | 295 junior, intermediate and senior professional Java consultants. | T o detect if PP reduces the time required to solve tasks correctly or increases the propor-tion of correct solu-tions. | Junior pair programmers achieved a significant in-crease in correctness compared with the indi-viduals and achieved ap-proximately the same de-gree of correctness as senior individuals. |

| Authors | Subjects | Goal of the experiment | Results |
|---|---|---|---|
| “Usage and Perception of Agile Software Development in an Industrial Context: An Exploratory Study,” Andrew Begel, Nachiappan Nagappan (2007) | 491 professionals (Microsoft developers, testes and managers who directly involved in the development of software). | To evaluate communication between team members, speed of releases and flexibility. | Advantages of Agile Software development: Improved communication between team members, quick releases and increased flexibility of Agile designs. |
| “The Social Dynamics of Pair Programming,” Jan Chong, Tom Hurlbutt (2007) | 10 professional programmers. | To investigate how professionals perform when working in pairs. | Pairs appeared to be more efficient when both programmers took on driver and navigator responsibilities. Equipping pair programmers with dual keyboards facilitates the rapid switching of keyboard control. |
| “Evaluating Performances of Pair Designing in Industry,” Gerardo Canfora, Aniello Cimitile,Felix Garcia, Mario Piattini, Corrado Aaron Visaggio (2006) | 18 professional programmers (5 pairs and 8 individual programmers). | To investigate how PP affects system design. The quality of the result is evaluated by 2 independent evaluators. | Designing in pairs decreases productivity but increases the quality of the product. |

Table 1: “Overview of existing studies on Pair Programming (PP)”

## 2.5. Research on tool usage

The first principle of agile methods says that if you want to succeed in software development you should give more value to people than to tools and processes. Still to evaluate and to improve performance of developers it is essential to understand how they work, which tools they use, and for which purposes.

Until now there have been conducted only few researches on tool usage. Table 2 gives an overview of some them. In some studies there has been investigated one specific tool and how it has been used. The goal of these studies mainly is to propose a better tool to perform the same tasks with better results. Other research works have been focused on which tools developers use during their daily work aiming to find a set of tools which they regularly use. The results of these studies show that the developers tend use regularly rather a small set of tools. On the other hand, they constantly test a large number of new tools for their daily work. Though, these studies were not investigating if there are dependencies among the tools the developers use and if some of these tools are typically used together.

| Authors | Subjects | Goal of the experiment | Results |
|---|---|---|---|
| "An Exploratory Study of Developers' Toolbox in an Agile Team," I. D. Coman and Giancarlo Succi, 2009 | 3 developers of a small Austrian company | To detect how many tools use developers, which tools are used frequently and for which purpose they serve. | Developers use 41 distinct tools.<br>Developers use 12, 11 and 13 tools respectively.<br>Five main activities: Documents, Navigating, Communication, Internet and Coding. |
| "Maintaining Mental Models: A Study of Developer Work Habits," T. D. LaToza et al., 2006 | 344 survey responses by Microsoft's software design engineers;<br>Interviews with 11 Microsoft's software design engineers | To detect which tools developers use, what their activities and practices are. | Developers use a variety of tools and search for better solutions.<br>Developers switch frequently between tools.<br>Developers prefer face-to-face communication than electronic one. |
| "An Examination of Software Engineering Work Practices," J. Singer et al., 1997 | Group that maintains a large telecommunication system: 6 survey responses, exploring work of 10 developers, the company's tool usage statistics. | To provide software engineers with a toolset to improve their daily work activities. | Developers most of their time use compilers and search tools. |
| "Understanding Software Maintenance Tools: Some Empirical Research," T. C. Lethbridge and J. Singer, 2007 | Team of software engineers. | To detect which tools developers and what should be improved to make them more productive. | Developers use more often editors and search tools than other tools |

Table 2: "Overview of existing studies on Tool Usage"

## 2.6 Contribution of this study to other research works on Pair Programming and Tool Usage

There has been reviewed a number of existing studies on Pair Programming (Appendix A). It has been found that most studies have been conducted with students and only few ones with professional software developers. Moreover, the time space during which the studies were held was no longer than few days. It has been noticed that there the researches do not investigate effects of Pair Programming on tool usage. In this study there has been investigated how developers use different tools when they work alone and when they work in pairs. Data for this study comes from a team of professional software developers from a large Italian manufacturing company. The developers use Pair Programming during their daily work. The data has been collected during a time space of 10 months from October 2007 to July 2008 non-invasively by means of PROM (PRO Metrics) [Sillitti et al., 2003]. PROM automatically collects data from the tools the developers use. Moreover, the developers were asked to constantly check its correctness. Thus, the data can be considered very reliable.

# 3. Research Methodology

## 3.1. The Goal-Question-Metric

Goal-Question-Metric (GQM) paradigm has been introduced by Victor Basili [Basili et al., 1994]. GQM is a top-down approach to Software Metrics to introduce a goal-driven measurement system for software development. When applying GQM, firstly goals are defined, then the questions that address these goals are stated, and then metrics that provide answers to the questions are identified.

GQM is a measurement model that consists of three levels (Figure 2): computational, operational, and quantitative levels. On the conceptual level the goals for the object that should be measured are defined. The object can be a product, a process or a resource. The product involves everything what can be produced during the system life cycle. On the operational level the questions that are aimed to characterize the defined goals are stated. On the quantitative level corresponding data for every question is identified. The data can be either objective when it depends only on the objet that should be measured or subjective when it depends not only on the object but also on a viewpoint from which the data has been taken.

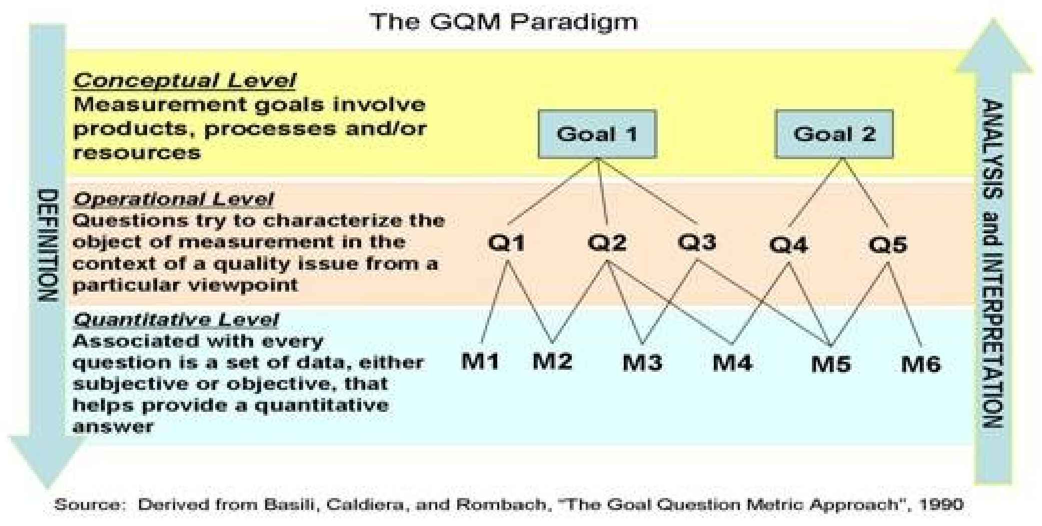

The GQM Paradigm
Conceptual Level
Measurement goals involve products, processes and/or resources
Operational Level
Questions try to characterize the object of measurement in the context of a quality issue from a particular viewpoint
Quantitative Level
Associated with every question is a set of data, either subjective or objective, that helps provide a quantitative answer
Goal 1
Goal 2
Q1
Q2
Q3
Q4
Q5
M1
M2
M3
M4
M5
M6
DEFINITION
ANALYSIS and INTERPRETATION
Source: Derived from Basili, Caldiera, and Rombach, "The Goal Question Metric Approach", 1990

Figure 2: "The GQM Paradigm"

## 3.2. Application of the Goal-Question-Metric to the research

There has been analyzed the performance of the team with respect to the following two factors:

- When the developers work in pairs and when they work alone

- Are the developers new to the team or they are already experts

With respect to the factors stated above there has been an interest in analyzing work of the following patterns of work:

- Experts solos (experts working alone)
- Experts pairs (pairs of 2 experts)
- Novices solos (novices working alone)
- Novices pairs (pairs of 2 novices)
- Mixed pairs (pairs of 1 novice and 1 expert)
- Pairs (experts pairs and novices pairs)
- Solos (experts solos and novices solos)

With respect to the patterns of work that have been identified above the following **goal** can be stated:

**Goal:** Analyze work of the team of the developers and to evaluate it taking into consideration if they are new to the team or not and if they work in pairs or alone.

To analyze the work of the team there have been selected nine core applications that are common for all patterns of work of developers and are used regularly during the whole period of this study. These applications cover 85% of total time spent by the developers on working activities. Applications that had been used only occasionally were excluded. These applications are: Browser, Outlook, Microsoft Office Excel, Microsoft Office Word, Microsoft Messenger, Microsoft Windows Explorer, Microsoft Management Console, Remote Desktop and Visual Studio.

As the result of this study the following **questions** should be answered:

- **Q1:** Do the developers when they work in pairs spend more time on programming activities than when they work alone?
- **Q2:** Do the developers when they work in pairs spend less time on browsing and writing e-mails than when they work alone?
- **Q3:** Do the developers when they work in pairs browse more for business purposes than for private ones than when they work alone?
- **Q4:** Is there a difference how the developers cycle from Visual Studio to one of the applications from the application set and back to Visual Studio (Visual Studio->Application1->Visual Studio)?

- **Q5:** Is there a difference how the developers cycle from Visual Studio to one of the applications from the application set, then to another application from the application set and then back to Visual Studio (Visual Studio->Application1->Application2->Visual Studio)?

After the goal of the research has been identified and the research questions have been raised the **metrics** to answer the questions can be identified:

- **M1:** Percentage of time spent in each application with respect to total time spent working in the applications
- **M2:** Average time the developers spent in each application
- **M3:** Probability to switch from one application to another
- **M4:** Percentage of time when developers browse for business purposes and when for private ones
- **M5:** Percentage of cycles of type *Visual Studio->Application1->Visual Studio*
- **M6:** Percentage of cycles of type Visual *Studio->Application1->Application2->Visual Studio*

# 4. Experimental Settings

## 4.1. Structure of this study

This study has been conducted in the following way:

1. Seven patterns of work of developers have been identified, taking into consideration their work experience in the team. These patterns are experts solos, experts pairs, novices solos, novices pairs, mixed pairs, solos, and pairs.
2. Applications that are common for all patterns of work and are used regularly during a time space of 10 months have been identified. These applications are Browser, Outlook, Microsoft Messenger, Microsoft Office Word, Microsoft Office Excel, Microsoft Windows Explorer, Microsoft Management Console, Remote Desktop, and Visual Studio.
3. The data to compute total time spent in each application, average time to stay in each application, and probability to switch among the applications have been extracted.
4. Graphs that visualize data obtained in step 3 have been built.
5. Cycles of type *Visual Studio->Application->Visual Studio* and of type *Visual Studio->Application1->Application2->Visual Studio* have been identified and computed. Total number of cycles of each type for all patterns of work of the developers has been computed. Total and average time spent in the cycles has been computed.

## 4.2. Data

### 4.2.1 The developers

The data has been collected from the team of developers of IT department of a large Italian manufacturing company that prefers to remain anonymous and it is the same as it has been detailed in Section 4.1.1 of [Phaphoom, 2010]. The study has covered a time space of 10 months from October 2007 to July 2008. The developers are Italians. All of them have university degrees in computer-related areas. The team is composed of professional developers who have programming experience from 10 to 15 years. During this time space the developers performed maintenance and improvement of the existing software. The percentage of Pair Programming was identified when the developers were working on different methods in classes. Table 3 represents characteristics of the collected data, showing the total number of methods accessed during the time space and the descriptive statistic of pair programming used. During the period of 10 months the team of developers interacted with 24765 methods and the mean of Pair Programming applied to each method is 5.95%.

The programming language that has been used is mainly C#. The developers use some of the Extreme Programming practices in the development process during their daily work. In particular, they use weekly iterations, Pair Programming, user stories, collective code ownership, coding standards, and test driven development. The team members use spontaneous Pair Programming, i.e. when they find it useful and appropriate. Each developer has his own workplace and workstation. The team is located in an open space what favours communication and knowledge transfers among them.

| | |
|---|---|
| Time space | October 2007 – July 2008 |
| Number of accessed methods | 24765 |
| Number of accessed classes | 3238 |
| AVG percentage of Pair Programming | 5.95 |
| Standard deviation | 21.32 |

Table 3: "Characteristics of the collected data"

### 4.2.2 Data collection

The data in this study represents all the activities of the developers at their computer. It has been collected non-invasively by means of PROM (PRO Metrics) [Sillitti et al., 2003]. PROM is a tool for automated data collection and analysis. It collects both code and process measures. PROM's architecture is based on plug-ins that collects data from the tools the developers use. Thus, PROM has information about all software application used by the developers, the time they spend in them, and the identifiers of the developers. If the developers do Pair Programming PROM will store information also about composition of pairs.

Before the beginning of this study the developers had an experience of working with the PROM. They also got comprehensive information about the tool and what kind of data it collected. Each team member could access to his own data and also to summary data of other team members. Moreover, the developers had rights to look at the data stored at their own machine and to decide if they want to send it to the central database or to delete it. The developers we asked to check the reliability of the data that had been collected. The developers collected the data throughout the study. Moreover, the participation in this study from the part of the developers was on a voluntary basis.

# 5. Results

## 5.1. Time distribution and usage of tools

In the beginning of this study there has been computed the amount of time the developers spend in different applications during the time space of ten months. For the analysis there have been selected nine core applications that are common for all seven patterns of work of the developers (experts solos, experts pairs, novices solos, novices pairs, mixed pairs, solos, and pairs) and are used regularly during this space of time. These applications are Visual Studio, Browser, Outlook, Microsoft Office Word, Microsoft Office Excel, Microsoft Messenger, Microsoft Windows Explorer, Microsoft Management Console and Remote Desktop.

Box plots were used to visualize the data. The box plots were created in R – an environment for statistical computing and graphics (www.r-project.org). It has been noticed that most time the developers spend in the following applications - Visual Studio, Browser, and Outlook.

Figure 3 represents how the developers applying different patterns of work use Visual Studio. It can be seen that the novices and the experts when they work alone behave similar and devote to programming activities about 33% of their time. The developers working in pairs spend significantly more time on programming activities than the developers working alone. The experts working in pairs spent almost twice more time in Visual Studio than experts working alone. The novices working in pairs also spend more time on programming activities than the novices working alone but the difference is less significant than between the experts working alone and the experts working in pairs. The mixed pairs spend 75% of their time on programming activities what is more than the developers working in the other patterns of work.

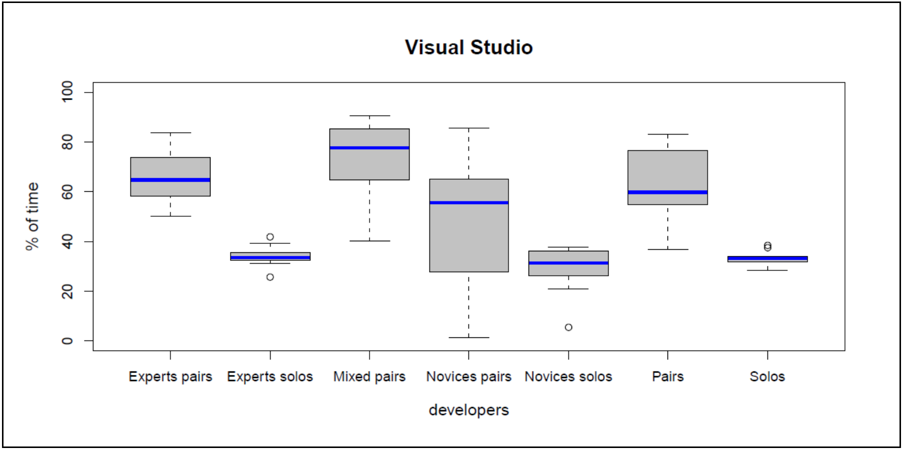


Figure 3: "Percentage of time spent in Visual Studio"

Figure 4 represents the amount of time the developers devote to writing and reading e-mails during their working time. It is noticeable that the developers when they work as mixed pairs spend less time on e-mail than when they apply any other pattern of work. The pairs spend significantly less time in Outlook than solos. The pairs spend only 5% of their time and the solos spend 21% of their time on writing and reading e-mails. The experts and the novices behave similar when they work alone. The experts working alone spend 21% of their time and the novices working alone 23%. The experts working in pairs spend a bit less time on e-mails than the novices working in pairs. The experts spend 13% of their time and the novices 15%. It is important to mention that the developers receive all the requirements for the projects they work on via e-mail. It could be possible that when the developers work alone they tend to check more often the requirements than when they work in pairs.

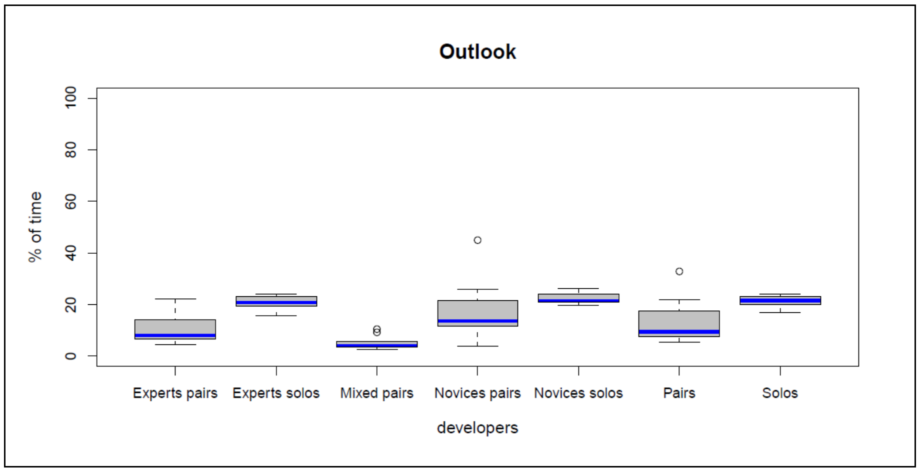


Figure 4: “Percentage of time spent in Outlook”

Figure 5 represents how much time the developers applying the seven patterns of work spend in Browser. In general, the developers working alone spend more time on browsing than the developers working in pairs. When they work alone they spend 10% of their time on browsing and when they work in pairs they spend 5% on the same activity. The novices working alone spend more time on browsing than the experts working alone. The novices spend 13% and the experts spend 9% of their time on browsing the Internet. Moreover, the novices working in pairs spend more time on browsing than the experts working in pairs. The novices spend 10% of their time and the experts only 3%. The developers working as mixed pairs spend 5% of their time on this activity.

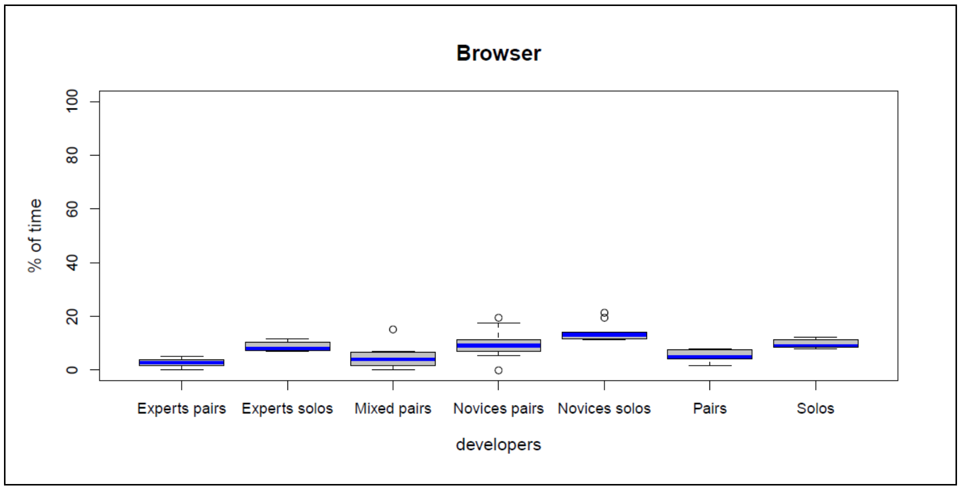


Figure 5: “Percentage of time spent in Browser”

It has been identified that the developers spend a noticeable part of their time on browsing the Internet. Thus, it has been interesting to see what exactly they do when they browse web pages. On the one hand, browsing could be interpreted as an activity that distracts the developers from their work. On the other hand, the developers when writing code might need to search for additional information or code examples for their work in the Internet. For the further analysis it has been decided to divide browsing into the two following categories: Private Browsing and Business Browsing. In the category Private Browsing there has been collected all the developer’s activities in the Internet that do not relate to their work. In the category Business Browsing there has been collected all the developers’ activities that are directly related to their work – searching for code examples, reading about technologies they use in their work and filling in time sheets. It has been identified that the developers working in different patterns of work spend from 71% to 84% of their Browsing time for business purposes. According to the results the expert developers working in pairs devote more time for business browsing than others. Table 4 summarizes the obtained results.

| | **Experts Solos** | **Experts Pairs** | **Novices Solos** | **Novices Pairs** | **Mixed Pairs** | **Solos** | **Pairs** |
|---|---|---|---|---|---|---|---|
| ***Private Browsing*** | 28% | 16% | 32% | 34% | 22% | 29% | 24% |
| ***Business Browsing*** | 72% | 84% | 68% | 66% | 78% | 71% | 76% |
| ***Total*** | 100% | 100% | 100% | 100% | 100% | 100% | 100% |

Table 4: “What the developers do when they browse”

## 5.2. Transitions and cycles in usage of tools

Seven graphs can be found in the Appendix B. The graphs visualize the data about the total and the average time spent in each application and the probabilities to switch among the applications for the seven patterns of work of the developers. A vertex represents an application. A size of the vertex depends on the total time spent in the application to which it corresponds. An arc represents a link between two applications if there is a switching between them. The arc can be one-directional, if there is a one-way switching, and bi-directional, if there is a two-way switching. In addition to the findings from the box plots above, the developers working in pairs stay in all applications longer than the developers working alone. Developers working in pairs in general stay significantly longer in Visual Studio than the developers working alone. It has been noticed that the developers in general tend to switch very often between Visual Studio and Browser, and between Visual Studio and Outlook.

Figure 6 represents how solos and pairs switch among Visual Studio, Browser and Outlook. It can be seen that the solos spend 30% of their time in Visual Studio and the pairs 60%. The solos spend 10% of their time in Browser and the pairs 10%. The solos spend 21% of their time on Outlook and the pairs only 5%. The solos stay on average in Visual Studio, Browser, and Outlook is 30 seconds, 25 seconds, and 31 seconds respectively. The pairs stay in these applications 136 seconds, 76 seconds, and 80 seconds respectively. The solos switch from Visual Studio to Browser with the probability 0.15 and from Browser to Visual Studio with the probability 0.34. The probability to switch from Browser to Outlook is 0.23 and the probability to switch from Outlook to Browser is 0.12. The developers working alone switch from Outlook to Visual Studio with the probability 0.37 and from Visual Studio to Outlook with the probability 0.26. The pairs switch from Visual Studio to Browser with the probability 0.11 and from Browser to Visual Studio with the probability 0.48. The probability to switch from Browser to Outlook is 0.17 and the probability to switch from Outlook to Browser is 0.07. The developers working in pairs switch from Outlook to Visual Studio with the probability 0.4 and from Visual Studio to Outlook with the probability 0.22.

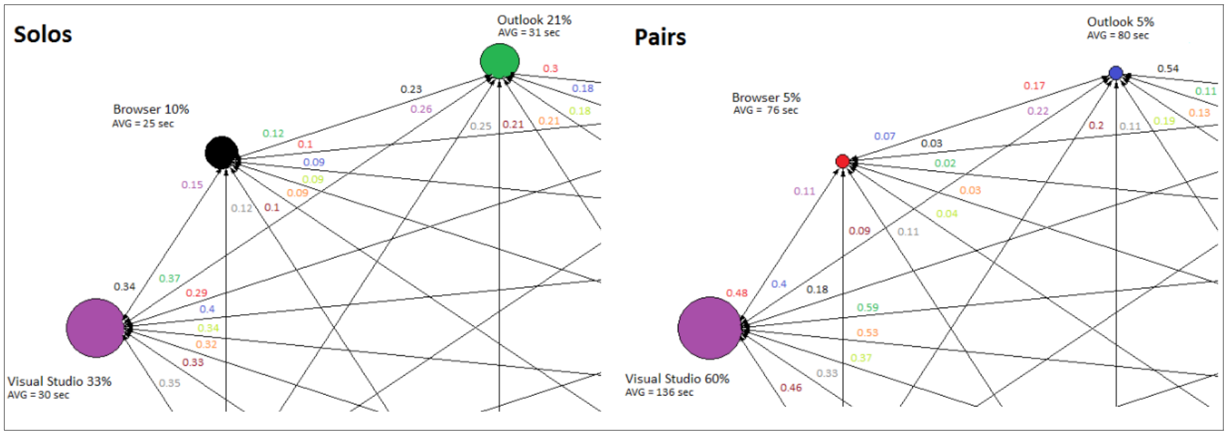


Figure 6: “Usage of tools by experts working in pairs”

Then there has been investigated how the developers cycle from one application to another. Cycles of the two following types have been investigated:

- Paths of size of 2 applications: when the developers switch from Visual Studio to one of the applications and then return to Visual Studio,
- Paths of size of 3 applications: when the developers switch from Visual Studio to one of the applications, then again to one of the applications, and then they return to Visual Studio.

It has been found that out of all the cycles the developers when they apply different patterns of work tend to spend most time in the following ones:

- Visual Studio->Browser->Visual Studio,
- Visual Studio->Outlook->Visual Studio,
- Visual Studio->Microsoft Messenger->Visual Studio.

The most time consuming cycle is when the developers switch from Visual Studio to Outlook and then return to Visual Studio. The developers applying different patterns of work spend in this cycle from 25% to 41% of time out of all the cycles of path of 2 applications. The total number of this cycle out of all cycles varies between 17% and 35%. On average the developers spend in this cycle between 75 and 611 seconds. Second time consuming cycle is when the developers switch from Visual Studio to Browser and then return to Visual Studio. The developers applying different patterns of work spend from 8% to 40% of time out of all cycles of path of 2 applications. The total number of this cycle out of all cycles varies between 9% and 43%. On average developers spend in this cycle between 62 and 350 seconds. Third time consuming cycle is when the developers switch from Visual Studio to Microsoft Messenger and then return to Visual Studio. The developers applying different patterns of work spend from 6% to 15% of time out of all cycles of paths of 2 applications. The total number of this cycle out of all cycles varies between 7% and 18%. On average developers spend in this cycle between 68 and 328 seconds. Table 4 represents the total number of the cycles, total time spent in the cycles and average time spent in the cycles mentioned above. The rest time is distributed among the remained applications. Since the developers have spent very little time in these cycles they haven't been represented in the Table 5.

| | ***Visual Studio -> Browser->Visual Studio*** | | | ***Visual Studio-> Outlook->Visual Studio*** | | | ***Visual Studio->Microsoft Messenger->Visual Studio*** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Total number of cycles %** | **Total time spent in cycles %** | **AVG time spent in a cycle** | **Total number of cycles %** | **Total time spent in cycles %** | **AVG time spent in a cycle** | **Total number of cycles %** | **Total time spent in cycles %** | **AVG time spent in a cycle** |
| **Experts solos** | 16% | 14% | 62 sec | 31% | 33% | 75 sec | 9% | 9% | 68 sec |
| **Experts pairs** | 9% | 8% | 221 sec | 24% | 37% | 391 sec | 14% | 12% | 230 sec |
| **Novices solos** | 24% | 22% | 74 sec | 35% | 41% | 94 sec | 8% | 9% | 84 sec |
| **Novices pairs** | 43% | 40% | 350 sec | 17% | 28% | 611 sec | 7% | 6% | 328 sec |
| **Mixed pairs** | 18% | 21% | 268 sec | 21% | 25% | 276 sec | 18% | 15% | 199 sec |
| **Solos** | 18% | 16% | 65 sec | 23% | 34% | 79 sec | 9% | 9% | 70 sec |
| **Pairs** | 15% | 15% | 284 sec | 23% | 35% | 423 sec | 13% | 9% | 84 sec |

Table 5: "Cycles of paths of 2 applications"

For the cycles of path of 3 applications there has been computed the same data as for the cycles of paths of 2 applications. Moreover, it has been noticed that these cycles are scarce in the daily work of the developers. Though, it has been found that among these cycles the most time consuming one is when the developers switch from Visual Studio to Browser then they switch to Outlook, and then return to Visual Studio. Table 6 represents how the developers applying different patterns of work devote time to this cycle. The percentage is computed out of all cycles when the developers switch from Visual Studio to Browser and from Browser to the rest 8 applications. It can be seen that the total number of these cycles varies from 20% to 83% out of all cycles, total time spent varies from 10% to 99.6%, and the average time spent in this cycle is between 67 and 741 seconds.

|  | ***Visual Studio ->Browser-> Outlook->Visual Studio*** |  |  |
|---|---|---|---|
|  | **Total number of cycles %** | **Total time spent in cycles %** | **AVG time spent in a cycle** |
| **Experts solos** | 39% | 40% | 100 sec |
| **Experts pairs** | 47% | 30% | 67 sec |
| **Novices solos** | 56% | 56% | 128 sec |
| **Novices pairs** | 83% | 99.6% | 741 sec |
| **Mixed pairs** | 20% | 10% | 412 sec |
| **Solos** | 42% | 44% | 107 sec |
| **Pairs** | 57% | 61% | 388 sec |

Table 6: “Cycles of paths of 3 applications”

# 6. Discussion

The goal of this study is to analyze and evaluate work of an industrial team of software developers working in a large Italian company. Since there is a growing interest to Pair Programming it has been taken into consideration if the developers when performing their daily tasks have been working alone or in pairs. It has been found essential to distinguish the developers according to their work experience in the company. The developers who work for the company more than 5 years are called the experts and the one who have recently joined the company are called the novices. There has been investigated how they use different applications during their daily work. Five research questions were stated to understand the behaviour of the developers. Six metrics were identified to answer these questions. Table 7 summarizes the answers to these questions.

It has been found that the expert developers when they work in pairs spend 64% of their time on programming activities. Moreover, when the experts work alone they spend only 34% of their time on programming activities. The novices when they work alone behave very similar to the experts and spend 32% of their time on programming activities. The novices working in pairs spend only 49% of their time on programming activities what is less that spend the experts working in pairs. In general, it can be concluded that the developers working in pairs spend almost twice more time on programming activities that the developers working alone.

It has identified that the developers tend to spend a noticeable part of their time on browsing web pages. In particular, the developers the developers working alone spend twice more time on browsing than the developers working in pairs. Since in the database there are stored only the headers of the visited pages they have been examined and a set of key words has been identified to distinguish business browsing from private one. The obtained results show that the developers applying different patterns of work tend to spend from 66% to 84% of time on business browsing. The highest number has been identified by the expert developers working in pairs.

It has been investigated how the developers switch between applications. It has been computed how much time on average the developers spend in each application before switching to another one and probabilities to switch among them. Further, the cycles of path of 2 applications when the developers switch from Visual Studio to one of the applications and then return to Visual Studio have been identified. The following numbers have been computed: the total number of the cycles, and the total and the average time spent in these cycles. It has been found that the developers tend to spend more time in the following three cycles: switching from Visual Studio to Outlook, or to Browser, or to Microsoft Messenger and then returning to Visual Studio. The reason why the developers switch so often from Visual studio to Outlook and back can be because they receive the requirements for their project via e-mail. Moreover, since the significant part of the browsing is devoted to Business browsing the developers might tend to switch so often from Visual Studio to Browser in searching for code examples and other information important for their work.

The cycles of path of 3 applications when the developers switch from Visual Studio to one of the applications, then again they switch to one of the applications, and only then they return to Visual Studio are scarce in the work of the developers. Though, it has been noticed that the developers tend to spend notably more time in the cycle when they switch from Visual Studio to Browser, then they switch to Outlook, and the return to Visual Studio than in any other cycle of path of 3 applications.

| | **Q1/M1, M2, M3** | **Q2/M1, M2, M3** | **Q3/M4** | **Q4/M5** | **Q5/M6** |
|---|---|---|---|---|---|
| ***Experts Solos*** | 34% of time devoted to Visual Studio | 9% of time devoted to Browser and 21% to Outlook | 28% of time devote to Private browsing and 72% to Business one | Most time (33%) devoted to ***Visual Studio-> Outlook-> Visual Studio*** | Most time (40%) devoted to ***Visual Studio -> Browser-> Outlook-> Visu-al Studio*** |
| ***Experts Pairs*** | 64% of time devoted to Visual Studio | 3% of time devoted to Browser and 13% to Outlook | 16% of time devote to Private browsing and 84% to Business one | Most time (37%) devoted to ***Visual Studio-> Outlook-> Visual Studio*** | Most time (30%) devoted to ***Visual Studio -> Browser-> Outlook-> Visu-al Studio*** |
| ***Mixes Pairs*** | 75% of time devoted to Visual Studio | 5% of time devoted to Browser and 6% to Outlook | 22% of time devote to Private browsing and 78% to Business one | Most time (25%) devoted ***to Visual Studio-> Outlook-> Visual Studio*** | Most time (56%) devoted to ***Visual Studio -> Browser-> Outlook-> Visu-al Studio*** |
| ***Novices Solos*** | 32% of time devoted to Visual Studio | 13% of time devoted to Browser and 23% to Outlook | 32% of time devote to Private browsing and 68% to Business one | Most time (41%) devoted to ***Visual Studio-> Outlook-> Visual Studio*** | Most time (99.6%) devoted to ***Visual Studio ->Browser-> Outlook-> Visu-al Studio*** |
| ***Novices Pairs*** | 49% of time devoted to Visual Studio | 10% of time devoted to Browser and 15% to Outlook | 34% of time devote to Private browsing and 66% to Business one | Most time (40%) devoted to ***Visual Studio-> Browser-> Visual Studio*** | Most time (10%) devoted to ***Visual Studio -> Browser-> Outlook-> Visu-al Studio*** |

|  | Q1/M1, M2, M3 | Q2/M1, M2, M3 | Q3/M4 | Q4/M5 | Q5/M6 |
|---|---|---|---|---|---|
| ***Solos*** | 33% of time devoted to Visual Studio | 10% of time devoted to Browser and 21% to Outlook | 29% of time devote to Private browsing and 71% to Business one | Most time (34%) devoted to ***Visual Studio-> Outlook-> Visual Studio*** | Most time (44%) devoted to ***Visual Studio -> Browser-> Outlook-> Visu-al Studio*** |
| ***Pairs*** | 60% of time devoted to Visual Studio | 5% of time devoted to Browser and 5% to Outlook | 24% of time devote to Private browsing and 76% to Business one | Most time (35%) devoted to ***Visual Studio-> Outlook-> Visual Studio*** | Most time (61%) devoted to ***Visual Studio -> Browser-> Outlook-> Visu-al Studio*** |

Table 7: “Summary on Research Questions and Metrics”

# 7. Conclusions and future work

The goal of this study was to observe daily work of 19 software developers from an IT department of a large Italian manufacturing company and to investigate how they use different tools. The data was collected during a time space of 10 month from October 2007 to July 2008. The daily work of the developers was analyzed taking into consideration their working experience in the company. The developers were applying spontaneous Pair Programming, i.e. when they found it appropriate. Compare to other studies conducted on Pair Programming the observation period is significantly large what makes it important for other studies in this area.

It has been identified that the developers working in pairs spend significantly more time on programming activities and less time on browsing and writing e-mails than the developers working alone. Moreover, all developers except experts working in pairs when they browse spend on average 70% of their time on business browsing. Experts working in pairs when they browse they spend 84% of their time for business browsing. It has also been noticed that the developers have cycles in their daily work. They tend to switch a lot from Visual Studio to either Outlook or Browser and back to Visual Studio.

In the future it is planned to repeat this experiment, taking into consideration the different kind of purposes people have when using a browser, in particular, when they use it for work and when for personal reasons.

## Bibliography

**[Arisholm et al., 2007]** Arisholm, E., Gallis, H., Dyba, T., and I.K. Sjoberg, D., “Evaluating Pair Programming with respect to System Complexity and Programmer Expertise,” IEEE Transactions on Software Engineering, 33(2), pp 65-86, February 2007

**[Baheti et al., 2002]** Baheti, P., Williams, L., Gehringer, E., Stotts, D., Smith, McC., J. “Distributed Pair Programming: Empirical Studies and Supporting Environments,” Technical Report TR02-009, Department of Computer Science, University of North Carolina at Chapel Hill, 2002

**[Basili et al., 1994]** Basili, V. R., Caldiera G., and Rombach H. D., "The Goal Question Metric Approach," Encyclopedia of Software Engineering, John Wiley & Sons, Inc., pp 528-532, 1994

**[Begel and Nagappan, 200**7**]** Begel, A. and Nagappan, N., “Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study,” In Proceedings of the First international Symposium on Empirical Software Engineering and Measurement, pp 255-264, September 2007

**[Begel and Nagappan, 2008]**Begel, A. and Nagappan, N., “Pair programming: what's in it for me?,” In Proceedings of the Second ACM-IEEE international Symposium on Empirical Software Engineering and Measurement, pp 120-128, October 2008

**[Braught et al., 2008]** Braught, G., Eby, L. M., and Wahls, T., “The effects of pair-programming on individual programming skill,” In Proceedings of the 39th Special Interest Group on Computer Science Education (SIGCSE), pages 200-204,, 40 (1), pp 200-204, February 2008

**[Canfora et al., 2007]** Canfora, G., Cimitile, A., Garcia, F., Piattini, M., and Visaggio, C. A., “Evaluating performances of pair designing in industry,” Journal of Systems and Software, 80 (8), pp 1317-1327, August 2007

**[Cockburn and Williams, 2001]** Cockburn, A. and Williams, L., “The costs and benefits of pair programming,” pair programming”. In G. Succi and M. Marchesi, editors, The XP Series. Extreme Programming Examined, Addison-Wesley Longman Publishing Co., pp 223-243, 2001

**[Coman et at., 2008]** Coman, I. D., Sillitti, A., and Succi, G., “Investigating the Usefulness of Pair-Programming in a Mature Agile Team,” In Proceedings of the 9th International Conference on Agile Porcesses and eXtreme Programming in Software Engineering (XP), pp 127-136, 2008

**[Coman et al., 2009 ]** Coman, I. D., Sillitti, A., and Succi, G., “A case-study on using an Automated In-process Software Engineering Measurement and Analysis system in an industrial environment,” In Proceedings of the 31st international Conference on Software Engineering, pp 89-99, May 2009

**[Chong and Hurlbutt, 2007]** Chong, J. and Hurlbutt, T. 2007. “The Social Dynamics of Pair Programming,” In Proceedings of the 29th International Conference on Software Engineering, pp 354-363, May 2007

**[Fronza et al, 2009]** Fronza, I., Sillitti, A., Succi, G., “Modeling Spontaneous Pair Programming when New Developers Join a Team,” XP 2009, pp 242 – 244, 2009

**[Gallis et al., 2003]** Gallis, H., Arisholm, E., and Dybå, T., “An Initial Framework for Research on Pair Programming,” In Proceedings of the International Symposium on Empirical Software Engineering, p 132, September – October 2003

**[Heiberg et al, 2003]** Heiberg, S., Puus, U., Salumaa, P., and Seeba A, “Pair-Programming Effect on Developers Productivity,” In Proceeding of the 4th International Conference on Agile Processes and eXtreme Programming in Software Engineering (XP), p 1016, 2003

**[Hulkko and Abrahamsson, 2005]** Hulkko, H. and Abrahamsson, P., “ A multiple case study on the impact of pair programming on product quality,” In Proceedings of the 27th international Conference on Software Engineering, pp 495-504, May 2005

**[Lui and Chan, 2003]** Lui, K. M and Chan, K. C. C., “When does a Pair Outperform Two Individuals?,” In Proceedings of the 4th international Conference on Extreme Programming and Agile Processes in Software Engineering, pp 225 – 233, May 2003

**[Lui and Chan, 2006]** Lui, K. M. and Chan, K. C., “Pair programming productivity: Novice-novice vs. expert-expert,” International Journal on Human-Computer Studies, 64(9), pp 915-925, September 2006

**[McDowell et al., 2002]** McDowell, C., Werner, L., Bullock, H., and Fernald, J., “The effects of pair-programming on performance in an introductory programming course,” In Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, pp 38-42, February - March 2002,

**[McDowell et al., 2003]** McDowell, C., Hanks, B., and Werner, L., “Experimenting with pair programming in the classroom,” In Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education, pp 60-64, June – July, 2003

**[Nawrocki et al.,2005]** Nawrocki, J., Jasiñski, M., Olek, L., Lange, B., “Pair programming vs. side-by-side programming,” In Proceedings of EuroSPI, pp 28–38, November 2005

**[Nawrocki and Wojciechowski, 2001]** Nawrocki, J., Wojciechowski, A., “Experimental evaluation of pair programming,” In Proceedings of the 12th European Software Control and Metrics Conference, pp 269–276, April 2001

**[Phaphoom, 2010]** Phaphoom, N., “Pair Programming and Software Defects. A Case Study”, Master Thesis, Free University of Bozen Bolzano, 2010

**[Sillitti et al., 2003]** Sillitti, A., Janes, A., Succi, G., and Vernazza, T., “Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data,” In Proceedings of the 29th Conference on EUROMICRO, pp 336 – 342, 2003

**[Strunk and White, 2004]** Strunk, W. and White, E. B., “The Elements of Style”, 4th edition, Allyn and Bacon, 2004

**[Succi et al., 2002]** Succi G., Pedrycz W., Marchesi M., and Williams L., “Preliminary analysis of the effects of pair programming on job satisfaction,” In Proceedings of 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002), 2002

**[Vanhanen and Korpi, 2007]** Vanhanen, J. and Korpi, H., “Experiences of Using Pair Programming in an Agile Project,” In Proceedings of the 40th Annual Hawaii international Conference on System Sciences, January 2007

**[Williams et al., 2000]** Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R., “Strengthening the Case for Pair Programming,” IEEE Software, 17 (4), pp 19-25, July 2000

**[Williams and Kessler, 2000]** Williams, L. and Kessler, R. R., “The Effects of ‘Pair-Pressure’ and ‘Pair-Learning’ on Software Engineering Education,” In Proceedings of the 13th Conference of Software Engineering Education and Training, p 59, 2000

**[Wonnacott and Wonnacott, 1990]** Wonnacott, T. H. and Wonnacott, R. J., “Introductory Statistics”, Wiley, 1990

## Appendix A

Table 8: “Overview of existing studies on Pair Programming”

| Authors | Subjects | Goal of the experiment | Results |
|---|---|---|---|
| “Pair Programming: What’s in it for Me? ,” Andrew Begel, Nachiappan Nagappan (2008) | 487 surveys. | To continue the precedent study (Begel et al., 2007). | PP allows the introduction of fewer bugs, spreading code understanding and over-all higher quality of the produced code.<br><br>Disadvantages of PP are cost-efficiency , work time scheduling difficulties and personality conflicts. |
| “The effect of Pair Programming on Individual Programming Skill,” Grant Braught, L. Marlin Eby, Tim Wahls (2008) | 151 students (77 paired and 74 individual). | To measure the effects of PP on the development of individual programming ability, comparing solo and pair programmers. | Students with lower SAT scores were able to achieve higher lab practica when using PP.<br><br>Students at all SAT levels who pair-programmed were more likely to complete the course successfully. |
| “Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise,” Erik Arisholm, Hans Gallis, Tore Dyba, Dag I. K. Sjoberg (2007) | 295 junior, intermediate and senior professional Java consultants. | To detect if PP reduces the time required to solve tasks correctly or increases the proportion of correct solutions. | Junior pair programmers achieved a significant increase in correctness comparing to the individuals and achieved approximately the same degree of correctness as senior individuals. |

| **Authors** | **Subjects** | **Goal of the experiment** | **Results** |
|---|---|---|---|
| "Usage and Perception of Agile Software Development in an Industrial Context: An Exploratory Study," Andrew Begel, Nachiappan Nagappan (2007) | 491 professionals (Microsoft developers, testes and managers who directly involved in the development of software). | To evaluate communication between team members, speed of releases and flexibility. | Advantages of Agile Software development: Improved communication between team members, quick releases and increased flexibility of Agile designs. |
| "The Social Dynamics of Pair Programming," Jan Chong, Tom Hurlbutt (2007) | 10 professional programmers. | To investigate how professionals perform when working in pairs. | Pairs were more efficient when both programmers took on driver and navigator responsibilities.<br><br>Equipping pair programmers with dual keyboards facilitates the rapid switching of keyboard control.<br><br>Recommendations: both programmers should be on the same level of knowledge. Pair rotation should be avoided late in a task. |

| Authors | Subjects | Goal of the experiment | Results |
|---|---|---|---|
| “Experiences of Using Pair Programming in an Agile Project,” Jari Vanhanen, Harri Korpi (2007) | 4 professional developers. | To develop an internal reposting system for the company using Java technologies and to pilot agile practices. | The driver rarely noticed defects during coding but the released code contained almost no defects.<br><br>Test-driven development and design in pairs probably decreased defects.<br><br>The developers considered that PP improved quality and knowledge transfer and was better suited for complex tasks than for simple. |
| “Evaluating Performances of Pair Designing in Industry,” Gerardo Canfora, Aniello Cimitile,Felix Garcia, Mario Piattini, Corrado Aaron Visaggio (2006) | 18 professional programmers (5 pairs and 8 individual programmers). | To investigate how PP affects system design. The quality of the result is evaluated by 2 independent evaluators. | Designing in pairs decreases productivity but increases the quality of the product. |
| “Pair Programming Productivity: Novice-Novice vs. Expert-Expert,” Kim Man Lui, Keith C. C. Chan (2006) | 40 part-time master’s students with full-time jobs. | To perform a repeated programming experiment (subjects repeatedly write the same program). | PP is more effective to increase the productivity of novices than of experts. |
| “A Multiple Case Study on the Impact of Pair Programming on Product Quality,” Hanna Hulkko, Pekka Abrahamsson (2005) | 4 software development projects. | To study the impact of PP on software product quality. | PP may not necessarily provide as extensive quality benefits as suggested in literature and on the other hand does not result in consistently superior productivity when compared to solo programming. |

| **Authors** | **Subjects** | **Goal of the experiment** | **Results** |
|---|---|---|---|
| "Pair Programming vs. Side-by-Side Programming," Jerzy R. Nawrocki, Michal Jasinski, Lukasz Olek, and Barbara Lange (2005) | 25 students. | To compare two styles of PP: XP-like and side-by-side (SbS) programming and solo programming. | 55% of the students preferred collaborative programming (SbS or XP approach) to individual. 40% had the opposite opinion and 5% had mixed feelings. Of those 55% SbS approach was preferred by 70% of the students and XP by 30%. 48% of the students working in pairs were satisfied in with their own code and 36% not. 45% were satisfied about partner's code and 45% not. The effort for SbS is smaller than for XP but the effort of individual code maintenance is greater for SbS than for XP. |
| "Pair-Programming Effect on Developers Productivity," Sven Heiberg, Uuno Puus, Priit Salumaa, and Asko Seeba (2003) | 110 students. | To verify how PP affects programmer's technical productivity. Students were divided to groups and into pairs inside the groups. One group was using PP and another group was using traditional teamwork technique. The experiment was divided into 2 phases. | Phase1: 1.7 times more pair-programmers passed first test cases than non-pair programmers.<br><br>Phase1: the average number of passed test cases per pair was 1.9 times higher at pair-programmers.<br><br>Phase2: non-pair-programmers passed no test cases. |

| **Authors** | **Subjects** | **Goal of the experiment** | **Results** |
|---|---|---|---|
| "When does a Pair Outperform Two Individuals? ," Kim Man Lui and Keith C. C. Chan (2003) | 15 industrial programmers. | To measure productivity of Pair Programming. | Pair Programming achieves higher productivity when a pair writes a more challenging program where they should spend more time on design. Pair programmers outperform solo programmers when a problem is new to the developers. |
| "Experimenting with Industry's "Pair Programming" Model in the Computer Science Classroom," Laurie A. Williams, Robert R. Kessler (2003) | 41 students. | To compare cycle time, productivity and quality results between pairs and individuals. | Pairs are not less productive then individuals. |
| "The Effects of Pair-Programming on Performance in an Introductory Course," Charlie McDowell, Linda Werner, Heather Bullock, Julian Fernald (2002) | ~600 students. | To investigate the effects of PP on student performance in an introductory programming class. | Students working in pairs produced code of better quality, completed the course at higher rates and performed about s well on the final exam as students who programmed independently. |
| "Preliminary Analysis of the Effects of Pair Programming on Job Satisfaction," Giancarlo Succi, Michele Marchesi, Witold Pedrycz, Laurie Williams (2002) | 108 responses on a questionnaire of 54 developers using PP and 54 developers not using it. | To analyze the effects of Pair Programming on job satisfaction. | PP has a significant, positive influence on the satisfaction of developers. |

| **Authors** | **Subjects** | **Goal of the experiment** | **Results** |
|---|---|---|---|
| "Distributed Pair Programming: Empirical Studies and Supporting Environments," Prashant Baheti, Laurie Williams, Edward Gehringer, David Stotts, Jason McC. Smith (2002) | 132 students divided into collocated teams without pairing (9 groups), collocated teams with pairs (16 groups), distributed team without pairs (8 groups), distributed team with pairs (5 groups). | To compare the different working arrangements of student teams developing object-oriented software. | It is feasible to develop software using distributed pair programming and the resulting software is comparable to software developed in collocated or virtual teams. |
| "The Costs and Benefits of Pair Programming," Alistair Cockburn, Laurie Williams (2001) | | To sum up all advantages of pair programming with respect to economics, satisfaction, design quality, continuous reviews, problem solving, learning, team building and communication, staff and project management. | The befits of PP are the following: many mistakes are detected when they are typed, the end defect content is statistically lower, the designs are better and size of code is smaller, the problems are solved faster in teams, the developers learn more in pairs, multiple people understand each part of the system, developers get experience in working together, developers are more satisfied about their work. |

| Authors | Subjects | Goal of the experiment | Results |
|---|---|---|---|
| “Experimental Evaluation of Pair Programming,” Jerzy Nawrocki, Adam Wojciechowski (2001) | A group of 21 students divided into tree sub-groups:<br>Personal Software Process (PSP) – 6 programmers;<br>XP for single programmers (XP1) – 5 programmers;<br>XP-like programming (XP2) – 5 pairs of programmers. | To evaluation pair programming. | Almost no difference between XP1 and XP2 what implies that PP is rather expensive technology. Experimentation and test-centred thinking reduces development time. PP is more predictable than individual one. XP1 is the most efficient programming technology, while PSP and XP2 are more or less the same. |
| “Strengthening the Case for Pair-Programming,” Laurie Williams, Robert R. Kessler, Ward Cunningham, Ron Jeffries (2000) | 41 students (13 solo programmers and 14 pairs). | To detect if PP helps in speeding up development and improving software quality. | The code produced by pairs passed more of the automated post-development test cases. When working in tandem programmers were able to complete their assignments 40-50% more quickly. |

| **Authors** | **Subjects** | **Goal of the experiment** | **Results** |
|---|---|---|---|
| “The effects of “Pair-Pressure” and “Pair-Learning” on Software Engineering Education,” Laurie A. Williams, Robert R. Kessler (2000) | 10 collaborative pairs of students. | To measure Pair-Pressure on quality, on students and on teaching staff. | All the projects were delivered on time and were of very high quality – the average grade was 98%. The same group of students when working alone had average grade 78.1%. The students performed much more consistently and with higher quality in pairs then they did individually. The students were extremely positive about their collaborative experience and “Pair-Pressure” according to anonymous surveys. “Pair-Learning” reduced the workload of the teaching staff since the students could solve more questions with their partners. |

# Appendix B

Figure 7: “Usage of tools by experts working alone”

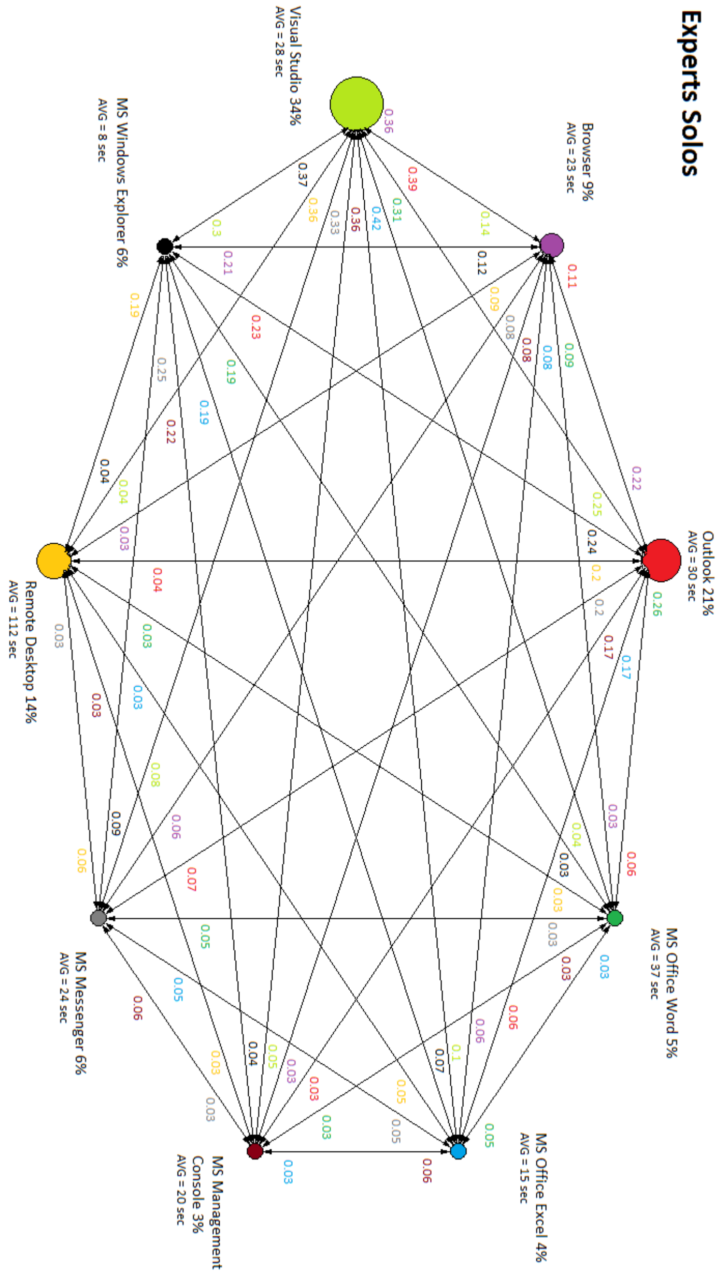


Figure 8: “Usage of tools by experts working in pairs”

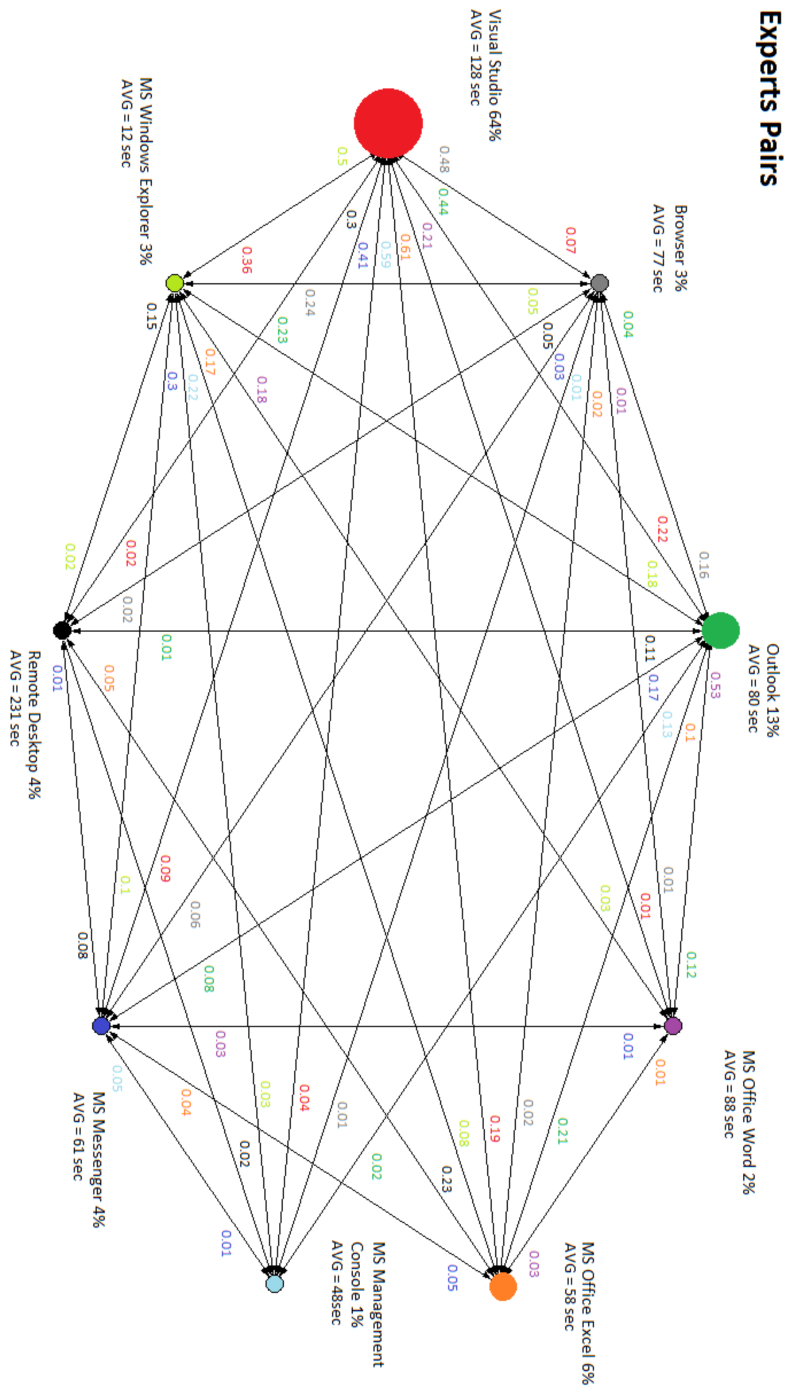


Figure 9: “Usage of tools by novices working alone”

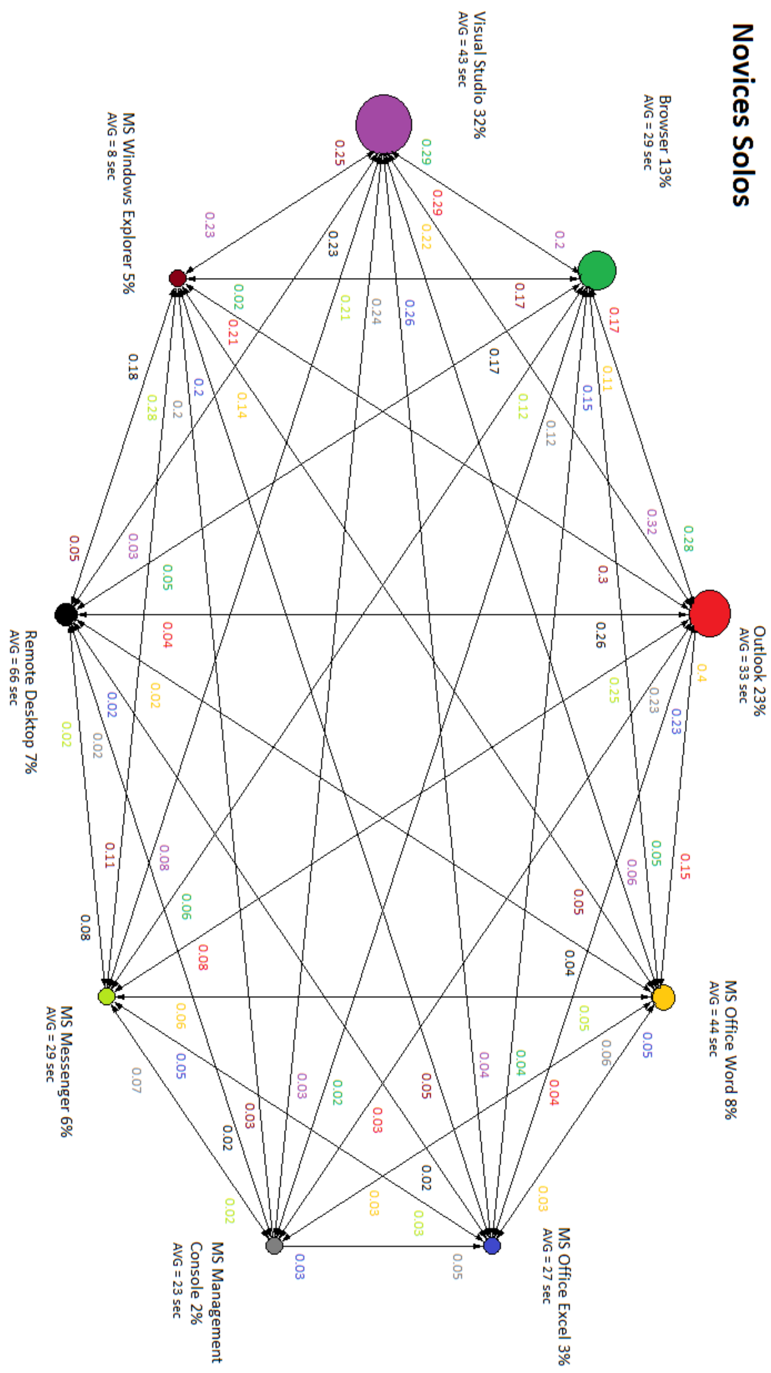


Figure 10: “Usage of tools by novices working in pairs”

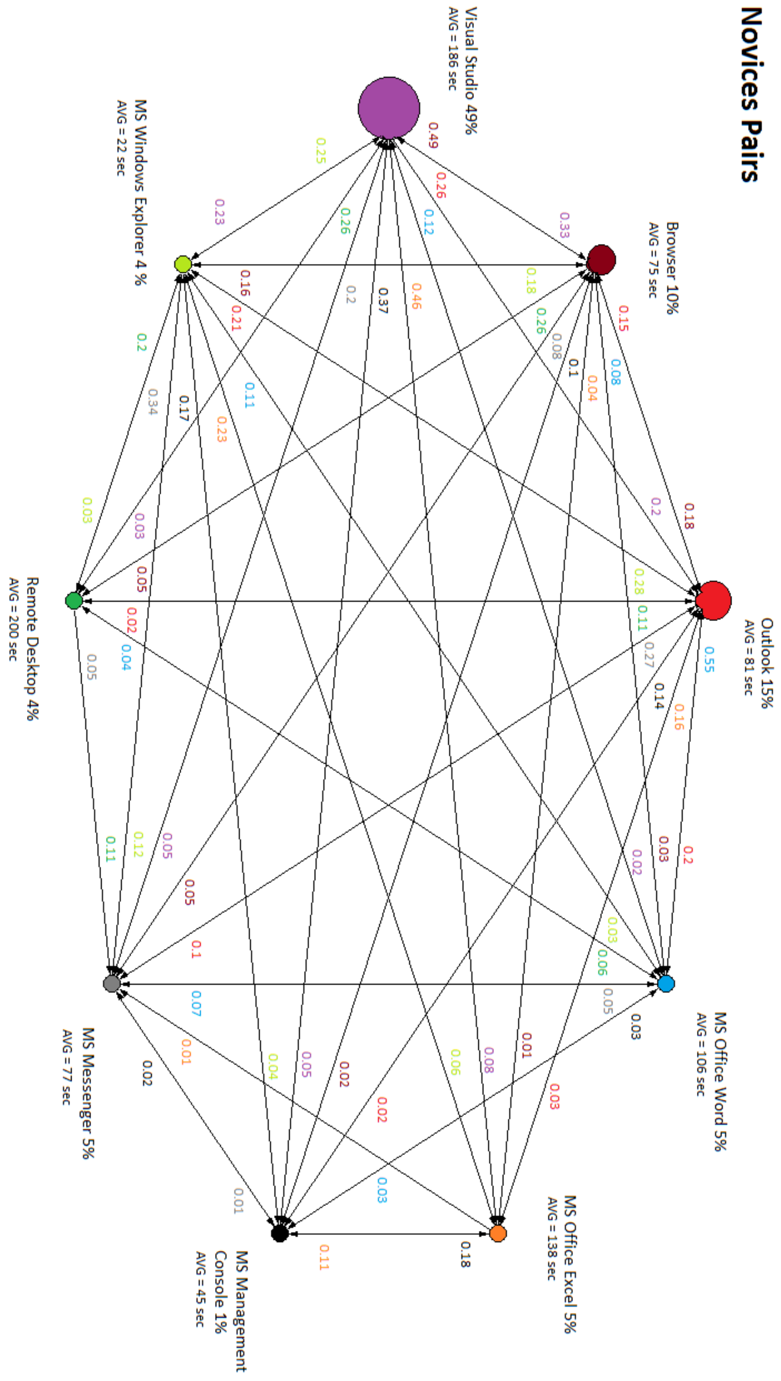


Figure 11: “Usage of tools by mixed pairs”

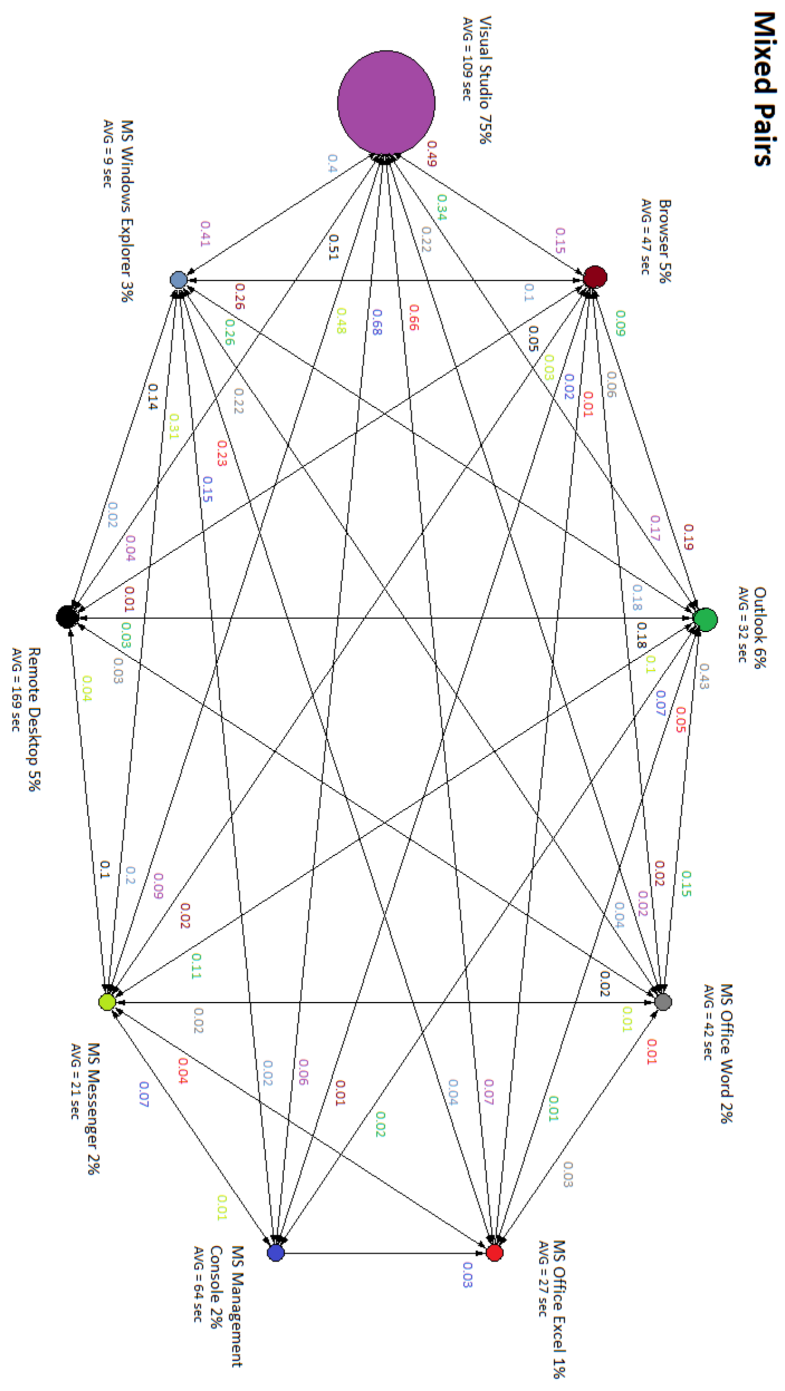


Figure 12: “Usage of tools by developers working alone”

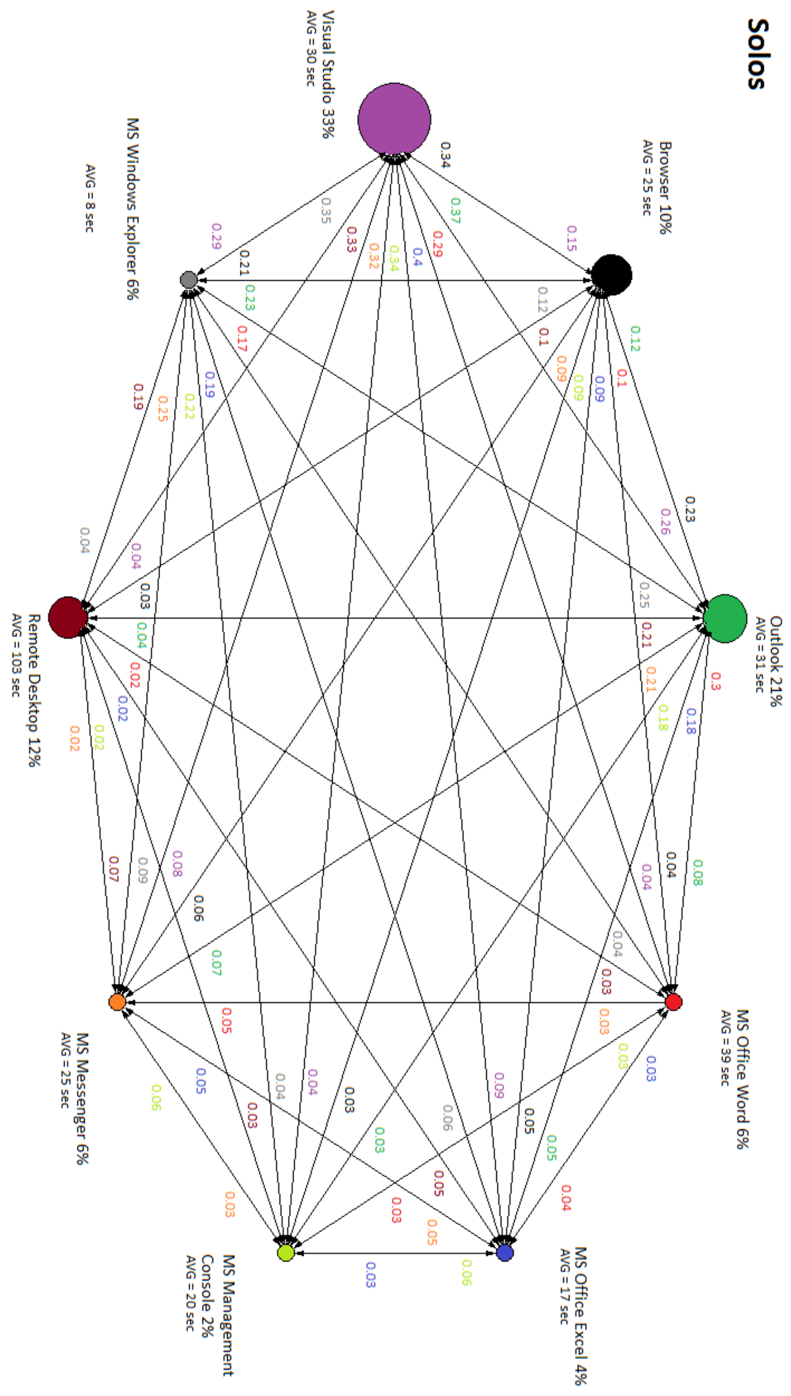
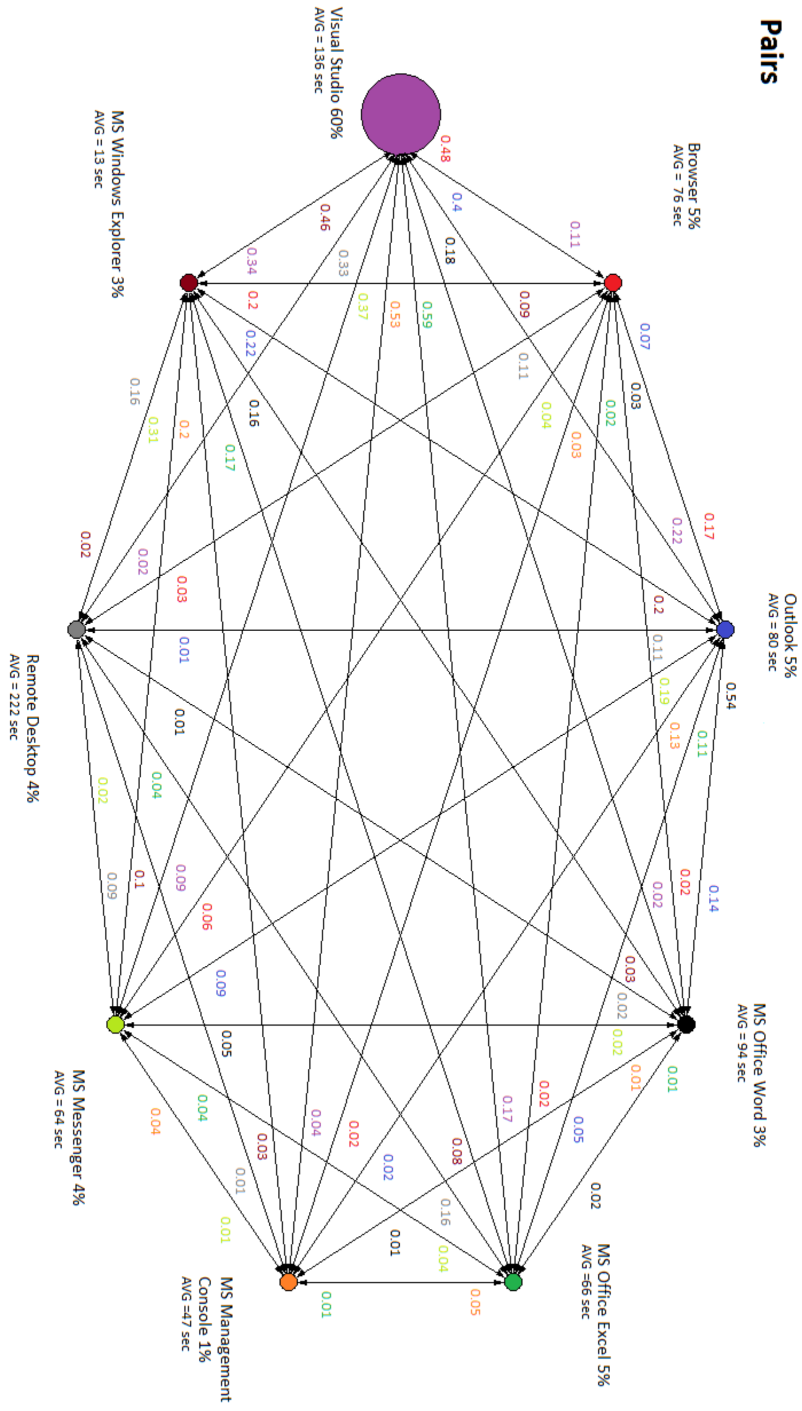


Figure 13: “Usage of tools by developers working in pairs”


Pairs
Visual Studio 60%
AVG = 136 sec
Browser 5%
AVG = 76 sec
MS Windows Explorer 3%
AVG = 13 sec
Outlook 5%
AVG = 80 sec
Remote Desktop 4%
AVG = 222 sec
MS Office Word 3%
AVG = 94 sec
MS Messenger 4%
AVG = 64 sec
MS Office Excel 5%
AVG =56 sec
MS Management Console 1%
AVG =47 sec