Exploring Developer's Tool Paths in Agile Environments

Jelena Vlasenko

Free University of Bolzano/Bozen

Piazza Domenicani – Domenikanerplatz, 3 I – 39100 Bolzano – Bozen, Italy
+39 0471 016138

Jelena.Vlasenko@stud-inf.unibz.it

Abstract

Many studies have been conducted to understand how people develop software. Still, software development process is mostly unclear. In this study we propose an idea of a cycle in daily work of developers. We investigate how the developers distribute their time and navigate among tools during their daily work. This understanding would be useful for identifying effective strategies for improvement, both of the development processes and of how computers and tools within computers are designed and used. Data for this have been collected non-invasively from team of professional software developers and represents a time span of 10 months.

Categories and Subject Descriptors

D.2.9 [Software Engineering]: Management – *productivity, programming teams*

General Terms

Management, Measurement, Human Factors.

Keywords

Cycles of work, non-invasive data collection, tool usage

1. The Research Problem and Motivation

In the last several years there have been countless studies on the positive effects of Pair Programming (PP). The claimed benefits of PP are several, including:

- reducing defect rate [24; 31; 32];
- improving design [4];
- increasing productivity [15; 20];
- improving the structure of a code [34];
- shortening the time-to-market [7; 10; 14; 23; 24; 25; 30];
- enhancing knowledge transfer and team communication [3; 5; 7; 8; 30; 31; 33];
- increasing job satisfaction [29] etc.

However, there are also studies that contradict these results, especially regarding productivity and cost-efficiency. [22] indicates no positive effect on development time. [16] evidences inconsistencies on the effects of PP and quality of the resulting software product. A large industrial case study [1] evidences that neither PP reduces time required to correctly perform change tasks nor increases the percentage of correct solutions. Results of the survey conducted by [3] in Microsoft indicate that there is a high level of skepticisms over efficiency of PP inside the organization. Altogether, there is a lack of precise qualifications of the effects of PP in real working environments. This work attempts to move in this direction taking a slightly different approach. It is not another empirical evaluation of effects of PP on various relevant aspects of software projects, such as defect rates, time-to-market, etc. Rather, it aims at studying empirically the effects of PP on the specific actions that people take when working. The understanding of such effects could then create a solid framework for understanding the consequential effects of PP on the other variables. This approach follows the lines of the approach of analytical generalization via the principle of causal explanation as detailed by [26] and applied in Software Engineering by [13]. In particular, we try to determine in a specific case study how PP affects how much developers stay focused on their tasks. Being focused on a task has been proven to have several positive effects on the work being performed, as explained in Section 2. We consider in particular three aspects of development sessions: how much time people spend in directly productive activities, how often they switch between tools, and how long they stay focused on a task before switching to another one. We analyze the work of a real software development team using a non-invasive measurement system. The team is part of an ICT department of a large Italian manufacturing company. It includes 17 developers and uses some of the practices of Extreme Programming [2] and a customized version of PP – Spontaneous Pair Programming. Developers decide themselves when and with whom to work. The data for this study is collected non-invasively by means of PROM [27]. PROM is an application that runs on a developer's machine and collects data about what people do without the need of any personal intervention, thus not altering their working habits. The result of this case study is that PP notably increases the focus people have on their work. When working in pairs, developers spend significantly more time in directly productive activities, switch less often between tools, and spend longer time on a task before switching to another task. Needless to say, this is one case study; replications are needed to identify strong causal relationships.

Despite almost 50 years of studies, it is still mostly unclear how people develop software, especially how they interact with each other and with computers to achieve the desired results. Such understanding would enable the definition of very effective strategies for improvement, both of the development processes and of how computers, and tools within computers, are designed and used. In particular, it is still mostly unknown how people use tools to work on their tasks. We know only that people tend to use classes of tools for specific tasks: say, they typically use IDE (Visual Studio, Eclipse, ...) for development, Word Processors

(Microsoft Word, OpenOffice, …) for text processing, etc. Still, we do not know what they exactly do with their tools – for instance, they could do some editing of C code with OpenOffice just because they have OpenOffice open and they would not like to launch an IDE. Moreover, developers tend to use multiple different tools to achieve desired results. For instance, often while developing the code they use both the IDE and the browser: they use the IDE to write and test the code and the browser to gather information. Still, very limited research has been done so far on how multiple tools cooperate together to achieve specific goals.

This problem contains two very important aspects. The **first** is on how data are collected on tool usage, so that it is possible to determine exactly which tool a developer is using at a given time. This has been addressed using tools like PROM [27]. The main advantage of PROM is that the developer does not even perceive that his/her activities are being recorded. Thus, using PROM for data collection allows to obtain the data that represent developer's real activities in real working environment. Still, as we know from statistics, having massive amount of data, such as the one coming from PROM, is useless unless it is properly structured for subsequent analysis. The **second** aspect is therefore how to structure the data on tool usage. This problem has been approached recently with L-Graphs [12, 28]. An L-Graph is a directed, labeled graph containing information on individual tool usage and on the transitions between tools. However, L-Graphs do not represent in details how different tools are used together to perform tasks. Such information is especially important to establish process improvement initiatives and to design more effective tools. To this end we introduce the concept of "cycle of work."

This work is organized as follows. In Section 2 we present some of the existing work done in this area; Section 3 describes the proposed approach; Section 4 discussed the results we have obtained so far; Section 5 draws some conclusions and outlines the future research we plan to do in this area.

2. Background and Related Work

Keeping people focused on what they do is important in many knowledge-based activities, not only in software development. A number of studies have been conducted to investigate how different levels of attention affect the quality of work. Organizational and work psychologists have experimentally found that keeping a high level of attention results in better and faster work [9]. Furthermore, there are studies that show that in the modern society people using computers are exposed to the risk of losing attention [11]. This phenomenon is even more acute for software developers. Hereby, limited attention can cause both lower quality in specific work tasks and an overall reduction of social ability [17]. This reduction may severely impact the interactions with customers and colleagues, with negative consequences on the overall software development process. As a consequence of the existing works, two possible options about how to increase the level of attention have been proposed:

- an external stimulus that keeps the attention level high;
- an internal cognitive control mechanism that keeps the attention aligned with the priorities of the tasks to complete.

Existing research is mainly focused on externally-driven intrusions on individual workers, but the nature of work, the work environment, and the team configuration may all influence how workers handle both externally and self-initiated interruptions. In software development, PP can act both as an external stimulus and as an internal control mechanism. It can be an external stimulus because the two people working together exchange constantly words, gestures, emotions, etc. It can also be an internal stimulus because people feel often "morally" obliged not to waste time when working with others and to do their best in their work; existing works on PP have already evidenced that the developers working in pairs are more motivated to work harder and with better results because they do not want to look incompetent in front their colleagues with whom they are paired [19]. After all, ethnographic observations indicate that interruption length, content, type, occurrence time, and interrupter/interruptee strategies differ for pair programmers versus solo programmers [6].

Very limited research has been published on how developers use tools. Most of the existing works are focused on identifying purposes and drawbacks of the tools that are used by developers aiming to propose substitutes or enhanced versions of these tools. For example, [21] investigates which features are the most used and best implemented in the installed CASE tools. It has been found that developers are not satisfied with the existing CASE tools and most of the advanced features are not used. Furthermore, in [18] authors aim to identify how to improve the tools that are used by developers to increase productivity. It has been identified that there is a need for a tool that would help to explore existing software. Still, it has not been studied yet how the different tools are used together and how they interact. The idea proposed in [28] approaches this problem. In [28] authors focus on time distribution, frequencies of switching between tools, and average time to stay in a tool before switching. L-Graphs are used to visualize these variables. This idea has been applied in [12]. The results indicate that the developers doing Pair Programming are more focused on directly productive activities than the developers working alone.

3. Approach and Uniqueness

In this study we introduce a definition of a cycle and show some preliminary results. Informally, we say that a cycle occurs when a developer starting from an application moves to other applications and eventually returns to the starting one. More formally, we can say that a developer initially works on an application A_0. Then, while working, he switches to other applications A_1, A_2, … A_{n-1}, until he reaches an application An which is equal to A_0, with the constraint that A_i<>A_0, for i∈ [1, n-1]. We call n the size of the cycle. For instance, if a developer starts working in Visual Studio (A_0 in the definition), and then moves to the Browser (A_1), and eventually gets back to Visual Studio (A_2), we say that there is a cycle of size 2. Cycles convey the information of the tools that are actually needed to perform given tasks. In the case above, the cycle could imply that the developer was initially working in Visual Studio, then moved to the browser for instance to gather some information on a specific API, and then returned to Visual Studio to continue the development.

In this study, we are interested in cycles of size 2, 3, and 4 that originate in Visual Studio (the main development tool). These cycles reasonably represent the situation when developers move from the IDE to other tools to gather relevant information and eventually they get back to the IDE. In particular, we consider the distributions of these cycles, the effort spent on them, and the breakdown of cycles per application visited. Also cycles are automatically extracted from the data collected non-invasively by the PROM system.

Analyzing cycles could help to understand better how different people develop software, how they organize their working process, and how to improve it. If there is a very strong connection between tools it might be useful to ingrate them. Moreover, it could help to identify the cycles when developers are unproductive and to identify the reasons why it so. In this paper we propose only a research idea of and analyze some preliminary results.

4. Results and contributions

The dataset for this study comes from the same team as in [28] and represents a time span of 10 months: from October 2007 to July 2008. The data have been collected from a team of professional software developers working in a IT department of a large Italian manufacturing company which prefers to remain anonymous. The team is composed of 17 developers who have more than 15 years of programming experience. In [28] we identified 9 tools that covered more than 80% of the total time spent in all 26 tools. In this study we focus on the same 9 tools:
(1) Visual Studio;
(2) Browser;
(3) Outlook;
(4) Microsoft Office Word;
(5) Microsoft Office Excel;
(6) Microsoft Management Console;
(7) Microsoft Messenger;
(8) Remote Desktop;
(9) Microsoft Windows Explorer.

The analysis of cycles confirms that developers working in pair are more focused than the ones working alone. As mentioned, we focus our attention to cycles of size 2, 3 and 4 that originate in Visual Studio (the main development tool). Table 1 contains the average effort spent in cycles. We notice that when doing PP developers spend considerably more effort in each individual cycle, with the implication that the process is more focused and the acquisition of information is more accurate. These differences are significant at the 0.01 level.

Table 1. Breakdown of the Size 2 Cycles per Target Application Visited by Pairs and Solos (VS stands for Visual Studio)

Cycles	Pairs	Solos
VS → Browser → VS	8.9%	16.7%
VS→Outlook→VS	23.9%	30.9%
VS→Word→VS	1.9%	4.1%
VS→Excel→VS	27.1%	11.5%
VS→Messenger→VS	13.9%	9.4%
VS→Windows Explorer→VS	17.0%	18.4%
VS→Management Console→VS	4.4%	5.0%
VS→Remote Desktop→VS	2.9%	4.1%

Table 2 contains the breakdown of cycles per size. In both cases there is a prevalence of the shorter cycles of size 2. However, such prevalence is much higher when working in pairs, with a reasonable interpretation that when doing PP developers are more effective in directly identifying the information they need and in directly moving to such application, rather than navigating between tools before finding what is needed.

Table 2. Average Effort (in Seconds) Spent in Cycles by Pairs and Solos

	Size 2	Size 3	Size 4
Pairs	258	450	526
Solos	71	111	146

Table 3 details the breakdown of the cycles of size 2 per application to switch. When working alone developers cycle more than in pairs the most to gather information already existing:

- to Outlook, where the requirements are stored;
- to the web browser, probably checking APIs and code snippets.

When doing PP, on the other side, they cycle more than when alone to acquire additional information:

- to Excel, to evaluate possible hypothesis for computations;
- to the Messenger, to ask other people in the company details about the requirements they posed

Also in this case we can interpret the results in the direction that PP enables developers to make better usage of the information available and that the switch to other tools is motivated more to acquire new, specific information than to gather basic or already provided information.

Table 3. Breakdown of the Cycles per Size for Pairs and Solos

	Size 2	Size 3	Size 4
Pairs	88.9%	7.0%	4.1%
Solos	75.8%	16.9%	7.3%

5. Conclusions and Future Work

This study provides a novel contribution to the literature about PP focusing on how people approach software development when they work in pairs compared to when they work alone. According to our data, people change in a relevant way their working habits when they work in pairs and tend to focus more on productive activities. They have significantly longer and uninterrupted working sessions focusing more on the assigned tasks and reducing the needs of retrieving information from sources other than the partner (e.g., through the Internet). This study is just a preliminary investigation about the working habits that PP modifies and a further investigation about the effectiveness of such modifications is required to understand if and how much PP contributes to increase the quality of software products.

6. References

[1] E. Arisholm, H. Gallis, T. Dyba, and D. I.K. Sjoberg. Evaluating pair programming with respect to system complexity and programmer expertise. IEEE Trans. Softw. Eng., 33(2):65–86, 2007.
[2] K. Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, us ed edition, October 1999.
[3] A. Begel and N. Nagappan. Pair programming: what's in it for me? In ESEM '08: Proc. of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, pages 120–128, New York, NY, USA, 2008. ACM.
[4] G. Canfora, A. Cimitile, C. A. Visaggio, F. Garcia, and M. Piattini. Performances of pair designing on software evolution: a controlled experiment. In European Conference on Software Maintenance and Reengineering, 0:197–205, 2006.
[5] J. Chong and T. Hurlbutt. The social dynamics of pair programming. In ICSE '07: Proc. of the 29th international conference on Software Engineering, pages 354–363, Washington, DC, USA, 2007. IEEE Computer Society.
[6] J. Chong and R. Siino. Interruptions on software teams: a comparison of paired and solo programmers. In Proc. of the Conference on Computer Supported Cooperative Work, Banff, Alberta, Canada, November 04 – 08, 2006.

[7] A. Cockburn and L. Williams. The costs and benefits of pair programming. pages 223–243, 2001.
[8] I. D. Coman, A. Sillitti, and G. Succi. Investigating the Usefulness of Pair-Programming in a Mature Agile Team. In XP'08: Proc. of the 9th International Conference on eXtreme Programming and Agile Processes in Software Engineering, pages 10-14, Limerick, Ireland, 2008. Springer 2008.
[9] R. Dukas. Behavioural and ecological consequences of limited attention, Philosophical Transactions B of the Royal Society 357, 2002, pp. 1539–1547.
[10] T. Dyba, E. Arisholm, D. I. K. Sjøberg, J. E. Hannay, and F. Shull. Are two heads better than one? on the effectiveness of pair programming. IEEE Softw., 24(6):12–15, 2007.
[11] J. Falkinger. Limited Attention as the Scarce Resource in an Information-Rich Economy. The Economic Journal 118, 532, 2008, pp. 1596–1620.
[12] Fronza, I., Sillitti, A., Succi, G., Vlasenko, J., Does Pair Programming Increase Developers' Attention? In Proceedings of ESEC/FSE2011, Szeged, Hungary, September 2011
[13] J. E. Hannay and M. Jørgensen. The Role of Deliberate Artificial Design Elements in Software Engineering Experiments. IEEE Trans. Softw. Eng., 34(2):242-259, 2008.
[14] J. E. Hannay, E. Arisholm, H. Engvik, and D. I. K. Sjoberg. Effects of personality on pair programming. IEEE Trans. Softw. Eng., 36(1):61–80, 2010.
[15] S. Heiberg, U. Puus, P. Salumaa, and A. Seeba. Pair-Programming Effect on Developers Productivity. In XP'03: Proc. of the International Conference on Agile Processes and eXtreme Programming in Software Engineering, Genova, Italy, 2003. Springer.
[16] H. Hulkko and P. Abrahamsson. A multiple case study on the impact of pair programming on product quality. In ICSE '05: Proc. of the 27th international conference on Software engineering, pages 495–504, New York, NY, USA, 2005. ACM.
[17] N. Lavie, A. Hirst, J. W. de Fockert, and E. Viding. Load theory of selective attention and cognitive control. Journal of Experimental Psychology 133, 3, 2004, 339–354.
[18] Lethbridge, T.C. and Singer, J., "Understanding Software Maintenance Tools: Some Empirical Research," In Workshop on Empirical Studies of Software Maintenance, pp 157-162, 1997
[19] K. M. Lui and K. C. C. Chan. Pair programming productivity: Novice-novice vs. expert-expert. International Journal of Human-Computer Studies 64, 9, pp. 915-925, 2006.
[20] K. M. Lui, K. C. C. Chan, and J. Nosek. The effect of pairs in program design tasks. IEEE Trans. Softw. Eng., 34(2):197–211, 2008.
[21] Maccari, A., Riva, C., and Maccari F., "On CASE tool usage at Nokia," In Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE'02), pp 59-68, 2002
[22] J. Nawrocki and A.Wojciechowski. Experimental evaluation of pair programming. In Proc. European Software Control and Metrics Conf. (ESCOM),, 2001.
[23] J. T. Nosek. The case for collaborative programming. Commun. ACM, 41(3):105–108, 1998.
[24] M. Phongpaibul and B. Boehm. An empirical comparison between pair development and software inspection in Thailand. In ISESE '06: Proc. of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering, pages 85–94, New York, NY, USA, 2006. ACM.
[25] M. Phongpaibul and B. Boehm. A replicate empirical comparison between pair development and software development with inspection. In ESEM '07: Proc. of the First International Symposium on Empirical Software Engineering and Measurement, pages 265–274, Washington, DC, USA, 2007. IEEE Computer Society.
[26] W. R. Shadish, T.D. Cook, and D. T. Campbell, Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Hought- on Mifflin, 2002.
[27] A. Sillitti, A. Janes, G. Succi, and T. Vernazza. Collecting, integrating and analyzing software metrics and personal software process data. In EUROMICRO '03: Proc. of the 29th Conference on EUROMICRO, page 336, Washington, DC, USA, 2003. IEEE Computer Society.
[28] A. Sillitti, G. Succi, and J. Vlasenko. Toward a better understanding of tool usage. In ICSE 2011: Proc. of the 33th International Conference on Software Engineering, Honolulu, HI, USA, 21 – 28 May 2011.
[29] G. Succi, W. Pedrycz, M. Marchesi, and L. Williams. Preliminary analysis of the effects of pair programming on job satisfaction. In In Proc. of the 3rd International Conference on Extreme Programming (XP), pages 212–215, 2002.
[30] J. Vanhanen and H. Korpi. Experiences of using pair programming in an agile project. In HICSS '07: Proc. of the 40th Annual Hawaii International Conference on System Sciences, page 274b, Washington, DC, USA, 2007. IEEE Computer Society.
[31] J. Vanhanen and C. Lassenius. Effects of pair programming at the development team level: an experiment. In International Symposium on Empirical Software Engineering. 2005, page 336-345, 2005.
[32] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the case for pair programming. In IEEE Softw., 17(4):19–25, 2000.
[33] L. Williams and R. R. Kessler. Pair Programming Illuminated. Addison-Wesley Longman Publ. Co., Inc., Boston, MA, USA, 2002.
[34] S. Xu and V. Rajlich. Empirical validation of test-driven pair programming in game development. In ICIS-COMSAR '06: Proc. of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering,Software Architecture and Reuse, pages 500–505, Washington, DC, USA, 2006. IEEE Computer Society.