Exploring Developer's Tool Path

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

In this study we introduce the concept of “cycle” in daily work of software developers. A cycle is occurs when a developer working on a tool switches to another tool, or also to more, and eventually goes back to the first tool. Using the concept of cycle we explore how the developers distribute their time and navigate among different tools during their work. Analysing the cycles can be beneficial for identifying effective strategies for improvement both of the development processes and of how computers (and their tools) are designed and used.
The approach has been validated on data collected non-invasively from team of professional developers for 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

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 [1]. PROM is a tool that runs on a background on a developer's machine and collects data about his/her activities. 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. The **second** aspect is how to structure the data on tool usage. This problem has been approached recently with L-Graphs [2, 5]. 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.
This paper 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

Very limited research has been published on how developers use tools. Most of the existing works focus on identifying purposes and drawbacks of the tools aiming to propose substitutes or enhanced versions of these tools. For example, [4] 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 [3] 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 [5] approaches this problem. In [5] 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 [2]. The results indicate that the developers doing Pair Programming are more focused on directly productive activities than the developers working alone.

SPLASH'11 Companion, October 22–27, 2011, Portland, Oregon, USA.
ACM 978-1-4503-0942-4/11/10.

3. Approach and Uniqueness

In this study we introduce a definition of a cycle and show some preliminary results. A cycle is defined as follows: assuming that a developer is working a tool A, a cycle occurs when he/she switches to a tool B and possibly to other tool, and eventually returns to A. The number of steps involved in the cycle is called the size of the cycle: if a user starts in A, then switches to B, and then goes back to A, then the size of the cycle is 2. Cycles could be of any size: between two usages of the same tool there can be any number of other tools usages. However, the meaningfulness of a cycle is only when we can assume that there is a cause in the use of the original tool that triggers the move to other tools and eventually the come back. In this work we introduce a concept of a cycle and show some preliminary results. In the future work we want to keep a track of history: we want to identify if a developer works on a task T in a tool A, then he switches to a tool B to continue working on the task T, and then goes back to the tool A to continue working on the same task. Analyzing cycles could help to understand better how different people develop software, how they organize their working process, and how to improve it [7]. One of the improvement proposals could be the following: if there is a very strong connection between tools it might be useful to ingrate them so that developers would not loose time and focus when switching. Another possible benefit of studying cycles is to it identify those cycles when developers are unproductive and to understand the reasons why it so.

4. Results and Contributions

The data for this study 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 dataset represents a time frame of 10 months: October 2007 – July 2008. The team is composed of 17 developers all with more than 15 years of experience. In our previous work [5] we found that during this period the developers were using 26 distinct tools and that 80% of the total time was devoted to the following 9 tools: Visual Studio, Browser, Outlook, Word, Excel, Microsoft Management Console, Microsoft Messenger, Remote Desktop, and Windows Explorer. In this study we are interested only in these 9 most time consuming tools.
To compute cycles, we extracted switching sequences in a time-based order. Each record in the sequence consists of two parameters: a tool itself and the time spent in this tool before switching to another one. Then we calculated all the possible

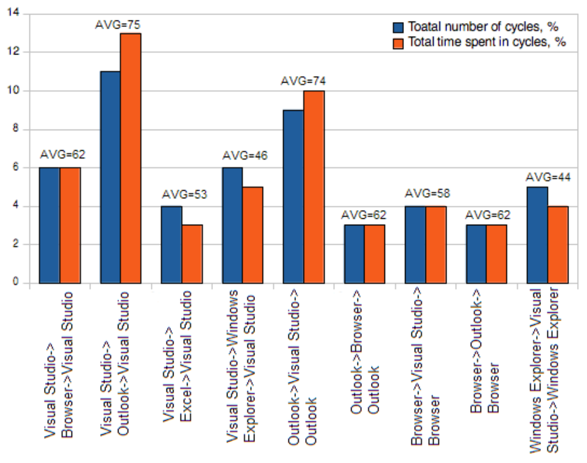

Figure 1: Cycles in daily work of software developers

cycles of size 2. We got 72 cycles. For these cycles we computed the following 3 parameters: percentage of time (represented by y-axis in Figure 1) developers spent in each cycle, percentage of cycles (represented by y-axis in Figure 1), and average time (AVG in the Figure 1) developers spent in a cycle. Percentage of time spent in a cycle, total number of cycles, and average time spent in a cycle were computed with respect to the whole time frame. We found that in the given time frame the developers spent more time in the cycles where Visual Studio was involved either as a starting or as a middle tool. Figure 1 represents 9 most time-consuming cycles. Other cycles consume less than 3% of the total time. The obtained results indicate that the most time consuming cycles are those where are involved the following 3 tools: Visual Studio, Browser, and Outlook. From the developers we know that they receive their requirements via Outlook what explains a very strong connection between email client and Outlook [6]. Furthermore, we noticed that the developers spend a lot of time switching from Visual Studio to Browser and back. We assume that the reason why the developers spend so much time in this cycle is because Browser is a source of knowledge and is essential for their daily work.

5. Conclusions and Future Work

In this study we introduced an idea of a cycle in daily work of software developers. A cycle is a unit of work when a developer goes to one tool, then starts switching between other tools, and eventually returns to the starting tool. The obtained results indicate that the developers spend a notable part of their time in those cycles where are involved the following 3 tools: Visual Studio, Browser, and Outlook. The idea proposed in this study helps to understand better how people use tools when developing software. In future work we want to expand the idea of a cycle and to consider also task history when switching between tools: i.e. we want to identify which tools developers use to work on particular tasks. This information could be helpful to improve software development process. It would be possible to identify the tools that are mainly distractive – when developers start switching between tools and loose focus, and the tools that really needed for work purposes and how to improve them to facilitate the process.

6. References

[1] Abrahamsson, P., Moser, R., Pedrycz, W., Sillitti, A., Succi, G. Effort Prediction in Iterative Software Development Processes – Incremental Versus Global Prediction Models. In Proceedings of ESEM2007, Madrid, Spain, 2007

[2] Fronza, I., Sillitti, A., Succi, G., Vlasenko, J. Does Pair Programming Increase Developers' Attention? In Proceedings of ESEC/FSE2011, Szeged, Hungary, September 2011

[3] 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

[4] Maccari, A., Riva, C., and Maccari F., "On CASE tool usage at Nokia," In Proceedings of ASE2002, pp 59-68, 2002

[5] Sillitti, A., Succi, G, Vlasenko, J. 2011. Toward a better understanding of tool usage. In Proceedings of ICSE2011, Honolulu, Hawaii, May 2011

[6] Sillitti, A., Ceschi, M., Russo, B., Succi, G., Managing uncertainty in requirements: a survey in documentation-driven and agile companies. In Proceedings of ESEM2005, Como, Italy, 2005

[7] Succi, G., Pedrycz, W., Liu, E., Yip, J. Package-oriented software engineering: a generic architecture. IT Professional, 3(2), pp 29 – 36, 2002