FACTORS AFFECTING PERSONAL SOFTWARE DEVELOPMENT PRODUCTIVITY: A CASE STUDY WITH PSP DATA

Mushtaq Raza, João Pascoal Faria
INESC TEC/Faculty of Engineering University of Porto
Porto, Portugal
uomian49@yahoo.com
jpf@fe.up.pt

ABSTRACT
Understanding the factors that affect the productivity of software developers and may cause productivity variations among individuals and projects is important for anyone interested in improving software engineering performance and estimates, and in particular for users of high-maturity processes, such as the Personal Software Process (PSP) and the Team Software Process (TSP). In order to contribute to the understanding of the personal and non-personal factors that affect productivity, we analyzed the data from more than 3000 developers that concluded successfully the 10 projects of the PSP for Engineers I/II training course. Regarding non-personal factors, by conducting a detailed per-phase analysis, we found significant variations of productivity among projects that can be partially explained by process changes. Regarding personal factors, we found significant variations among individuals that can be partially explained by personal experience.

KEY WORDS
Productivity; factors; software development; PSP

1. Introduction

Software engineers are faced with the need to develop increasingly complex software systems with increasing quality, agility and efficiency, but many lack the self-management skills that, together with the more technical skills, are needed to succeed. The Personal Software Process (PSP) [1] is an example of a process framework that was specifically designed by the Software Engineering Institute (SEI) to help individual software engineers improve their performance and become effective team members following the additional team building and team management practices of the Team Software Process (TSP). The PSP embodies a rich measurement framework and a set of effective project management, quality management and process improvement practices. These practices are introduced stepwise in a training environment, where the engineers have to complete a sequence of small projects, applying increasingly elaborated process versions or levels. In the classic "PSP for Engineers I/II" version of the training course there are 10 projects (the same for all individuals) and 6 process levels. By the end of the training, and at regular times afterwards, PSP developers should analyze their gathered project data to understand their current performance and decide on improvement actions. However, little tool support exists to help PSP developers in that kind of analysis, which is problematic because of the amount of data to analyze and of the expert knowledge required.

To overcome that problem, we are conducting a research project which goal is to develop models and tools to help PSP students and practitioners analyze their performance, namely, identify performance problems, root causes and possible improvement actions. In previous work [2][3][4], we identified a set of factors affecting directly or indirectly the time estimation performance, together with indicators and recommended values for all the variables involved. In order to arrive at a similar model for the productivity performance, the main goal of this paper is to determine which factors affect the productivity of PSP developers, based on the analysis of the PSP for Engineers I/II course data. The knowledge of those factors may be of interest not only for performance analysis and improvement, but also for improving project estimates.

The rest of the paper is organized as follows. After a short review of the state of the art related with the specific goals and context of this paper in section 2, we clarify the research questions and methods in section 3. After providing some background information needed to interpret the data regarding the PSP and the PSP for Engineers I/II course in section 4, we present the main results of the analysis and contributions of this paper in sections 5 (for non-personal factors) and 6 (for personal factors). We conclude the paper in section 7 with a summary of major findings and points for future work.

2. State of the art

In this section we review productivity measures and factors affecting productivity mentioned in the literature.

Software development productivity is usually measured in function points per time unit or lines of code (LOC) per time unit [5][6][7]. However, both productivity measurement techniques have some limitations. On one hand, the measurement of function points remains subjective even after the completion of the software development project. On the other hand, productivity measures based on LOC have limitations due to the lack of counting standards and the dependence on the programming

language [8]. In our case study, considering the base measures gathered in the PSP, we used LOC/hour as the productivity measure of individual engineers. Whilst the usage of this metric may be criticized for comparing the productivity of different engineers, particularly when they use different programming languages, that's not the main purpose in the PSP. Instead, it is used mainly to analyze the evolution of personal productivity along a sequence of projects, and to estimate personal development effort based on the personal historical productivity and a size estimate for the current project. The PSP effort estimation approach is based on the observation that is, although there are significant differences of productivity between developers, each developer tends to have a stable productivity [1].

Regarding factors affecting productivity, from previously published studies [9], it is known that the engineers' productivity during the PSP training decreases in the first projects and recovers in the last projects. An explanation that is usually mentioned [10] is that the initial decrease is caused by the introduction of process changes, and recovery occurs as the new processes or process components are practiced. But, to our knowledge, there is a lack of detailed studies providing evidence in favor of that explanation. Besides that, significant variations of productivity among individuals are often observed [11], but, to our knowledge, no detailed studies exist that analyze the causes of those variations in the context of the PSP.

In a broader context, several factors affecting individual developers productivity are mentioning in the literature, such as process used [12], size of the software [8], year and hardware [13], generation of programming languages [14], experience of programmers [15], and task complexity [7]. However, few quantitative studies exist (such as [16]) in support of some of the claims.

3. Research questions and methods

Considering the motivation previously stated, we aim at answering the following research questions and sub-questions:

- RQ1: What non-personal factors affect the evolution of overall productivity and productivity per phase of PSP developers during their PSP training projects?
  - RQ1.1: Do process changes affect productivity?
  - RQ1.2: Do other project characteristics affect productivity?
- RQ2: What personal factors may explain productivity variations among individuals for the same projects?
  - RQ2.1: Does personal programming experience affect productivity?

Considering the data available, in this study productivity is measured in LOC/hour, following the PSP literature [1]. We also use the inverse of productivity in minutes/LOC. Regarding RQ1, by analyzing the evolution of the productivity per phase we expect to obtain a better understanding of the influence of process changes, since they occur in specific phases.

In order to answer the research questions, we analyzed SEI's PSP for Engineers I/II course data, including data from 31140 submissions by 3114 students for 10 projects, produced during 295 training classes occurred between 1994 and 2005.

We started by selecting the relevant tables and columns for the analysis. For each submission, we selected the following data: actual effort, actual size, estimated effort, estimated size, actual effort per phase, student number, and projects number (1 to 10). For each student, was also selected the following information: years of programming experience and size of code previously developed using the course programming language. Additional information was occasionally inspected.

The next step was to clean the data. We excluded all submissions with 0 minutes for any phase (except for optional or non-applicable phases), or with a significant discrepancy (>2 min) between the actual effort and the summation of the actual effort per phase. In the end, we had 26140 records (submissions) selected.

Subsequently, we analyzed the selected data to answer the research questions and determine the non-personal factors, as described in section 5, and the personal factors, as described in section 6. Before presenting the analysis results, we review the PSP training context in section 4.

4. PSP Training Context

Table 1 describes briefly the programming projects that are part of the PSP for Engineers I/II course and the PSP version (level) used in each project. Tables 2 and 3 describe briefly the contents of each PSP level and the process changes introduced between consecutive PSP levels.

Table 1
Sequence of programming projects and PSP levels throughout the PSP training course

Project number	Project description	PSP level
1	Mean and standard deviation calculation	PSP0
2	Size counting for a program	PSP0.1
3	Size counting for a program and its parts	PSP0.1
4	Linear regression parameters	PSP1
5	Simpson's rule integration with normal distribution	PSP1.1
6	Prediction intervals with linear regression and t-distribution.	PSP1.1
7	Correlation and significance	PSP2
8	Sort list of pairs	PSP2.1
9	Degree to which data fits a normal distribution	PSP2.1
10	Multiple linear regression	PSP2.1

Table 2
PSP levels and changes between levels: PSP0 to PSP1

PSP level	Description
PSP0	Baseline process for developing individual programs or components of larger systems, with the following phases: • PLAN: planning, including empirical effort estimates; • DLD: detailed design; • CODE: coding in the programming language chosen by each developer; • COMP: compile and corresponding bug fixing; • UT: (unit) testing and corresponding bug fixing; • PM: post-mortem, including the review of all gathered data. Defect and time (effort) logging are performed along all phases. It is expected that COMP and UT phases progressively benefit (in terms of time spent) from defect logging, because developers will become increasingly aware of their mistakes and will tend to avoid them in previous DLD and CODE phases.
PSP0.1	The following process phases are changed: • PLAN: empirical size estimation is introduced; • CODE: coding standards are introduced, but no significant impacts are expected regarding the amount of work; • PM: program size measurement (with partial tool support) and process improvement proposals (PIPs) are introduced.
PSP1	The following process phases are changed: • PLAN: empirical estimates are replaced by the PROBE estimation method, involving the identification (in a so-called conceptual design) and T-shirt sizing of parts to develop, and the derivation of size and effort estimates using historical data (with tool support); • UT: a simple test report is added, documenting test cases and results, without requiring significant additional work; • PM: counting and measurement of program parts (methods, functions, etc.) is introduced (with partial tool support). The following process phase is expected to benefit (in terms of effort required) from changes introduced in other phases: • DLD: is expected to benefit from the construction of a conceptual design in the PLAN phase.

Table 3
PSP levels and changes between levels: PSP1.1 to PSP2.1

PSP level	Description
PSP1.1	The following process phase is changed: • PLAN: task and schedule planning are introduced; however, they are only of interest for multi-day projects, which is not usually the case during the training.
PSP2	The following process phases are added, immediately after the DLD and CODE phases, respectively: • DLDR: design review (guided by checklists derived from personal historical defect data) and corresponding bug fixing; • CR: code review (guided by checklists derived from personal historical defect data) and corresponding bug fixing. The following process phases are changed: • PLAN: quality planning is introduced, namely, estimates of review times and of defects removed in reviews; because DLDR and CR are introduced in PSP2, the first time PSP2 is used these estimates have to be done manually given the absence of historical data; in subsequent projects with PSP2 or PSP2.1, these calculations are done automatically based on historical data; • PM: process yield measurement is introduced (% of defects removed before compile). The following process phases are expected to benefit (in terms of effort required) from changes introduced in other phases: • COMP, UT: may benefit from CR (PSP developers are asked to perform code reviews before compile and test).
PSP2.1	The following process phases are changed: • PLAN: prediction intervals are introduced; they are calculated automatically, so no additional work is required from the user; • DLD: several design specification templates are introduced (operational, logic, functional and state specification templates); • DLDR: design verification techniques are introduced, related with some of the design specification templates. The following process phase is expected to benefit, in terms of effort required, from changes introduced in other phases: • CODE: is expected to benefit from the introduction of design specification templates in the DLD phase.

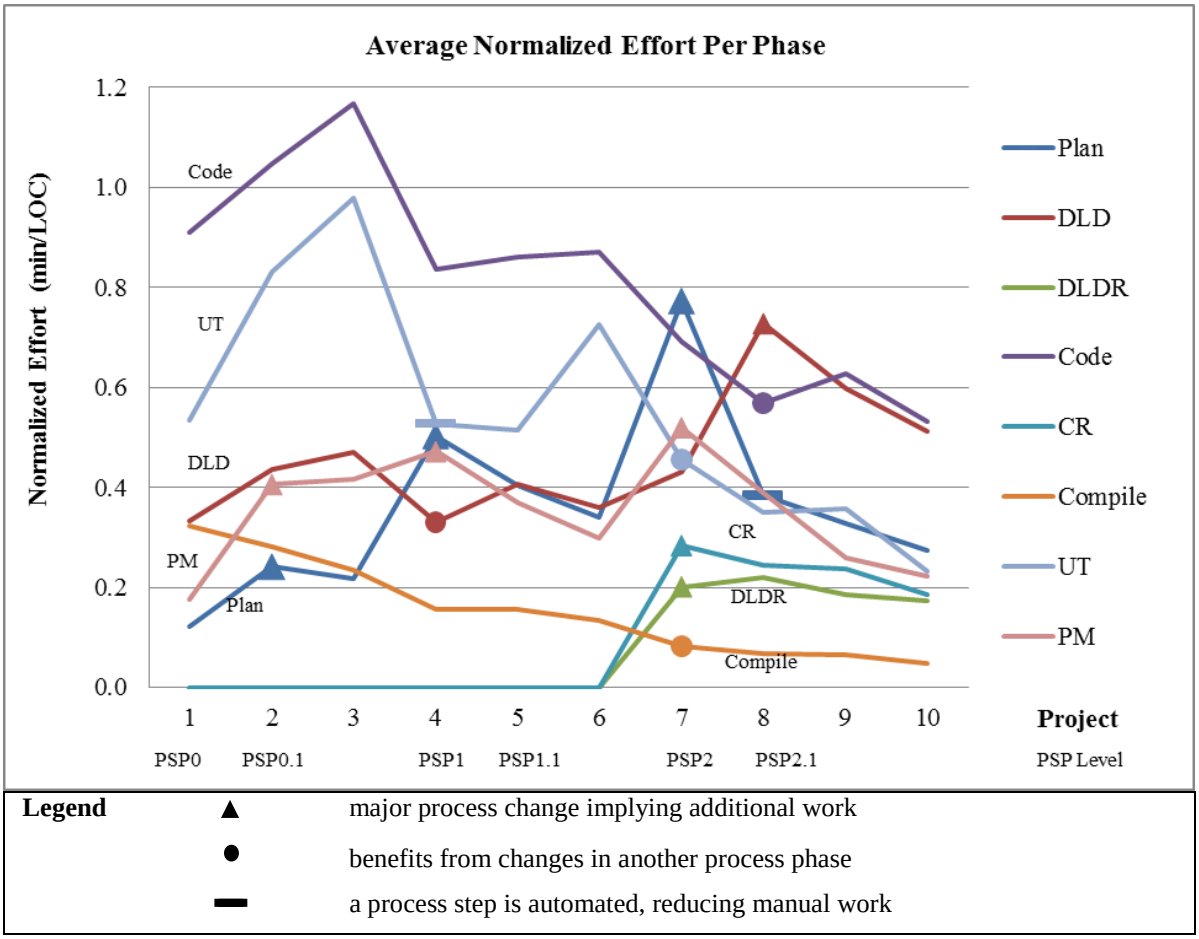

Figure 1. Evolution of the average normalized effort per phase throughout the projects.

5. Analysis of Non-Personal Factors

In order to visualize the impact of process changes (RQ1.1) on the evolution of productivity, we computed the chart in Figure 1 from the data set. We computed the productivity per phase, instead of the overall productivity, to obtain a better insight of the influence of process changes, since they occur in specific phases. To facilitate summations and comparisons, we measured the inverse of the productivity, i.e., the normalized effort in min/LOC. In order to exclude personal factors, we computed the average for all students. For an easier visual analysis, we marked significant process changes (based on the information from Tables 1, 2, and 3) with special symbols.

A simple visual inspection of Figure 1 shows that most of the biggest changes of productivity per phase are associated with process changes, with the notable exception of the CODE and UT phases. An ANOVA test [17] confirmed that productivity changes between projects are statistically significant in all phases, with p-values round about 2e-16. To confirm where exactly the significant differences in mean productivity lay throughout the projects in all phases (comparisons of mean of consecutive projects), a Tukey´s HSD test [17] in conjunction with ANOVA was performed and detailed p-values are presented in Table 4.

Table 4
Analysis of variation of average normalized effort per phase, showing p-value, and actual and expected variation

Proj.	PLAN	DLD	DLDR	CODE	CR	COMP	UT	PM
1-2	.000↑↑	.000↑→		.000↑→		.001↓→	.000↑→	.000↑↑
2-3	.810→↘	.380→→		.000↑→		.000↓→	.000↑→	1.00→↘
3-4	.000↑↑	.000↓↓		.000↓→		.000↓→	000↓→	.003↑↑
4-5	.000↓↘	.000↑→		.968→→		1.00→→	1.00→→	.000↓↘
5-6	.000↓↘	.097→→		1.00→→		.559→→	.000↑→	.000↓↘
6-7	.000↑↑	.001↑→	.000↑↑	.000↓→	.000↑↑	.000↓↓	.000↓↓	.000↑↑
7-8	.000↓↓	.000↑↑	.001↑↑	.000↓↓	.000↓↘	.938→→	.002↓→	.000↓↘
8-9	.007→→	.000↓↘	.000↓↘	.187→→	.319→↘	1.00→→	1.00→→	.000↓↘
9-10	.019→→	.000↓↘	.035→↘	.001↓→	.000↓→	.932→→	.000↓→	.451→→

In Table 4, p-values less than 0.05 indicate a statistically significant difference of average normalized effort between the projects indicated in the first column, for

the phase indicated in the first row. Each cell contains the p-value followed by two arrows indicating the direction of the actual and expected variation of normalized effort per phase, respectively: ↑ - significant increase; ↓ - significant decrease; → - no significant change; ↘ - significant decrease or no significant change. Cells where the actual productivity evolves as expected are painted in green; the others are painted in red.

Regarding the expected variations of normalized effort per phase between consecutive projects, caused by process changes alone, we distinguished the following cases:

- ↑ - cases where there is a process change that is expected to imply significant additional work (i.e., a significant increase of normalized effort per phase);
- ↓ - cases where a process phase benefits from changes in other phase or a process step is automated, leading in both cases to less manual work in a process phase;
- ↘ - cases where there is no process change, but a process change of the first type was introduced recently (say, in the previous or before previous project); in these cases, one may expect a reduction of normalized effort because of learning, or at least no significant change;
- → - no process change was introduced in this or in previous two projects, so no significant change of productivity is expected.

We conclude that in 44 out of 62 cases (71%), the productivity evolves according to the expectations (green cells in Table 4), and in 18 out of 62 cases (29%), the productivity evolves differently from expected (red cells in Table 4), suggesting that there are other factors (besides process changes) which are affecting productivity behavior. All the discrepancies correspond to cases where the process is stable but nevertheless significant variations of productivity are observed, being the UT phase the most problematic one. This is not surprising, because the UT phase is often considered the less predictable one [1]. By looking at the projects descriptions in Table 1, a possible explanation for the higher normalized effort in the UT phase in projects 2 and 3 could be related with the application domain – text processing – different (and possibly more error prone) from all other projects (numerical problems). However, such explanation requires further investigation.

Figure 1 also allows us to observe the magnitude of productivity changes that occur for each process change. The most noticeable impacts occur with the changes in the PLAN phase in project 7 (introduction of quality planning) and in the DLD phase in project 8 (introduction of design templates). In both cases, the time spent in the phase affected exceeds the time spent in the Code phase.

Regarding the overall impact of the training, Figure 1 also shows that there is an increase of Design time and a decrease of Code time throughout the training, with similar values by the end of the training. There is also a significant reduction of Compile and Test time and a closer balance between appraisal (reviews) and failure (bug fixing) efforts by the end of the training.

6. Analysis of Personal Factors

In this section we aim at identifying, based on the available data, possible personal factors that explain productivity variations among individuals for the same projects (RQ2). Firstly, we'll check whether there are groups of individuals that consistently perform better than others.

6.1 Productivity variations among individuals

Figure 2 shows the mean productivity of each group of PSP training students (G1 to G5), for the 10 projects. The groups were defined in order to stratify the students into groups of equal size according to their mean productivity throughout the 10 projects. For example, G1 contains the students with the 20% lowest values of mean productivity during the 10 projects. Figure 2 shows that: (1) there are significant differences in productivity among individuals; (2) individuals have a consistent productivity during the 10 projects (i.e., groups keep their relative position throughout the 10 projects).

The extremely small p-value (<2e-16) obtained in the ANOVA analysis (see Figure 2) confirms the hypothesis that the differences of mean productivity among the groups are statistically significant (significance threshold < 0.1%).

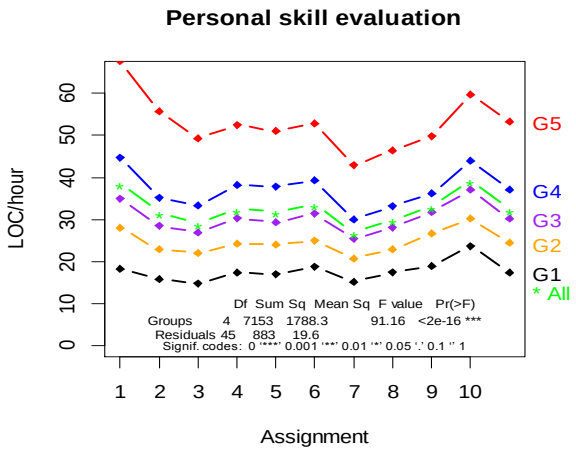

Figure 2. Differences of mean productivity for different groups of individuals throughout the 10 projects. The last column refers to the average for all projects.

6.2 Impact of programming experience on productivity

In order to find an explanation for the differences among individuals, we analyzed existing data characterizing the individuals that attended the courses, namely their experience in terms of amount of code developed and years of programming experience, obtaining the charts shown in Figure 3. The labels in the horizontal axis show the classes considered for each characteristic, and the numbers immediately above indicate the number of individuals in each class. The vertical axis shows the average productivity, in LOC/hour, of the students in each class.

The results obtained show that both characteristics analyzed influence the productivity during the course, with

best values for 6-8 years of programming experience, and 20-100 KLOC previously developed in the programming language used in the course. However, these results should be taken with some caution, because the information analyzed is available only for a fraction of the students.

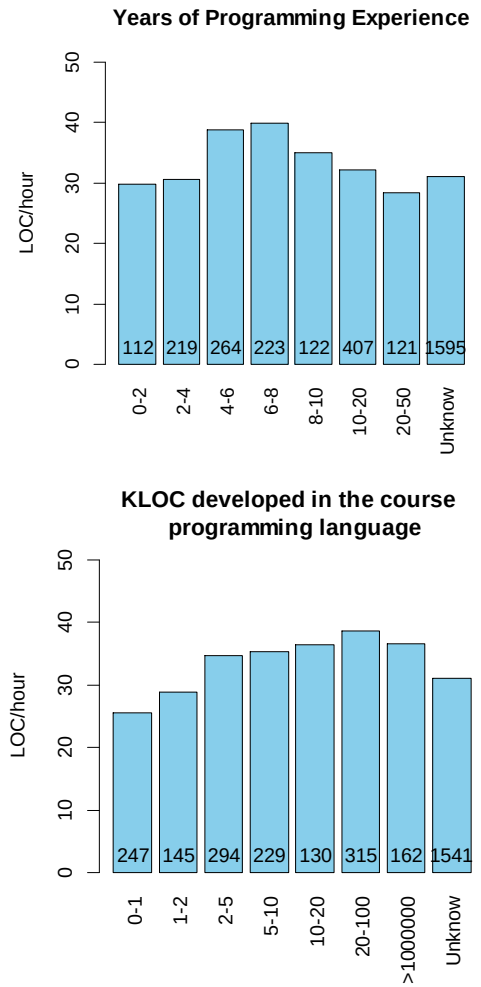

Figure 3. Charts showing the impact of programming experience on productivity.

For each characteristic analyzed (Years of Programming Experience and KLOC developed in the Course Programming Language), an ANOVA test confirmed (with p-values of 1.61e-08 and 2e-16, respectively) that the differences in characteristic are statistically significant, i.e., there is significant difference in productivity among classes, shown in Figure 3. A Tukey´s HSD test [17] in conjunction with ANOVA is performed to compare the mean productivity of all classes with each other for each characteristic and to find where exactly are the significant differences. The p-values representing these differences are shown in Table 5. Cells in green represent those classes which mean productivities are significantly different.

Table 5
Analysis of variance between all pairs of classes of personal skills

Years of programming experience		KLOC developed in the course programming language	
Classes	p-value	Classes	p-value
(2-4)-(0-2)	0.8880	(1-2)-(0-1)	0.633
(4-6)-(0-2)	0.007	(2-5)-(0-1)	0.000
(6-8)-(0-2)	0.0008	(5-10)-(0-1)	0.000
(8-10)-(0-2)	0.2668	(10-20)-(0-1)	0.000
(10-20)-(0-2)	0.9782	(20-100)-(0-1)	0.000
(20-50)-(0-2)	0.9950	(100-1000)-(0-1)	0.000
(4-6)-(2-4)	0.0732	(2-5)-(1-2)	0.000
(6-8)-(2-4)	0.008	(5-10)-(1-2)	0.000
(8-10)-(2-4)	0.8220	(10-20)-(1-2)	0.003
(10-20)-(2-4)	0.9970	(20-100)-(1-2)	0.000
(20-50)-(2-4)	0.4158	(100-1000)-(1-2)	0.006
(6-8)-(4-6)	0.9816	(5-10)-(2-5)	0.829
(8-10)-(4-6)	0.9650	(10-20)-(2-5)	0.999
(10-20)-(4-6)	0.0020	(20-100)-(2-5)	0.109
(20-50)-(4-6)	0.0002	(100-1000)-(2-5)	0.999
(8-10)-(6-8)	0.6882	(10-20)-(5-10)	0.986
(10-20)-(6-8)	0.0000	(20-100)-(5-10)	0.922
(20-50)-(6-8)	0.0000	(100-1000)-(5-10)	0.853
(10-20)-(8-10)	0.4342	(20-100)-(10-20)	0.578
(20-50)-(8-10)	0.0484	(100-1000)-(10-20)	0.999
(20-50)-(10-20)	0.6105	(100-1000)-(20-100)	0.207

7. Conclusions and Future Work

Regarding non-personal factors affecting productivity, by looking into the evolution of the productivity per phase of PSP students during the PSP training, the study shows that most of the productivity variations throughout the 10 projects are associated with process changes and could be explained by those process changes. However, in a significant number of cases, we observed productivity variations in spite of a stable process.

We found that a possible explanation for some of the variations found might be attributed to a higher complexity of the application domain in some projects, but this hypothesis requires further investigation. This finding is particularly important for PSP users, because it suggests that, contrarily to what is usually assumed in PSP estimation methods, historical productivity (in LOC/hour) may be a weak estimator for productivity in subsequent projects, even in the presence of a stable development process.

Regarding personal factors, we found that the programming experience (years and amount of code developed) has a significant impact on productivity. A somewhat surprising (but understandable) result was the tendency for productivity to decrease after a certain level of years of programming experience.

As future work, we intend to build a quantitative process performance model to help identifying and ranking root causes of productivity problems (by using the model in the backward direction), and predicting the impact of improvement actions (by using the model in the forward

direction). A similar analysis for other performance indicators will be conducted based on the PSP training data.

Acknowledgements

The authors would like to acknowledge the Carnegie Mellon Software Engineering Institute for giving access to the PSP training data for performing this study.

This work is partially funded by the Portuguese Foundation for Science and Technology (FCT - Fundação para a Ciência e a Tecnologia), under research grant SFRH/BD/85174/2012.

References

[1] W. Humphrey, *PSP^{sm}: A Self-Improvement Process for Software Engineers*, Addison-Wesley Professional, 2005.

[2] C. Duarte, *Automated Software Processes Performance Analysis and Improvement Recommendation*, MSc Thesis, Faculty of Engineering of the University of Porto, 2012.

[3] C.B.Duarte, J. P. Faria and M. Raza, PSP PAIR: Automated Personal Software Process Performance Analysis and Improvement Recommendation, *Proceedings of the 8th International Conference on the Quality of Information and Communications Technology (QUATIC 2012)*, 131-136.

[4] C. B. Duarte, J. P. Faria, M. Raza and P. C. Henriques, Model and Tool for analyzing Time Estimation Performance in PSP, *TSP Symposium 2012 Proceedings*, CMU/SEI-2012-SR-015, 2012, 21-40.

[5] S. Wagner, & M. Ruhe, A Systematic Review of Productivity Factors in Software Development, *Proceedings of 2nd International Workshop on Software Productivity Analysis and Cost Estimation (SPACE 2008)*, State Key Laboratory of Computer Science, Institute of Software, 2008.

[6] K. D. Maxwell, & P. Forselius, Benchmarking Software Development Productivity, *IEEE Software, 17*(2), January/February 2000, 80-88.

[7] P. Goparaju, A. Farooq, and S. Patnaikc, Measuring Productivity of Software Development Teams, *Serbian Journal of Management* 7 (1) (2012), 65-75.

[8] D. N. Card, The Challenge of Productivity Measurement, *Proceedings of the Pacific Northwest Software Quality Conference*, Portland, OR, 2006.

[9] W. Hayes and J. W. Over, *The Personal Software Process^{SM} (PSP^{SM}): An Empirical Study of the Impact of PSP on Individual Engineers*, CMU/SEI-97-TR-001, Software Engineering Institute, 2007.

[10] D. Rombach, J. Münch, A. Ocampo, W. S. HumphreY and D. Burton, Teaching disciplined software development, *Journal of Systems and Software* 81(5): 2008, 747–763.

[11] S. Wen-Hsiang, H. Nien-Lin, and L. Wei-Mann, Assessing PSP effect in training disciplined software development: A Plan–Track–Review model, *Information and Software Technology*, Volume 53, Issue 2, February 2011, 137–148.

[12] W. Scacchi, Understanding Software Productivity. *Software Engineering and Knowledge Engineering: Trends for the Next Decade*, W.Hurley (ed.), World Scientific Press, 1995.

[13] R. Premraj, , B. Kitchenham, M. Shepperd, & P. Forselius, An Empirical Analysis of Software Productivity over Time, *11th IEEE International Symposium on Software Metrics*, 2005.

[14] C. Comstock, Z. Jiang, & P. Naudé, Strategic Software Development: Productivity Comparisons of General Development Programs, International Journal of Computer and Information Engineering 1:8 2007, 486-491.

[15] R. D. Banker, & R. J. Kauffman, Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study, *MIS Quarterly*, September 1991, 14(3):374-401.

[16] R. Lagerström, L. von Würtemberg, H. Holm, O. Luczak, Identifying factors affecting software development cost and productivity, *Software Quality Journal*, Volume 20, Issue 2, June 2012, 395-417.

[17] W. Navidi, *Statistics for Engineers and Scientists*, *Third Edition*, McGraw-Hill, 2011.