ProcessPAIR: A Tool for Automated Performance Analysis and Improvement Recommendation in Software Development

Mushtaq Raza
INESC TEC/University of Porto - Faculty of Engineering
Rua Dr. Roberto Frias, s/n
4200-465 Porto PORTUGAL
+351 225081400
mushtaq.raza@fe.up.pt

João Pascoal Faria
INESC TEC/University of Porto - Faculty of Engineering
Rua Dr. Roberto Frias, s/n
4200-465 Porto PORTUGAL
+351 225081400
jpf@fe.up.pt

ABSTRACT
High-maturity software development processes can generate significant amounts of data that can be periodically analyzed to identify performance problems, determine their root causes and devise improvement actions. However, conducting that analysis manually is challenging because of the potentially large amount of data to analyze and the effort and expertise required. In this paper, we present ProcessPAIR, a novel tool designed to help developers analyze their performance data with less effort, by automatically identifying and ranking performance problems and potential root causes, so that subsequent manual analysis for the identification of deeper causes and improvement actions can be properly focused. The analysis is based on performance models defined manually by process experts and calibrated automatically from the performance data of many developers. We also show how ProcessPAIR was successfully applied for the Personal Software Process (PSP). A video about ProcessPAIR is available in https://youtu.be/dEk3fhhkduo.

CCS Concepts
• Software and its engineering~Software development process management

Keywords
software process; performance analysis; improvement recommendation

1. INTRODUCTION
Software development processes, making intensive use of metrics and quantitative methods, such as the Team Software Process (TSP) [1] and Personal Software Process (PSP) [2], can generate large amounts of data that can be periodically analyzed by developers to identify their performance problems, determine root causes and devise improvement actions [3]. Although tools exist to automate data collection and produce performance charts and reports for manual analysis of TSP/PSP data [4][5][6], practically no tool support exists to automate developer performance analysis. The manual analysis of performance data for determining root causes of performance problems and devising improvement actions is challenging because of the amount of data to analyze [3] and the effort and expertise required.

Conference'10, Month 1–2, 2010, City, State, Country.

DOI: http://dx.doi.org/10.1145/12345.67890

ProcessPAIR, is a novel tool designed to help developers analyze their performance data with less effort, by automatically identifying and ranking performance problems and potential root causes, so that subsequent manual analysis for the identification of deeper causes and improvement actions can be properly focused. The analysis is based on a performance model (PM) defined by experts in the process under consideration, and calibrated automatically from the data of many process users. In previous work [7], we developed the overall technique, PMs specific for the PSP, and a prototype tool. In this tool demonstration paper, we present a significantly improved version of ProcessPAIR, available freely in http://blogs.fe.up.pt/processpair/, together with several tutorials and videos. A video tutorial is available in http://blogs.fe.up.pt/processpair/tutorials/videos/.

The paper is organized as follows. Section 2 presents the overall approach, tool architecture, and underlying metamodels. Sections 3 and 4 explain the model calibration and performance analysis processes and user interfaces. Section 5 presents some experimental results. Some related work is presented in Section 6. Section 7 concludes the paper.

2. APPROACH, ARCHITECTURE, AND METAMODELS

2.1 Overall Approach and Architecture
Our approach involves three main steps (see Figure 1):

1. *Define*: Process experts define the structure of a PM suited for the development process under consideration. In our approach, a PM comprises a set of performance indicators (PIs) organized hierarchically by cause-effect relationships [7].

2. *Calibrate*: The PM is automatically calibrated by ProcessPAIR based on the performance data of many process users. The statistical distribution of each PI and statistical relations between PIs are computed from the data set [7].

3. *Analyze*: Once a PM is defined and calibrated, the performance data of individual developers can be automatically analyzed with ProcessPAIR, to identify and rank performance problems and root causes.

ProcessPAIR currently comprises a core framework (representing 75% of the code base), independent of the process under analysis, and an extension for the PSP (representing 25% of the code base),

as depicted in Figure 2. Other extensions may be easily implemented in the future for other processes. The core framework comprises three layers: a graphical user interface layer at the top (`gui` package); an intermediate logic layer responsible for the representation and manipulation of PMs (`performancemodel`) and subject data under analysis (`subjectdata`); a layer with common utilities at the bottom (`statistics`). The PSP extension (`pspextension`) contains the definition of PMs for the PSP and subject data loaders from the most relevant project management tools used by PSP Developers – the SEI's PSP Student Workbook and Process Dashboard (http://www.processdash.com/).

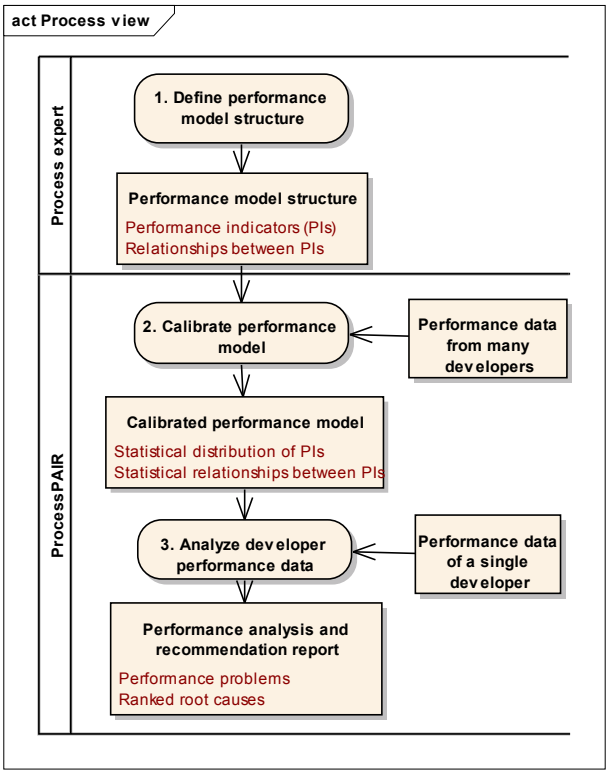

Figure 1. UML activity diagram depicting the main activities and artifacts in the ProcessPAIR approach.

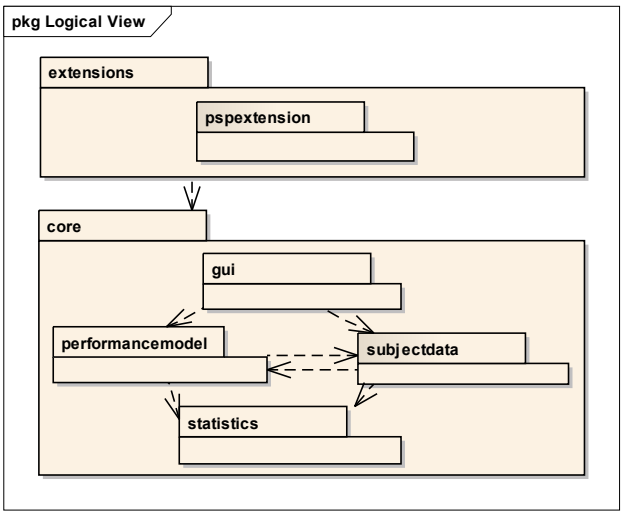

Figure 2. UML package diagram depicting the logical architecture of the ProcessPAIR tool.

2.2 Performance Model Metamodel

A PM for a development process under consideration is defined by means of the following information (see Figure 3):

- Set of relevant base measures generated by the development process, with name, description, scale, and units;
- Set of relevant PIs, with the same attributes as the base measures, plus the formula for its computation from base measures and the optimal value (usually implied by the definition of the PI);
- Subset of top-level PIs;
- Dependencies between PIs, representing cause-effect relationships, determined by a formula or statistical evidence;
- Sensitivity coefficients [8] $\sigma_{X_i \to Y} = \frac{\partial Y}{\partial X_i}\left(\frac{X_i}{Y}\right)$, $i=1, \ldots, n$, for each PI Y that depends on PIs $X_1, \ldots, X_n$, according to a formula $Y=f(X_1, \ldots, X_n)$.

For example, our PM for the PSP [7] comprises three top-level indicators: *Time Estimation Accuracy, Process Quality Index* and *Productivity*. The *Time Estimation Accuracy* is computed from base measures as a ratio *ActualTime*/*EstimatedTime*, being 1 the optimal value. Since in the PSP's PROBE estimation method [2], a time estimate is obtained based on a size estimate of the deliverable (in added or modified size units) and a productivity estimate (in size per time units), we consider that the *Time Estimation Accuracy (TimeEA)* is affected by (or depends on) the *Size Estimation Accuracy (SizeEA)* and the *Productivity Estimation Accuracy (ProdEA)*. From their definitions [7], we conclude that these PIs are related by the formula $TimeEA = SizeEA/ProdEA$, so the sensitivity coefficients are $\sigma_{SizeEA \to TimeEA} = 1$ and $\sigma_{ProdEA \to TimeEA} = -1$. Sensitivity coefficients are used for ranking the causes of performance problems.

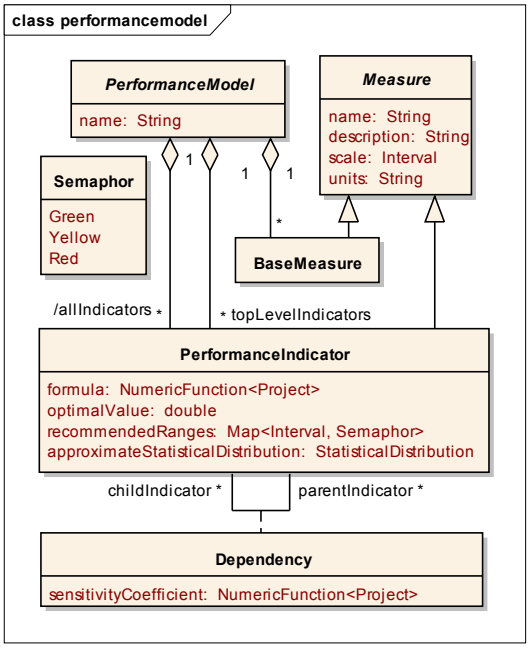

Figure 3. UML class diagram depicting the main concepts in the `performancemodel` package.

The PM is automatically calibrated by ProcessPAIR from training data sets, generating the following data (also visible in Figure 3):

- approximate statistical distribution of each PI, represented by a cumulative distribution function;

- recommended performance ranges for each PI;
- sensitivity coefficients between PIs not related by an exact formula.

The approximate cumulative distribution function of each PI is computed by linear interpolation between a few percentiles computed from the training data.

Performance ranges are needed for classifying values of each PI of a subject under analysis into three categories (semaphores): green - no performance problem; yellow - a possible performance problem; red - a clear performance problem. Such ranges are determined automatically from the statistical distribution of the training data so that there is an even distribution of data points by the colors.

Sensitivity coefficients between PIs not related by an exact formula are computed by first determining a linear regression equation from the training data and subsequently computing the corresponding sensitivity coefficient.

2.3 Subject Data Metamodel

The base performance data of a subject under analysis (developer, team or company) that need to be uploaded by ProcessPAIR, consists in the values of the base measures defined in the PM for a sequence of projects (see `Subject`, `Project`, and `ProjectBaseMeasure` in Figure 4).

Based on that information, ProcessPAIR computes the following data for each PI and project (see `ProjectIndicator` and `IndicatorInstance` in Figure 4):

- value – computed from the base measures and PI's formula;
- percentile – computed from the previous value and the statistical distribution of the PI in the PM, normalized so that 100% corresponds to the optimal value and 0% corresponds to extreme values to the left or to the right of the optimal value;
- semaphore – computed from the previous percentile as follows: green for the 66.7%-100% range, yellow for the 33.3%-66.7% range, and red for the 0%-33.3% range;
- percentile coefficient – computed from the percentile and the statistical distribution of the PI, as explained in [7]; it is used as an indicator of the 'cost' (or difficulty) of improving the value of the PI, based on the idea that the closer a value is to the optimal, the more difficult it is to improve.

Summary information for each PI is computed at the subject level (see `SubjectIndicator` and `IndicatorInstance` in Figure 4):

- minimum, maximum, average – simple statistics calculated from the values computed at the project level;
- percentile – weighted average of the percentiles computed at the project level, using an exponentially decaying weight for older projects with a configurable time constant;
- semaphore and percentile coefficient – computed from the previous percentile.

For each dependency defined in the PM and project, it is computed the following information (see `DependencyInstance` in Figure 4):

- sensitivity coefficient – computed from the project data and the sensitivity formula defined in the PM;
- ranking coefficient - computed as the product of the previous sensitivity coefficient and the percentile coefficient of the child PI; it is used to rank child PIs based on a cost-benefit estimate of improvement actions (with the cost factor given by the percentile coefficient and the benefit factor given by the sensitivity coefficient) [7];
- ranking label – a discretization of the ranking coefficient, by orders of magnitude, for user presentation purposes.

Similar information is also computed at the subject level (by summarization) and between leaf and top-level indicators (by using the laws of partial differentiation of composite functions for computing leaf-to-top sensitivity coefficients).

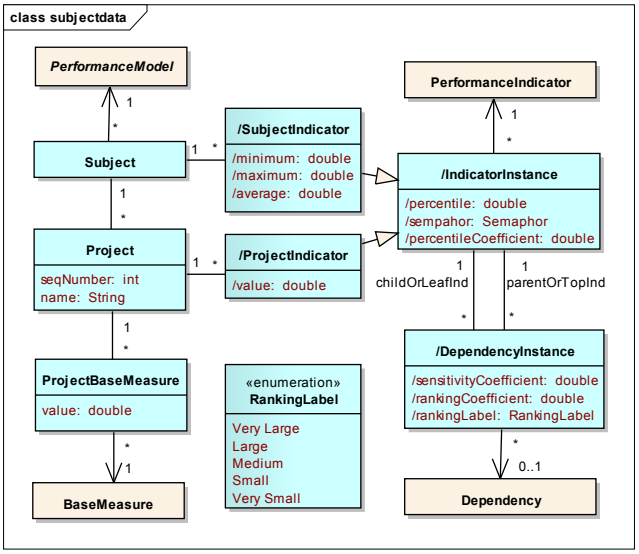

Figure 4. UML class diagram depicting the main concepts in the `subjectdata` package.

3. MODEL CALIBRATION

The user interface for performing the automatic calibration is shown in Figure 5. The user has to select the PM to be calibrated (from the list of PMs previously defined as tool extensions), the file with the data set to be used for calibration (in a format supported by the data loaders defined together with the PM) and the XML file for saving the calibration results.

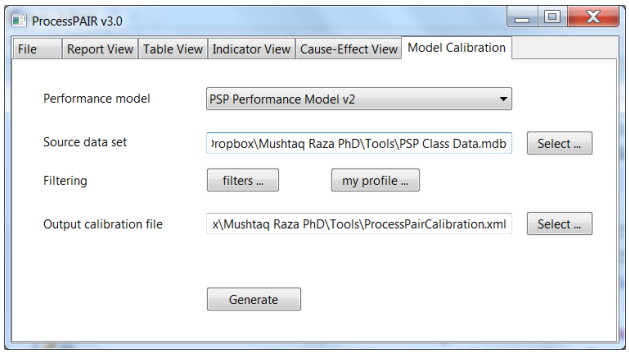

Figure 5. Model calibration window.

In this example, to calibrate the PSP PM, we used a large PSP data set from the Software Engineering Institute (SEI) referring to 31,140 projects concluded by 3,114 engineers during 295 classes of the classic PSP for Engineers I/II training courses running between 1994 and 2005. In this training course, targeting professional developers, each engineer develops 10 small projects.

ProcessPAIR performs several data quality checks during the calibration process (according to rules defined together with the

PM) and presents a summary of problems encountered at the end of the calibration process, as illustrated in Figure 6.

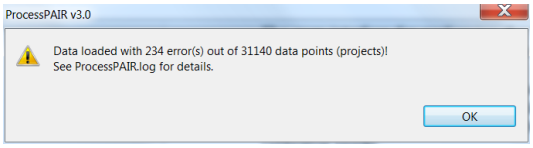

Figure 6. Summary of calibration results.

Instead of using the full dataset for calibration, it is also possible to filter the data points to be used for calibration. One possibility is to restrict the data points (projects) to the ones most similar to a given user profile, as illustrated in Figure 7. The parameters that can be provided depend on the PM and data loader. Similarity is computed with the Gower similarity coefficient [9]. In this example (see Figure 8), only the 50 most similar data points were selected (minimum number required by the tool for statistical significance), with a similarity coefficient greater than 0.889.

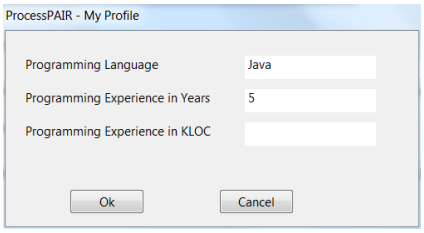

Figure 7. Dialog for providing a user profile.

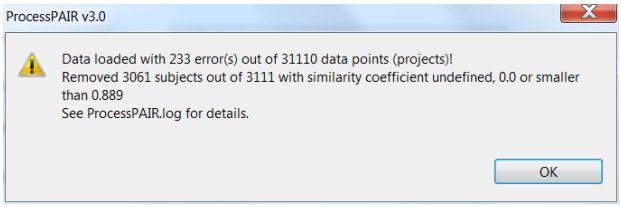

Figure 8. Calibration results with filtering.

4. PERFORMANCE ANALYSIS

Having defined and calibrated the PM, the performance data of individual developers can be automatically analyzed by ProcessPAIR, to identify and rank performance problems and potential causes. As exemplified in Figure 9, the user has to select the PM (from the list of PMs previously defined as tool extensions), the calibration file (generated as previously explained), the type of input file with performance data to analyze (according to the data loaders defined together with the PM), and the file with the actual data. By pressing the "Analyze file" button, the analysis is performed and the results are presented in multiple views.

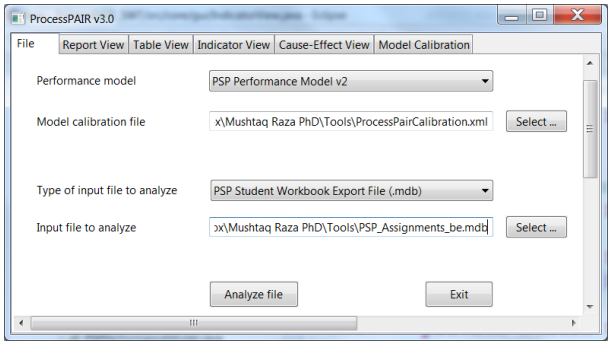

Figure 9. Entry window.

4.1 Table View

The Table view (Figure 10) shows the values of the PIs defined in the model for the projects described in the input file, as well as summarized performance information. Each cell is colored green, yellow or red, in case its value suggests no performance problem, a potential performance problem, or a clear performance problem, respectively (see calculations in Section 2). This way, the Table View helps in quickly identifying the performance problems. The exact ranges considered can be consulted in the "Indicator View". The "Percentile (all)" column shows an overall percentile for each PI, computed from the per project values (with higher importance for the last projects), and colored according to the percentile.

The PIs are organized hierarchically, starting from the top-level indicators (*Time Estimation Accuracy*, *Process Quality Index*, and *Productivity* in this case), and descending to lower level indicators (child indicators) that affect the higher level ones according to a formula or statistical evidence [6]. This way, by drilling down from the top-level indicators to the lower level ones, focusing on the red (or yellow) colored cells, one can easily identify potential root causes of performance problems.

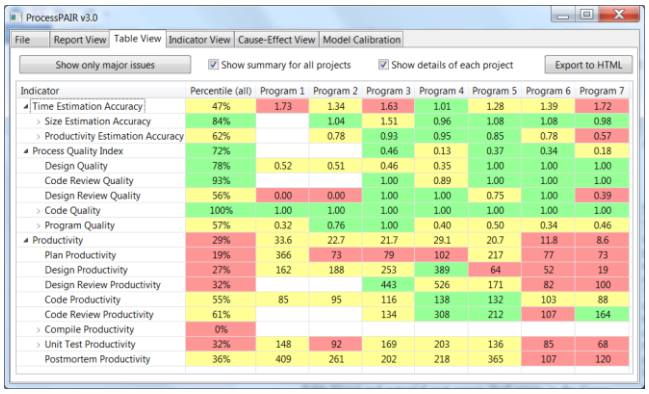

Figure 10. Table view example (partially expanded).

4.2 Report View

The goal of the Report view (Figure 11) is to indicate in a simple way, overall ("Summary") or project by project, the most relevant top-level performance problems (colored red or yellow in the Table View) and potential root causes (leaf causes in the Cause-Effect View) properly prioritized (according to the ranking coefficients explained in Section 2). Intermediate causes can be consulting by unchecking the "Show only leaf causes" checkbox. Comboboxes allow selecting information for specific projects and/or PIs. The links skip to the Indicator View, for detailed information about each PI.

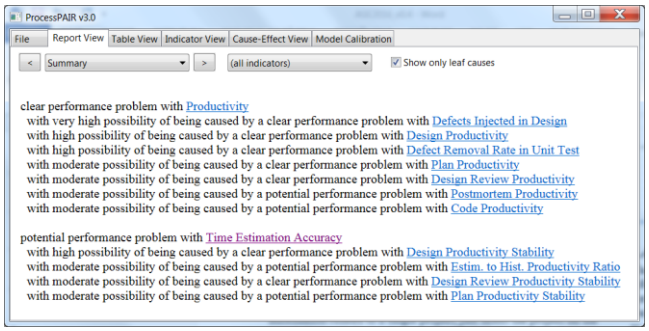

Figure 11. Report view example.

4.3 Indicator View

The goal of the Indicator view (Figure 12) is to show the behavior of each PI along the projects under analysis and provide associated model definition and calibration information (description, units, optimal value, recommended performance ranges and statistical distribution).

In the bottom left, it is presented the statistical distribution of the PI in the data set used for calibrating the model. The colors correspond to the performance ranges. The actual values in the file under analysis are also shown, marked with the "+" symbol, for benchmarking purposes.

The user may also select multiple PIs for comparative visualization in a single chart.

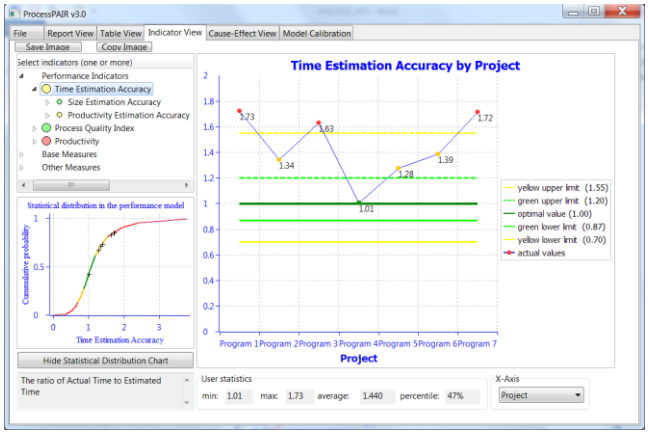

Figure 12. Indicator view example.

4.4 Cause-effect View

The Cause-Effect view (Figure 13) is an advanced view that provides essentially the same information as the report view with additional details but in a diagrammatic way.

The goal of the Cause-Effect View it to help identifying and prioritizing, project by project or overall, the root causes of performance problems, so that subsequent improvement actions can be properly directed. The child indicators are sorted according to the value of the ranking coefficient.

As explained in Section 2, the ranking coefficient represents a cost-benefit estimate that relates the cost of improving the value of the child PI with the benefit on the value of the parent PI.

Intermediate causes may be consulted by unselecting the "Show only leaf causes" checkbox. By default, the ranking coefficients are shown by means of T-shirt sizes (ranking labels). The numerical values of the ranking coefficients can be consulted by selecting "Numerical Ranking Labels" in a combo box.

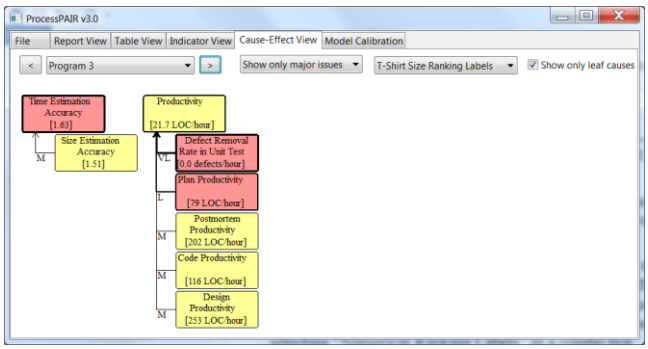

Figure 13. Cause-effect view example.

5. EXPERIMENTAL RESULTS

Two experiments have been conducted to evaluate ProcessPAIR.

5.1 Postmortem Experiment

The goal of the first experiment was to assess the accuracy of automatic performance problem and root cause identification with ProcessPAIR.

To that end, we used as input the PSP performance data and final reports of 10 master students from Tec de Monterrey in Mexico that attended the "Software Quality and Testing" course in 2015. In that course, each student developed 6 projects using the PSP and collected base measures with Process Dashboard (http://www.processdash.com/). In the end of the sequence of projects, the students analyzed their personal performance in those projects and documented their findings and improvement proposals in a "PSP Final Report".

We compared the performance problems and root causes identified and documented by the students in their final reports, with the performance problems and root causes identified automatically by ProcessPAIR from the students' performance data.

Regarding problem identification, from the 187 cases in which students explicitly characterized their performance (regarding a specific PI and a specific project or all projects), we compared the student assessment with the tool-based assessment, and got the following results:

- In 96% of the cases, the results of manual and automatic analysis matched (i.e., both the student and the tool indicated good performance or bad performance);
- In 1% of the cases, the tool indicated a clear or potential problem and the manual analysis indicated good performance (false positives);
- In 3% of the cases, the tool indicated no performance problem but the developer explicitly indicated a performance problem (false negatives).

For each performance problem identified both in manual and automatic analysis and with root causes explicitly pointed out by the students (52 cases), we compared the causes identified in manual and automatic analysis, and got the following results:

- In 19% of the cases, the tool and the developer pointed out the same causes (tool benefit: eliminate manual effort);
- In 54% of the cases, the tool accurately pointed out intermediate causes, and the developer pointed out deeper causes (tool benefit: reduce manual effort);
- In 27% of the cases, the causes identified were inconsistent, because of faults in manual analysis (tool benefit: prevent user errors).

These results show that ProcessPAIR has indeed the potential to accurately identify performance problems and causes, and consequently, reduce the user effort and errors in performance analysis.

5.2 Controlled Experiment

The second experiment is an ongoing controlled experiment, involving 61 master students from Tec de Monterrey in Mexico that are attending the "Software Quality and Testing" course edition in 2016. The main goal is to quantify the benefits of using ProcessPAIR in performance analysis, in terms of time spent and quality of the results.

In their final assignment, students were asked to analyze their personal performance along the PSP projects and document their findings and improvement proposals in a "PSP Final Report". To

perform the assignment, students were randomly split into two groups: a control group and an experimental group. The students in the control group did the final assignment in a traditional way, by inspecting their performance data stored in the Process Dashboard tool through the standard PSP forms, charts, and reports. The 30 students in the experimental group used ProcessPAIR for analyzing their performance data.

Upon completion of the assignment, students in both groups responded a questionnaire containing some free text questions plus 14 questions in a five-point scale related with installability, usability, efficiency, usefulness and level of support provided by the tool they used for conducting the performance analysis. The average scores given by the students were as follows:

- Average score given by the 30 students that used ProcessPAIR: 4.78 (in a scale of 1 to 5);
- Average score given by the 31 students that used Process Dashboard: 3.81 (in a scale of 1 to 5).

This shows a very favorable evaluation of ProcessPAIR. The time spent by the students in performing their final assignment and the grades given by their instructor (still being collected) will allow us to assess the benefits of ProcessPAIR as compared to the traditional approach in terms of effort needed and quality of results produced.

6. RELATED WORK

Our approach draws inspiration from existing work on process performance models (PPM) [8][10], benchmark-based approaches for software product evaluation [11], and defect causal analysis (DCA) techniques [12].

In the context of the CMMI process improvement framework, a PPM is a description of the relationship among attributes of a process or sub-process and its outcomes, developed from historical performance data, and calibrated using collected process and product measures [13]. The main difference is that our PM conveys additional elements needed to identify performance problems (in the outcomes) and rank potential root causes (factors): recommended ranges for each PI; approximate statistical distribution of each PI; sensitivity coefficients (derived from exact or regression equations).

In our approach, in order to enable the automated identification of performance problems, after deciding on the relevant PIs, one has to decide on the relevant ranges. Our approach for defining such ranges draws inspiration from the benchmark-based approach developed by researchers of the Software Improvement Group [11][14] to rate the maintainability of software products, with adaptations for process evaluation instead of product evaluation.

The DCA approach [12] is essentially complementary to our approach. The main advantage of our approach is that it has the potential to identify relevant performance problems and causes in a fully automatic way so that subsequent manual activities can be conducted in a more focused and efficient way, to further determine root causes and devise improvement actions.

7. CONCLUSIONS AND FUTURE WORK

We presented a novel tool (ProcessPAIR) for automating the identification and prioritization of performance problems and root causes in software development, and showed its successful application for the PSP.

As future work, we intend to add to ProcessPAIR the capability of recommending detailed improvement actions for the identified causes of performance problems. We also intend to apply ProcessPAIR for other software development processes.

8. ACKNOWLEDGMENTS

The authors would like to acknowledge the SEI and Tec de Monterrey for facilitating the access to the PSP data for performing this research and AWKUM for their partial initial grant. This work is partially financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project «POCI-01-0145-FEDER-006961», and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia as part of project UID/EEA/50014/2013 and research grant SFRH/BD/85174/2012.

9. REFERENCES

[1] Davis, N., and Mullaney, J. 2003. The Team Software Process (TSP) in Practice: A Summary of Recent Results. CMU/SEI-2003-TR-014.

[2] Humphrey, W. 2005. PSP[sm]: A Self-Improvement Process for Software Engineers. Addison-Wesley Professional.

[3] Burton, D. and Humphrey, W. 2006. Mining PSP Data. In *TSP Symposium 2006 Proceedings*.

[4] The Software Process Dashboard Initiative home page. http://www.processdash.com/.

[5] Philip, J., Kou, H., Agustin, J., Christopher, C., Moore, C., Miglani, J., Zhen, S., Doane, W. 2003. Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined. In *ICSE 2003*. Portland, Oregon.

[6] Shin, H., Choi, H., and Baik, J. 2007. Jasmine: A PSP Supporting Tool. In *Proc. of the Int. Conf. on Software Process* (ICSP 2007), LNCS 4470, Springer-Verlag, 73-83.

[7] Raza, M., Faria, J. 2015. A Model for Analyzing Performance Problems and Root Causes in the Personal Software Process. *Journal of Software: Evolution and Process*, John Wiley & Sons

[8] Saltelli, A., Chan, K., Scott, E. M. 2008. Sensitivity Analysis, Wiley.

[9] Gower, J. C. 1971. A General Coefficient of Similarity and Some of Its Properties. *Biometrics*. Vol 27, No. 4 (Dec., 1971), pp. 857-87.

[10] Tamura, S. 2009. Integrating CMMI and TSP/PSP: Using TSP Data to Create Process Performance Models. CMU/SEI-2009-TN-033.

[11] Alves, T., Ypma , C., Visser, J. 2010. Deriving Metric Thresholds from Benchmark Data. In *2010 IEEE International Conference on Software Maintenance (ICSM),* 1-10.

[12] Card, D.N. 2005. Defect Analysis: Basic Techniques for Management and Learning. *Advances in Computers,* vol. 64, 259-295, Elsevier.

[13] Chrissis, M. B., Konrad, M., Shrum, S., 2003. CMMI: Guidelines for Process Integration and Product Improvement, 2nd Edition. Addison-Wesley.

[14] Alves, T. 2012. Benchmark-based Software Product Quality Evaluation. PhD Thesis. U. Minho