A Benchmark-based Approach for Ranking Root Causes of Performance Problems in Software Development

Mushtaq Raza, João Pascoal Faria

INESC TEC and Department of Informatics Engineering,
Faculty of Engineering, University of Porto
Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal
uomian49@yahoo.com, jpf@fe.up.pt

Technical Report TR-PROCPAIR-2014-01 © FEUP 2014

Abstract. The data generated by high-maturity software development processes, supported by modern cloud-based application lifecycle management tools, can be periodically mined for benchmarking purposes, namely to: compare the performance of an individual developer or organization with the community of peers, and hence identify areas of inferior performance for improvement; determine factors that influence performance in the community of peers, and use that information, together with the specifications of derived measures, to drill down the performance problems of an individual developer or organization, and suggest and rank root causes where improvement actions should focus on. In this paper we present an approach for automatically ranking potential root causes of performance problems, based on a cost-benefit estimate. The approach presented was tuned and applied for the Personal Software Process, because of the availability of a homogeneous data set referring to more than 30,000 finished projects, but it can be replicated in other contexts.

Keywords: Ranking, Root causes, Performance problems, Personal Software Process

1 Introduction

Currently, according to [1], the top two software engineering challenges are (1) the increasing emphasis on rapid development and adaptability, and (2) the increasing software criticality and need for assurance. High-maturity software development processes, such as the Team Software Process (TSP) and the accompanying Personal Software Process (PSP), can help individuals and teams improve their performance and produce virtually defect free software on time and budget [2, 3], addressing current software development challenges. One of the pillars of the TSP/PSP is its measurement framework: based on four simple measures - effort, schedule, size and defects - it supports several quantitative methods for project management, quality management and process improvement [4].

High-maturity software development processes, such as the TSP/PSP, supported by modern cloud-based application lifecycle management tools, can generate large amounts of data from a multitude of users that can be periodically analyzed to identify performance problems, determine their root causes and devise improvement actions [5]. Although several tools exist to automate data collection and produce performance charts, tables and reports for manual analysis of TSP/PSP data [6, 7, 8, 9], practically no tool support exists for automating the performance analysis. There are also some studies that show cause-effect relationships among performance indicators [10, 11], but no automated root cause analysis is proposed. The manual analysis of performance data for determining root causes of performance problems and devising improvement actions is problematic because of the lack of benchmarks, the amount of data to analyze, and the expert knowledge required to do the analysis.

To address those shortcomings, we have been developing models and tools to automate the analysis of performance data produced in the context of the TSP/PSP and other high maturity processes, namely, identify performance problems, identify and rank their root causes and recommend improvement actions. In previous work [13, 14, 15] we developed a prototype tool and a performance model, calibrated based on a large PSP data set referring to more than 30,000 finished projects, to enable the automated identification of performance problems and root causes of individual developers. In this paper we propose a novel approach to rank the identified root causes, based on a cost-effect estimate, so that subsequent improvement actions can be focused on the highest-ranked root causes.

The rest of the paper is organized as follows. Section 2 provides background information on our overall performance analysis approach and performance model. Section 3 presents the ranking approach, which builds upon existing sensitivity analysis methods. Section 4 presents a case study to illustrate the application of the approach. Section 5 presents the conclusions and points out future work.

2 Background: Performance Analysis Approach and Model

2.1 Performance Analysis Approach

An overview of the artifacts and steps involved in our approach for automated performance analysis is shown in Fig. 1.

In order to enable the automated identification of performance problems and root causes for individual developers or organizations, a set of performance indicators (PIs), recommended performance ranges for each PI, and cause-effect relationships between PIs have to be defined, based on specifications of performance measures, literature review, and analysis of existing data sets from the community of peers. That was the subject of our previous work.

When multiple potential root causes are identified for a performance problem, it is important to rank (prioritize) the root causes, so that subsequent improvement actions can focus on the highest-ranked root causes. That is the subject of this paper. The ranking approach (to be detailed in section 3) is based on a cost-benefit estimation

that requires as inputs an approximate statistical distribution of each PI and sensitivity data between related PIs.

To enable the automated recommendation of improvement actions for the highest-ranked root cases, a catalogue of possible improvement actions has to be set up for each possible root cause, based on expert knowledge and data from the community of users and experts. That will be the subject of future work.

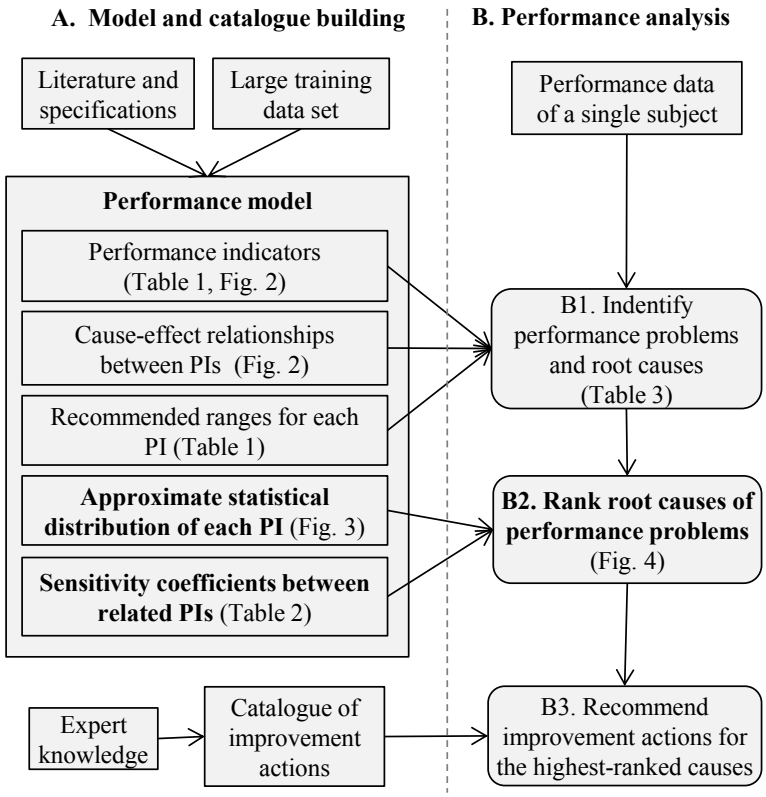

Fig. 1. UML activity diagram depicting our overall performance analysis approach.

2.2 Performance Analysis Model

Fig. 2 and Table 1 present the performance indicators (PIs), cause-effect relationships between PIs and performance ranges developed in our previous work [15] (with minor updates and simplifications) for analyzing the performance of individual PSP developers, based on PSP specifications (of base and derived measures, estimation methods, etc.), literature review, and the analysis of a large PSP data set from the Software Engineering Institute (SEI) containing 31,140 data points (project submissions) from 3,114 engineers that performed 10 projects each, during 295 classes of the classic PSP for Engineers I/II running between 1994 and 2005.

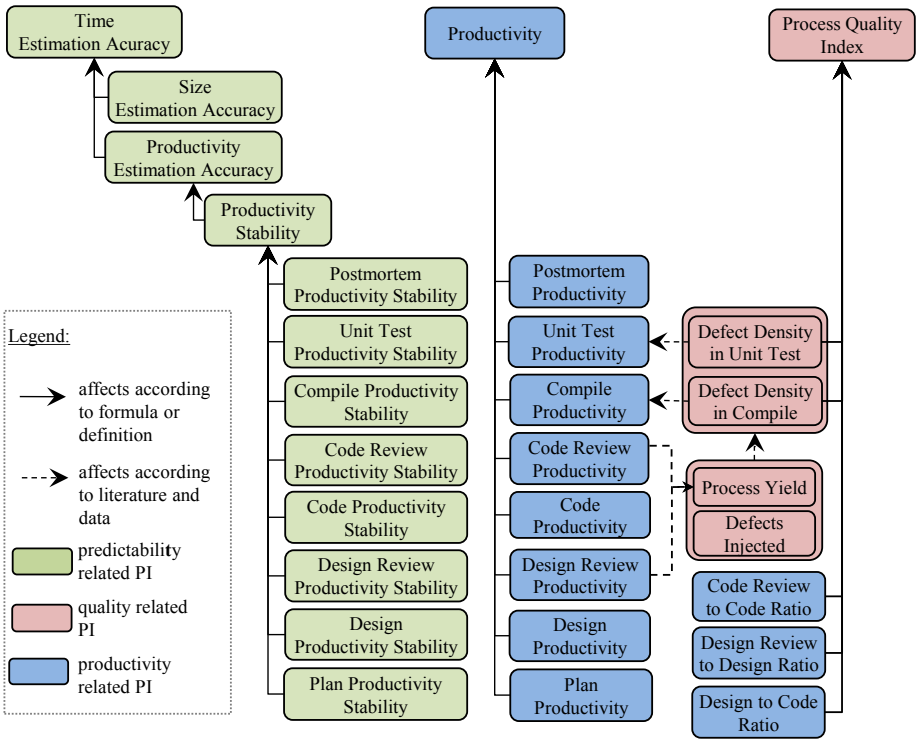

Fig. 2. Performance indicators and cause-effect relationships.

We considered the usual three top-level performance characteristics in software development—predictability (estimation accuracy), quality and productivity—measured in a way specific to the PSP context.

In the PSP, a time (effort) estimate is obtained based on a size estimate of the deliverable (in lines of code, function points, etc.), and a productivity estimate (in added/modified size units per time unit). So, the accuracy of the time estimate will depend on the accuracy of the size and productivity estimates as shown in Fig. 2. Since in the PSP productivity estimates are based on historical productivity, their accuracy depends on the stability of the productivity, as indicated in Fig. 2. In the PSP time is recorded per process phase, so the logical step to follow when an overall productivity stability problem is encountered is to analyze the productivity stability per phase, in order to determine the problematic phase(s). Hence, Fig. 2 shows a set of PIs for the productivity stability per phase, which together affect the overall productivity stability. It is worth noting that the scope of the PSP is the development of small programs or components of larger programs, reason why Requirements, High Level Design and System Testing phases are not included, but can be found in the more complete TSP. In the case of projects developed with programming languages or environments without a separate Compile phase, the Compile phase may be absent.

Table 1. Performance indicators and ranges.

Indicator	Formula	Performance Ranges		
		Green*	Yellow	Red
Time Estimation Accuracy (TimeEA)	$\frac{Actual\ Time}{Estimated\ Time}$	[0.8, 1.2] <u>1</u>	[0.6, 0.8[∪]1.2, 1.4]	[0, 0.6[∪]1.4,∞[
Size Estimation Accuracy (SizeEA)	$\frac{Actual\ Size}{Estimated\ Size}$	[0.8, 1.2] <u>1</u>	[0.55, 0.8[∪]1.2, 1.45]	[0,0.55[∪]1.45,∞[]
Productivity Estimation Accuracy (PEA)	$\frac{Actual\ Productivity}{Estimated\ Productivity}$	[0.8, 1.2] <u>1</u>	[0.6, 0.8[∪]1.2 1.4]	[0, 0.6[∪]1.4,∞[
Productivity Stability (ProdS)	$\frac{Current\ Productivity}{Historical\ Productivity*}$ (*total size/total effort)	[0.8, 1.2] <u>1</u>	[0.6, 0.8[∪]1.2 1.4]	[0, 0.6[∪]1.4,∞[
Process Quality Index (PQI)	$\min(\frac{D2C}{1}, 1) \times \min(\frac{DR2D}{0.5}, 1) \times \min(\frac{CR2C}{0.5}, 1) \times \min(\frac{20}{DDC+10}, 1) \times \min(\frac{10}{DDUT+5}, 1)$	[0.25, <u>1</u>]	[0.06, 0.25[	[0, 0.06[
Defect Density in Unit Test (DDUT)	$\frac{\#Defects\ found\ and\ removed\ in\ Unit\ Test}{Actual\ Size\ (KLOC)}$	[<u>0</u>, 10]	]10, 30]	]30, ∞[
Defect Density in Compile (DDC)	$\frac{\#Defects\ found\ and\ removed\ in\ Compile}{Actual\ Size\ (KLOC)}$	[<u>0</u>, 10]	]10, 40]	]40, ∞[
Defects Injected (DI)	$\frac{\#Defects\ found\ in\ all\ phases}{Actual\ Size\ (KLOC)}$	[<u>0</u>, 50]	]50, 100]	]100, ∞[
Process Yield (PY)	$\frac{\#Defects\ removed\ before\ Compile\ \&\ Test}{\#Defects\ injected\ before\ Compile\ \&\ Test}$	[70,<u>100</u>]	[50, 70[	[0, 50[
Design to Code Ratio (D2C)	$\frac{Design\ Time}{Code\ Time}$	[0.5, 1.5] <u>1</u>	[0.2, 0.5[∪]1.5,2.0]	[0,0.2[∪]2.0, ∞[
Design Review to Design Ratio (DR2D)	$\frac{Design\ Review\ Time}{Design\ Time}$	[0.3, <u>0.5</u>]	[0.1, 0.3[∪]0.5, 0.8]	[0, 0.1[∪]0.8, ∞[
Code Review to Code Ratio (CR2C)	$\frac{Code\ Review\ Time}{Code\ Time}$	[0.3, <u>0.5</u>]	[0.1, 0.3[∪]0.5, 0.6]	[0, 0.1[∪]0.6, ∞[
Productivity (Prod)	$\frac{Actual\ Size\ (LOC)}{Actual\ Time\ (hours)}$	[35, <u>∞</u>[	[20, 35[	]0, 20[
Plan Productivity (PProd)	$\frac{Actual\ Size\ (LOC)}{Plan\ Time\ (hous)}$	[400, <u>∞</u>[	[200, 400[	]0, 200[
Design Productivity (DProd)	$\frac{Actual\ Size\ (LOC)}{Design\ Time\ (hous)}$	[300, <u>∞</u>[	[120, 300[	]0, 120[
Design Review Productivity (DRProd)	$\frac{Actual\ Size\ (LOC)}{Design\ \ Review\ Time\ (hours)}$	[200,400] <u>300</u>	[115,200[∪]400,700]	]0, 115[∪[700,∞[
Code Productivity (CProd)	$\frac{Actual\ Size\ (LOC)}{Code\ Time\ (hours)}$	[120, <u>∞</u>[	[60, 120[	]0, 60[
Code Review Productivity (CRProd)	$\frac{Actual\ Size\ (LOC)}{Code\ \ Review\ Time\ (hours)}$	[150,300] <u>200</u>	[100,150[∪]300,500]	]0, 100[∪]500,∞[
Compile Productivity (CompProd)	$\frac{Actual\ Size\ (LOC)}{Compile\ \ Time\ (hours)}$	[1500, <u>∞</u>[	[500, 1500[	]0, 500[
Unit Test Productivity (UTProd)	$\frac{Actual\ Size\ (LOC)}{Unit\ Test\ \ Time\ (hours)}$	[300, <u>∞</u>[	[100, 300[	]0, 100[
Postmortem Productivity (PMProd)	$\frac{Actual\ Size\ (LOC)}{Postmortem\ \ Time\ (hours)}$	[400, <u>∞</u>[	[200, 400[	]0, 200[

* Optimal value underlined.

Product quality is usually measured by post-delivery defect density [16]. However, since the scope of the PSP is the development of small programs or components of large programs and information about post-delivery defects is often not available, we use the Process Quality Index (PQI) as the top-level quality performance indicator. According to [17], it constitutes an effective predictor of post-delivery defect density. The PQI is computed based on five factors [4]: the ratio of design time to coding time (indicator of design quality); the ratio of design review time to design time (indicator of design review quality); the ratio of code review time to coding time (indicator of code review quality); the ratio of compile defects to a size measure (indicator of code quality); the ratio of unit test defects to a size measure (indicator of program quality). In turn, the analyzed data shows that both the *Defect Density in Compile* and the *Defect Density in Unit Test* are significantly affected by the total density of *Defects Injected* (and found) and the percentage of defects removed before compiling and testing (called *Process Yield* in the PSP) [15]. In turn, existing data shows that the *Process Yield* is significantly affected by the *Design Review Productivity (Rate)* and the *Code Review Productivity,* measured in size units reviewed per time unit [10, 15].

Measuring software development productivity is controversial and all the known productivity measures have limitations [16, 18, 19, 20, 21]. In the PSP, productivity is measured in 'size' units per hour; any size measure can be used (such as function points, lines of code (LOC), etc.) as long as it correlates with effort (in order to enable effort estimation based on size estimation) and can be objectively measured (to automate size measurement and compare actuals and estimates). In this study, we use LOC/hour as the productivity measure, in spite of its limitations, because LOC is the size measure available in the data set. Since in the PSP time is recorded per process phase, the logical step to follow when an overall productivity problem is encountered is to analyze the productivity per phase, in order to determine the problematic phase(s). Hence, we indicate in Fig. 2 a set of PIs for the productivity per phase, which together affect the overall productivity. In turn, the analyzed data shows that the productivity in the *Compile* and *Unit Test* phases is significantly affected by the *Defect Density in Compile* and the *Defect Density in Unit Test*, respectively [15].

Table 1 shows the ranges defined for classifying values of each PI into three categories: green - no performance problem; yellow - a possible performance problem; red - a clear performance problem. These ranges were defined based on recommended values from the literature and the actual distribution of the analyzed PSP data set, so that there is an approximately even distribution of data points by the different colors, in a way similar to benchmark-based software product quality evaluation [25]. In most cases, the 'green' range is located in one of the extremes of the scale, the 'red' range in the other extreme, and the 'yellow' range in the middle. For example, the 'green' range for the Process Quality Index is located in the high values of the [0, 1] scale, whilst for the Defect Density in Unit Test (DDUT) it is located in the low values of the [0, ∞[scale. For several other PIs, the 'green' range is located somewhere in the middle, in order to balance conflicting aspects, such as productivity and quality, as is the case with the *Code Review Productivity*. Table 1 also shows the optimal value considered for each PI, for ranking purposes.

3 Ranking Root Causes of Performance Problems

The presented performance model allows the automated identification of performance problems and root causes for individual engineers. However, when multiple root cases are identified for a performance problem, it does not provide enough information to prioritize (or rank) those root causes. For example, Table 3 identifies 5 causes for the poor productivity in project P7— poor productivity in Plan, Design, Design Review, Unit Test and Postmortem phases — but does not indicate their relative importance.

The main idea for ranking root causes is to use a combination of a *percentile ranking coefficient,* indicating how far a given value is from the optimal value, and a *sensitivity ranking coefficient,* indicating the impact of a change in the affecting PI on the affected PI.

3.1 Sensitivity Coefficients Between Related Performance Indicators

In the presented performance model, several PIs are related by algebraic equations of the general form $Y = f(X_1, ..., X_n)$, where Y denotes the affected PI, and the X_i denote the affecting PIs or factors (see formulas in Table 2). The impact of changes in the value of a factor X_i on the value of Y, whilst keeping all the other factors unchanged, can be computed by the following sensitivity coefficient [26]:

$$\phi_{X_i \to Y} = \frac{\partial Y}{\partial X_i}\left(\frac{X_i}{Y}\right) \tag{1}$$

A sensitivity ranking based on this coefficient will basically compare the relative variations in Y, $\varepsilon_Y = \frac{\Delta Y}{Y}$, for equal relative variations in each of the factors, $\varepsilon_{X_i} = \frac{\Delta X_i}{X_i}$. In fact, the implied variation in Y for a small variation in X_i will be:

$$\varepsilon_Y = \frac{\Delta Y}{Y} \approx \frac{\Delta X_i \times \frac{\partial Y}{\partial X_i}}{Y} = \frac{\partial Y}{\partial X_i}\left(\frac{X_i}{Y}\right)\frac{\Delta X_i}{X_i} = \phi_{X_i \to Y}\, \varepsilon_{X_i} \tag{2}$$

For example, a value $\theta_{X_i \to Y} = 0.5$ means that a 1% change of the current value of X_i will produce approximately a 0.5% relative change in the value of Y. For equal small variations $\varepsilon_{X_1} = \cdots = \varepsilon_{X_n}$, comparing the derived ε_Y reduces to comparing $\phi_{X_1 \to Y}$, ..., $\phi_{X_n \to Y}$. The factor X_i/Y makes the coefficient independent of the scales used. Inherent to this coefficient are the assumptions that the higher ordered partial derivatives are negligible for small variations and that there is no correlation between the input parameters (so that one independent variable can be changed at a time) [26].

In the cases where there isn't an algebraic equation, we use linear regression [27].

Table 2 shows the formulas that relate the PIs indicated in Fig. 2, and the corresponding values or formulas for the sensitivity coefficient.

For example, the sensitivity of the overall productivity on the productivity of a specific phase k is given by the fraction of time in phase k, implying that productivity improvement efforts should be directed towards the more time consuming phases.

Table 2. Dependencies between performance indicators and sensitivity coefficients.

Affected Indicator (Y)	**Affecting Indicator (X)**	**Exact Formula or Regression Formula Y=f(X1, ..., Xn)**	**Sensitivity Coefficient** $\phi_{X\to Y} = \frac{\partial Y}{\partial X}\left(\frac{X}{Y}\right)$
Time Estimation Accuracy *(TimeEA)*	Size Estimation Accuracy *(SizeEA)*	$TimeEA = \frac{SizeEA}{PEA}$	1
	Productivity Estimation Accuracy *(PEA)*		-1
Productivity Estimation Accuracy (*PEA*)	Productivity Stability (*ProdS*)	$PEA \sim 0.593 + 0.455 \times ProdS$	$0.455 \times \frac{ProdS}{PEA}$
Productivity Stability (*ProdS*)	Productivity Stability in Phase *k* ($ProdS_k$)	$ProdS = \frac{1}{\sum_k \frac{HistF_k}{ProdS_k}}$, where $HistF_k$ = historical fraction of time in phase *k*	Fraction of time in phase *k* (in current project)
Process Quality Index (*PQI*)	Defect Density in Unit Test (*DDUT*)	$PQI = \min\left(\frac{10}{DDUT+5}, 1\right) \times \min\left(\frac{20}{DDC+10}, 1\right) \times \min\left(\frac{D2C}{1}, 1\right) \times \min\left(\frac{DR2D}{0.5}, 1\right) \times \min\left(\frac{CR2C}{0.5}, 1\right)$	$-\frac{DDUT}{DDUT+5}$, if DDUT > 5 0, otherwise
	Defect Density in Compile (*DDC*)		$-\frac{DDC}{DDC+10}$, if DDC > 10 0, otherwise
	Design to Code Ratio (*D2C*)		1, if D2C < 1 0, otherwise
	Design Review to Design Ratio (*DR2D*)		1, if DR2D < 0.5 0, otherwise
	Code Review to Code Ratio (*CR2C*)		1, if CR2C < 0.5 0, otherwise
Defect Density in Unit Test (*DDUT*)	Process Yield (*PY*)	$DDUT \sim 28.5 - 0.20 \times PY$	$-0.20 \times \frac{PY}{DDUT}$
	Defects Injected (*DI*)	$DDUT \sim -1.6 + 0.38 \times DI$	$0.38 \times \frac{DI}{DDUT}$
Defect Density in Compile (*DDC*)	Process Yield (*PY*)	$DDC \sim 28.33 - 0.22 \times PY$	$-0.22 \times \frac{PY}{DDC}$
	Defects Injected (*DI*)	$DDC \sim -0.19 + 0.45 \times DI$	$0.45 \times \frac{DI}{DDC}$
Process Yield (*PY*)	Design Review Productivity (*DRProd*)	$PY \sim 57.59 - 0.0030 \times DRProd - 0.0048 \times CRProd$	$-0.0030 \times \frac{DRProd}{PY}$
	Code Review Productivity (*CRProd*)		$-0.0048 \times \frac{CRProd}{PY}$
Productivity (*Prod*)	Productivity in Phase *k* ($Prod_k$)	$Prod = \frac{1}{\sum_k \frac{1}{Prod_k}}$	Fraction of time in phase *k*
Unit Test Productivity (*UTProd*)	Defect Density in Unit Test (*DDUT*)	$UTProd \sim 552 - 4.2 \times DDUT$	$-4.2 \times \frac{DDUT}{UTProd}$
Compile Productivity (*CompProd*)	Defect Density in Compile (*DDC*)	$UTProd \sim 2308 - 17 \times DDC$	$-17 \times \frac{DDC}{CompProd}$

3.2 Percentiles and Combined Ranking Coefficient

The sensitivity ranking approach presented so far compares the impact on Y of relative variations of equal value in the factors $X_1, ..., X_n$, ignoring how far the value of each factor is from its optimal value. Intuitively, the closest a value is to the optimal value, in terms of percentiles, the more difficult (or costly) it is to improve it. So, we propose to compare variations of equal 'cost' instead of variations of equal value.

Let x denote an actual value of X_i, let $F_i(x)$ denote the approximate cumulative distribution function of X_i, let $f_i(x) = \frac{dF_i(x)}{dx}$ denote the approximate probability density function of X_i, let z_i denote the optimal value of X_i (as indicated in Table 1), and let $G_i(x) = F_i(z_i) - F_i(x)$ denote the percentile distance of x to the optimal value.

Our base hypothesis for deriving a combined ranking coefficient is that equal relative variations in the G_i's have equals costs. Then, the combined ranking coefficient becomes the product of two sensitivity coefficients:

$$\rho_{X_i \to Y} = \phi_{G_i \to X_i} \phi_{X_i \to Y} \tag{3}$$

with

$$\phi_{G_i \to X_i} = \frac{\partial X_i}{\partial G_i}\left(\frac{G_i}{X_i}\right) = \frac{F_i(x) - F_i(z_i)}{x f_i(x)} \tag{4}$$

The ranking based on this coefficient will basically compare the relative variations in Y, $\varepsilon_Y = \frac{\Delta Y}{Y}$, for equal relative variations in the percentile distance to the optimal value of each factor X_i, $\varepsilon_{G_i} = \frac{\Delta G_i}{G_i}$. In fact, the implied variation in Y for a small variation in G_i will be:

$$\varepsilon_Y = \frac{\Delta Y}{Y} \approx \frac{\Delta G_i \times \frac{\partial Y}{\partial G_i}}{Y} = \frac{\Delta G_i \times \frac{\partial Y}{\partial X_i} \times \frac{\partial X_i}{\partial G_i}}{Y} = \left[\frac{\partial X_i}{\partial G_i}\left(\frac{G_i}{X_i}\right)\right]\left[\frac{\partial Y}{\partial X_i}\left(\frac{X_i}{Y}\right)\right]\frac{\Delta G_i}{G_i} = \rho_{X_i \to Y}\varepsilon_{G_i} \tag{5}$$

For equal small variations $\varepsilon_{G_1} = \cdots = \varepsilon_{G_n}$, comparing the derived ε_Y reduces to comparing $\rho_{X_1 \to Y}$, ..., $\rho_{X_n \to Y}$.

For example, a value $\rho_{X_i \to Y} = 0.5$ means that a 1% relative change of the current percentile distance to the optimal value of X_i, will produce approximately a 0.5% relative change in the value of Y.

In order to be able to compute $\phi_{G_i \to X_i}$ for each X_i, one needs to know the approximate cumulative distribution function of each X_i. Since some of the performance indicators do not fit known theoretical distributions (e.g., the *Process Yield* follows a hybrid continuous-discrete distribution with non-zero probability at both ends of the scale - see Appendix), we construct an approximate distribution by linear interpolation between a small number of percentile values computed from the training data set (see Appendix). The calculation of $\phi_{G_i \to X_i}$ is illustrated in Fig. 3.

F	CProd
0	0
0.05	27
0.1	38
0.2	53
0.3	67
0.4	81
0.5	95
0.6	113
0.7	134
0.8	165
0.9	221
0.95	284
1	18960

$$\phi_{G_{CProd} \to CProd} = \frac{F(Cprod) - F(\infty)}{f(Cprod) \times CProd} = \frac{0.45 - 1}{0.0069 \times 87.8} = -0.91$$

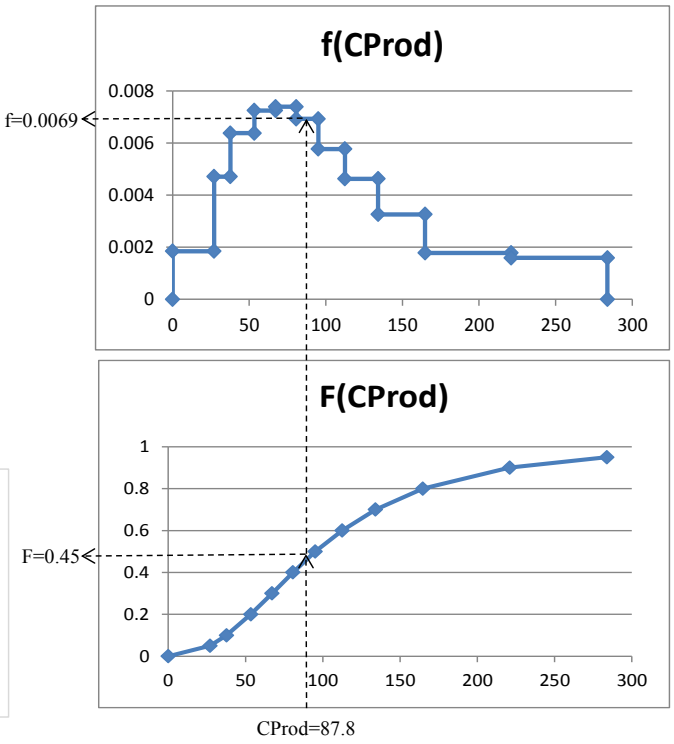

Fig. 3. Computing a ranking coefficient based on percentiles extracted from the data set.

4 Case Study

In the end of the PSP training and at regular times afterwards, developers should analyze their personal performance along the series of projects developed, and document their findings and a set of prioritized and quantified process improvement proposals in a Performance Analysis Report. A goal of our research is to help partially automating this kind of analysis. In this section we describe how the performance of an individual PSP developer can be analyzed based on the proposed model and ranking method. We also compare the results of the model-based analysis with the results of the manual analysis.

In this case, the PSP training sequence (Fundamentals and Advanced) comprised 7 projects. The programming language was Java, without an explicit Compile phase.

The evaluation of the 3 top-level PIs for the 7 projects, together with all 'child' PIs defined in our performance model, is shown in Table 3. The main top-level performance problems occur in time estimation (projects P1, P3 and P7) and in productivity (projects P6 and P7). In order to illustrate and assess the applicability of the ranking method proposed, we computed the ranking coefficients for all pairs of related PIs in project P7, obtaining the results shown in Fig. 4. The child PIs are sorted by descending values of the ranking coefficient. In general, factors with a ranking coefficient

below some threshold (e.g., 1) can be ignored and hidden from the user, leaving only the boxes and links with thick lines.

Table 4 compares the results of manual and model-based analysis, showing that similar conclusions are drawn.

Table 3. Evaluation of the full set of PIs in the case study.

Indicator	P1	P2	P3	P4	P5	P6	P7
Time Estimation Accuracy (TimeEA)	**1.73**	**1.34**	**1.63**	**1.01**	**1.28**	**1.39**	**1.72**
Size Estimation Accuracy (SizeEA)		1.04	1.51	0.96	1.08	1.08	0.98
Productivity Estimation Accuracy (PEA)		0.78	0.93	0.95	0.85	0.78	0.57
Productivity Stability (ProdS)		0.68	0.80	1.17	0.79	0.48	0.37
Plan Productivity Stability (PProdS)		0.20	0.66	0.99	2.13	0.67	0.66
Design Productivity Stability (DProdS)		1.16	1.45	2.00	0.28	0.35	0.15
Design Review Prod. Stability (DRProdS)				1.19	0.35	0.27	0.41
Code Productivity Stability (CProdS)		1.12	1.29	1.42	1.24	0.93	0.80
Code Review Prod. Stability (CRProdS)				2.31	1.08	0.53	0.88
Unit Test Productivity Stability (UTProdS)		0.62	1.50	1.61	0.96	0.60	0.50
Potmortem Productivity Stability (PMProdS)		0.64	0.64	0.82	1.46	0.40	0.49
Process Quality Index (PQI)			**0.38**	**0.07**	**0.29**	**0.26**	**0.12**
Defect Density in Unit Test (DDUT)	26	8	0	20	15	24	17
Defects Injected (DI)	60	16	25	47	67	122	133
Process Yield (PY)			100%	57%	78%	80%	88%
Design to Code Ratio (D2C)	0.52	0.51	0.46	0.35	2.07	1.96	4.54
Code Review to Code Ratio (CR2C)			0.87	0.45	0.62	0.96	0.54
Design Review to Design Ratio (DR2D)			0.57	0.74	0.37	0.64	0.19
Productivity (Prod)	**33.6**	**22.7**	**21.7**	**29.1**	**20.7**	**11.8**	**8.6**
Plan Productivity (PProd)	366	73	79	102	217	77	73
Design Productivity (DProd)	162	188	253	389	64	52	19
Design Review Productivity (DRProd)			443	526	171	82	100
Code Productivity (CProd)	85	95	116	138	132	103	88
Code Review Productivity (CRProd)			134	308	212	107	164
Unit Test Productivity (UTProd)	148	92	169	203	136	84	68
Defect Density in Unit Test (DDUT)	26	8	0	20	15	24	17
Postmortem Productivity (PMProd)	409	261	202	218	366	107	120

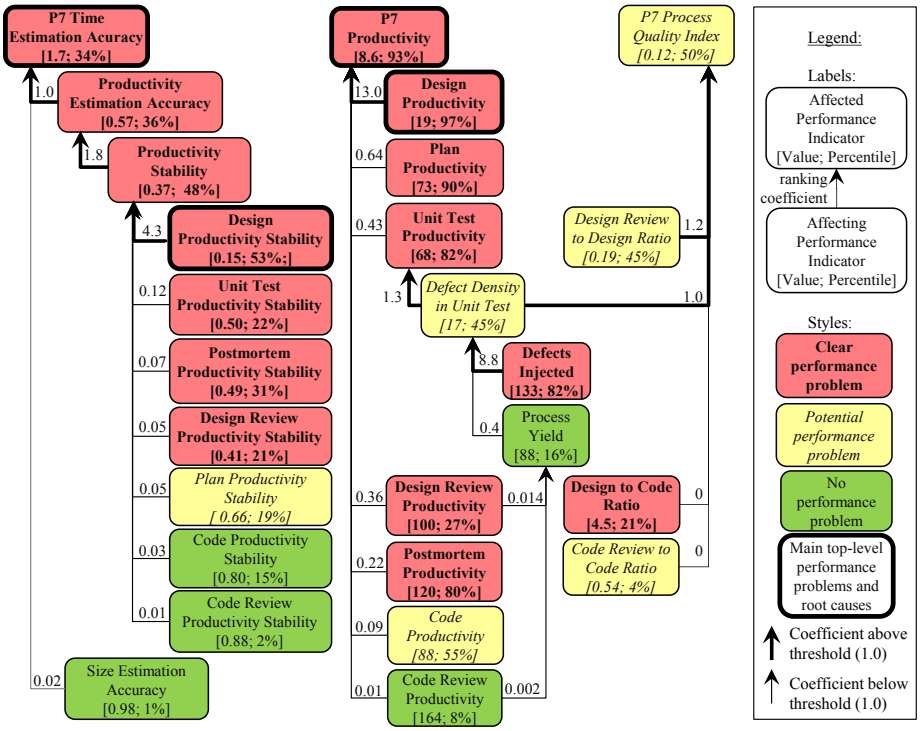

Fig. 4. Ranked root causes of performance problems in project P7 of the case study.

Table 4. Comparison of problems and root causes identified in manual and model-based analysis.

Manual Analysis (PAR)	Model-Based Analysis
Poor time estimation accuracy, with time underestimation in P3 caused by size underestimation, and time underestimation in P7 due mainly to an inneficient and unstable DLD process.	Significant time estimation problems in 3 projects (P1, P3, P7), caused in P3 by a size estimation problem, and in P7 by productivity instability in several phases, notably in DLD.
Product quality problems, with average DDUT well above the recommended value of 5, caused by a high number of defects injected.	No significant process (PQI) and product (DDUT) quality problems, as compared to benchmarks.
Productivity problems, namely at DLD phase, caused by an inneficient DLD process (long design specification documents following PSP templates).	Significant productivity problems in projects P6 and P7, caused by slow performance in several process phases, notably in DLD.

5 Conclusion and Future Work

We proposed a benchmark-based approach for identifying and ranking the root causes of performance problems, and showed its application in the context of the PSP. The case study conducted shows that the approach is able to successfully point out the most important root factors.

We are currently extending our PSP PAIR (Performance Analysis and Improvement Recommendation) tool [13] to support the ranking method presented in this paper. The tool analyzes performance data produced by PSP developers in their projects, and pinpoints performance problems, possible root causes and suggestions for remedial actions.

As future work, we intend to build a comprehensive catalogue of improvement actions to recommend for the highest-ranked root causes, conduct further experiments, and extend the approach for analyzing performance data produced in the context of other processes (namely TSP and Scrum with TSP combinations) and tools (namely cloud-based application lifecycle management tools).

Acknowledgments. The authors would like to acknowledge the SEI, in particular W. Nichols and J. Over, for facilitating the access to the PSP data for performing this study. The work of J. Faria is partly funded by FEDER (Fundo Europeu de Desenvolvimento Regional) through the Portuguese ON.2 Program (Programa Operacional Região do Norte), under project reference SI IDT - 21562/2011. The work of M. Raza 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. Bohem, B.: Some Future Software Engineering Opportunities and Challenges. In: S. Nanz (ed.) The Future of Software Engineering, pp. 1-32, Springer (2011)
2. Humphrey, W.: PSP^{sm}: A Self-Improvement Process for Software Engineers. Addison-Wesley Professional (2015)
3. Rombach, D., Münch, J., Ocampo, A., Humphrey, W., Burton, B.: Teaching disciplined software development. Journal of Systems and Software 81(5): 2008, pp. 747-763, Elsevier (2008)
4. Pomeroy-Huff, M., Cannon, R., Chick, T., Mullaney, J., Nichols, W.: The Personal Software Process^{SM} (PSP^{SM}) Body of Knowledge (Version 2.0). CMU/SEI-2009-SR-018 (2009)
5. Burton, D., Humphrey, W.: Mining PSP Data. In: TSP Symposium 2006 Proceedings (2006)
6. The Software Process Dashboard Initiative, `http://www.processdash.com/`
7. Hackystat, `http://code.google.com/p/hackystat/`
8. Shin, H., Choi, H., Baik, J.: Jasmine: A PSP Supporting Tool. In: Proceedings of the International Conference on Software Process (ICSP 2007), LNCS, vol. 4470, pp. 73-83, Springer (2007)

9. Nasir, M., Yusof, A.: Automating a Modified Personal Software Process. In: Malaysian Journal of Computer Science, vol. 18, pp. 11–27 (2005)
10. Kemerer, C., Paulk, M.: The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data. IEEE Transactions on Software Engineering, vol. 35, Issue 4, pp. 534-550 (2009)
11. Shen, W., Hsueh, N., Lee, W.: Assessing PSP effect in training disciplined software development: A Plan–Track–Review model. Information and Software Technology 53, pp. 137–148 (2011)
12. Humphrey, W.: Personal Software Process (PSP). Encyclopedia of Software Engineering. John Wiley & Sons (2002)
13. Duarte, C., Faria, J.P., Raza, M.: PSP PAIR: Automated Personal Software Process Performance Analysis and Improvement Recommendation. In: Proceedings of the 8th International Conference on the Quality of Information and Communications Technology, pp. 131-136, IEEE (2012)
14. Raza, M., Faria, J.P., Henriques, P., Nichols, W.: Factors Affecting Productivity Performance in PSP Training. In: TSP Symposium 2013 Proceedings, CMU/SEI-2013-SR-022, pp. 35-45, Carnegie Mellon University (2013)
15. Raza, M., Faria, J.P.: A Model for Analyzing Estimation, Productivity and Quality Performance in the Personal Software Process. In: 2014 International Conference of Software and System Process, pp. 10-19, ACM (2014)
16. Jones, C.: Software Assessments, Benchmarks, and Best Practices. Addison Wesley (2000)
17. Humphrey, W.: The Software Quality Profile. SEI, http://www.sei.cmu.edu/library/abstracts/whitepapers/qualityprofile.cfm (2009)
18. Wagner, S., Ruhe, M.: A Systematic Review of Productivity Factors in Software Development. In: Proceedings of 2nd International Workshop on Software Productivity Analysis and Cost Estimation (SPACE 2008), State Key Laboratory of Computer Science, Institute of Software (2008)
19. Maxwell, K., Forselius, P.: Benchmarking Software Development Productivity. IEEE Software, 17(2), pp. 80-88 (2000)
20. Goparaju, P., Farooq, A., Patnaikc, S.: Measuring Productivity of Software Development Teams. Serbian Journal of Management 7 (1), pp. 65-75 (2012)
21. Card, D.: The Challenge of Productivity Measurement. In: Proceedings of the Pacific Northwest Software Quality Conference, Portland, OR. (2006)
22. Scacchi, W.: Understanding Software Productivity. Software Engineering and Knowledge Engineering: Trends for the Next Decade. World Scientific Press (1995)
23. Comstock, C., Jiang, Z., Naudé, P.: Strategic Software Development: Productivity Comparisons of General Development Programs. International Journal of Computer and Information Engineering 1:8, pp. 486-491 (2007)
24. Banker, R., Kauffman, R.: Reuse and Productivity in Integrated Computer-Aided Software Engineering: An Empirical Study. MIS Quarterly, Sept 1991, 14(3):374-401 (1991)
25. Alves, T.: Benchmark-based Software Product Quality Evaluation. Doctoral Thesis, University of Minho (2012)
26. Hamby, D.M.: A Review of Techniques for Parameter Sensitivity Analysis of Environmental Models. Environmental Monitoring and Assessment, Springer, September 1994, Volume 32, Issue 2, pp. 135-154 (1994)
27. Navidi, W.: Statistics for Engineers and Scientists, Third Edition. McGraw-Hill (2011)

Appendix: Cumulative Distribution Functions