JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/smr

A Model for Analyzing Performance Problems and Root Causes in the Personal Software Process

M. Raza, J. P. Faria

INESC TEC and Department of Informatics Engineering, Faculty of Engineering, University of Porto Porto, Portugal

SUMMARY

High-maturity software development processes, such as the Team Software Process (TSP) and the accompanying Personal Software Process (PSP), can generate significant amounts of data that can be periodically analyzed to identify performance problems, determine their root causes and devise improvement actions. However, there is a lack of tool support for automating that type of analysis, and hence diminish the manual effort and expert knowledge required. So, we propose in this paper a comprehensive performance model, addressing time estimation accuracy, quality and productivity, to enable the automated (tool based) analysis of performance data produced by PSP developers, namely, identify and rank performance problems and their root causes. A PSP data set referring to more than 30,000 projects was used to validate and calibrate the model.

Received …

KEYWORDS: Personal Software Process; Performance Analysis; Performance Model.

1. INTRODUCTION

Currently, according to [1], the top two software engineering challenges are the increasing emphasis on rapid development and adaptability, and the increasing software criticality and need for assurance. The Team Software Process (TSP) [2] and the accompanying Personal Software Process (PSP) [3] are examples of processes that can help individuals and teams improve their performance and produce virtually defect free software on time and budget [4], addressing those 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 and quality management and process improvement [2].

Software processes that make intensive use of metrics and quantitative methods, such as the TSP/PSP, can generate large amounts of data 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 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 (PIs) [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 challenging because of the potentially large amount of data to analyze [5] and the effort and expert knowledge required.

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 and rank performance problems and their root causes and recommend improvement actions. In [12] we presented a model (validated based on a small data set) and a tool to automate the analysis of time estimation accuracy of PSP developers. In [13] we investigated factors affecting the productivity of PSP developers, based on a large PSP data set referring to more than 30,000 projects. In [14] we presented a comprehensive performance model (PM), covering the estimation, quality and productivity aspects, calibrated based on a large PSP data set, to enable the automated analysis of performance data produced by PSP developers. In the current paper we tackle the following research question ***(RQ1)****: Is it possible to automatically analyze performance data produced in the context of the PSP, namely identify and rank performance problems and their potential root causes, with a similar accuracy as in manual analysis but with less effort?* To that end we present an improved approach and PM, with tool support, covering not only the identification of performance problems and potential root causes, but also their ranking, with the following main contributions:

- an overall approach for automated model-based process performance analysis and improvement recommendation, independent of the process under analysis, with minor improvements with respect to our previous work (Section 2);
- a detailed ranking approach for prioritizing the factors affecting the performance problems identified in top-level PIs, according to a cost-benefit estimate, independent of the process under analysis (Section 5);
- a comprehensive PM to instantiate the overall approach for the PSP, with minor simplifications and significant extensions as compared to our previous work; we removed from the PM some factors that exhibited a weak correlation with the top-level PIs (Sections 3 and 4); we added to the PM new attributes needed to support the ranking approach—sensitivity coefficients between pairs of related PIs and approximate statistical distribution of each PI (Section 5 and Appendix A);
- a detailed description of a case study demonstrating how the model can be applied in the real world and the associated benefits (Section 6);
- a novel tool, named ProcessPAIR, for automating the performance analysis, freely available, and described together with the case study (Section 6).

The rest of the paper is organized as follows. Section 2 describes our overall approach. Sections 3 presents the PM conceived for analyzing PSP performance data. Section 4 describes the model validation and calibration procedures. Section 5 describes the ranking approach. Section 6 presents a case study to illustrate the application of the model and compare the results of model-based and manual analysis. Comparison with related work and limitations and threats to validity are discussed in Sections 7 and 8. Section 9 presents the conclusions. Appendix A shows the approximate statistical distributions of the PIs. Appendix B shows an example of ranking calculations.

2. OVERALL 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 (B1), one has to first decide on the relevant PIs (A1) and ranges (A3). In order to enable the automated identification of root causes of performance problems (B2), one has to first decide on the relevant cause-effect relationships (A2). When multiple root causes are identified for a performance problem, it is important to rank them according to a cost-benefit estimate of improvement efforts (B3). In the approach proposed in this paper (see section 5), the cost of improving an affecting PI (root cause) is estimated based on its approximate statistical distribution (A4), and the benefit on the affected PI is estimated based on a sensitivity coefficient (A5). The final step is to recommend improvement actions to address the highest ranked causes (B4). To enable the automated recommendation of such actions, a catalogue of possible improvement actions for each possible root cause has to be set up (A6).

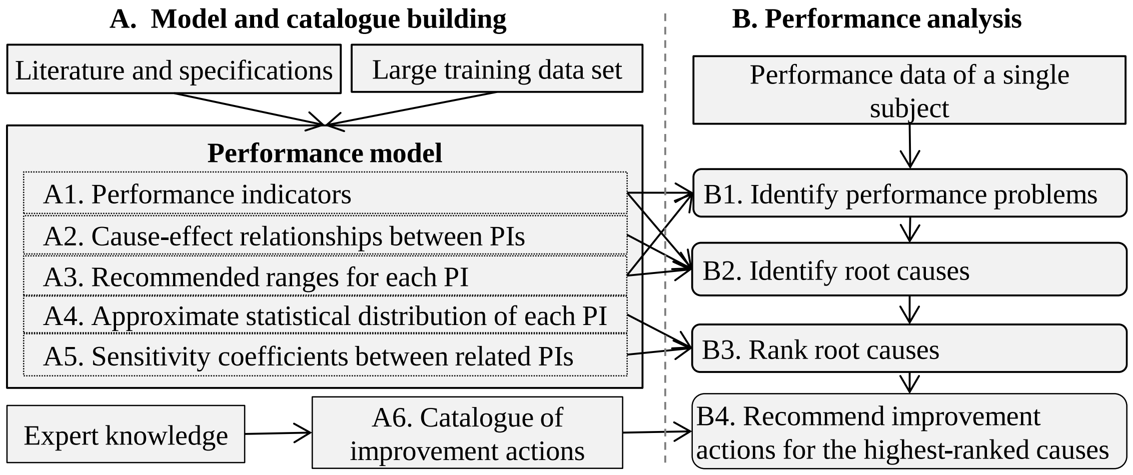

A. Model and catalogue building
B. Performance analysis
Literature and specifications
Large training data set
Performance data of a single subject
Performance model
A1. Performance indicators
A2. Cause-effect relationships between PIs
A3. Recommended ranges for each PI
A4. Approximate statistical distribution of each PI
A5. Sensitivity coefficients between related PIs
B1. Identify performance problems
B2. Identify root causes
B3. Rank root causes
Expert knowledge
A6. Catalogue of improvement actions
B4. Recommend improvement actions for the highest-ranked causes

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

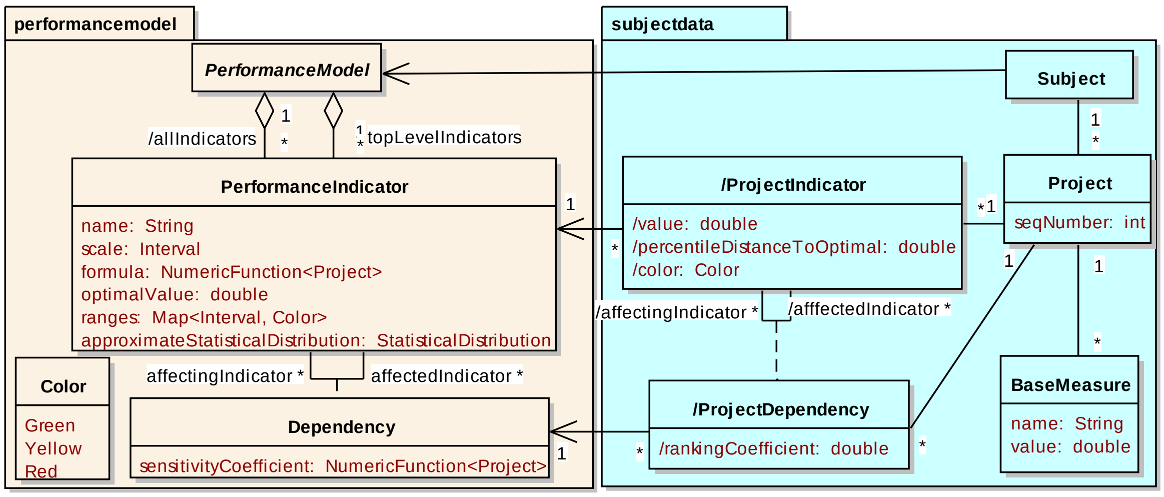


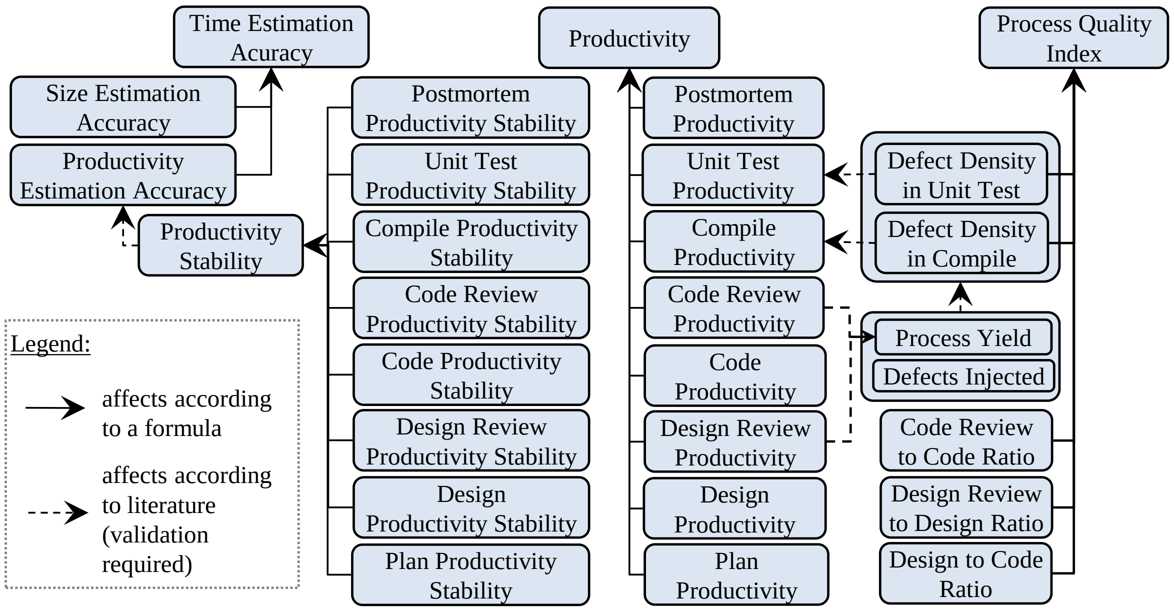
Figure 2. UML class diagram depicting the conceptual structure of the artifacts in our approach.

The conceptual structure of the artifacts involved in our approach is depicted in Fig. 2. A *performance model* comprises a set of *performance indicators* (top level and nested) and *dependencies*, with their respective attributes. For any given *subject* under analysis, we assume relevant *base measures* are available for the *projects* under analysis. The outputs of the model-based performance analysis are indicated in Fig. 2 by derived classes and attributes. For each project, it is computed the value, percentile and 'color' of each PI, based respectively on the formulas, approximate statistical distributions and ranges defined in the PM; it is also computed the value of a *ranking coefficient* (see section 5) for each dependency defined in the PM, so that affecting indicators can be filtered and sorted when drilling down from higher-level (affected) to lower-level (affecting) PIs.

In the next sections it will be presented a PM specifically conceived for the PSP, comprising artifacts A1 through A5, as well as further details about our approach.

3. MODEL CONSTRUCTION

Figure 3, Table 2 and Table 3 summarize the PIs and dependencies of the PM that we conceived, based on literature review and PSP specifications, for analyzing performance problems and root causes in the context of the PSP. The three top-level PIs refer to the major performance aspects usually analyzed: predictability, quality and productivity.



Time Estimation Acuracy
Productivity
Process Quality Index
Size Estimation Accuracy
Productivity Estimation Accuracy
Productivity Stability
Postmortem Productivity Stability
Unit Test Productivity Stability
Compile Productivity Stability
Code Review Productivity Stability
Code Productivity Stability
Design Review Productivity Stability
Design Productivity Stability
Plan Productivity Stability
Postmortem Productivity
Unit Test Productivity
Compile Productivity
Code Review Productivity
Code Productivity
Design Review Productivity
Design Productivity
Plan Productivity
Defect Density in Unit Test
Defect Density in Compile
Process Yield
Defects Injected
Code Review to Code Ratio
Design Review to Design Ratio
Design to Code Ratio
Legend:
affects according to a formula
affects according to literature (validation required)

Figure 3 Performance indicators and dependencies.

3.1 *Predictability*

The major predictability PI in the PSP is the *Time Estimation Accuracy*, which we measure by the ratio between actuals and estimates, to simplify ranking calculations. Since in the PSP's PROBE estimation method [2], a time (effort) 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 indicate in Fig. 3 that the *Time Estimation Accuracy* is affected by the *Size Estimation Accuracy* and the *Productivity Estimation Accuracy* (the exact formula for this dependency is shown in Table 3).

Since in the PROBE method productivity estimates are based on historical productivity [2], we indicate in Fig. 3 that the *Productivity Estimation Accuracy* depends on the *Productivity Stability* (see exact definition in Table 2).

Since in the PSP time is recorded per process phase [2], when a productivity stability problem is encountered one can analyze the productivity stability per phase, in order to determine the problematic phase(s); hence, we indicate in Fig. 3 a set of PIs for the productivity stability per phase, which together affect the overall productivity stability (the exact formula for this dependency is shown in Table 3). Since the scope of the PSP is the development of small programs or components of larger programs, the Requirements, High Level Design and System Testing phases are absent (they can be found in the more complete TSP [3]). In some programming environments the Compile phase may be absent.

3.2 Quality

Product quality is usually measured by post-delivery defect density [15]. However, since the scope of the PSP is the development of small programs or components of large programs, post-delivery defects are seldom known. The PSP proposes an aggregated quality measure—the *Process Quality Index (PQI)*—that constitutes an effective predictor of post-delivery defect density [2][16]. Hence, we use the PQI as the top-level quality indicator to analyze. The PQI is computed based on five components: 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). The components are normalized to [0, 1] such that 0 represents poor practice and 1 represents desired practice (see exact formula in Table 3). Hence, in Fig. 3 we indicate those components as factors that affect the PQI according to a formula.

In turn, both the *Defect Density in Compile* and *Unit Test* are affected by the total density of *Defects Injected* (and found) and the percentage of defects removed before compile and test (called *Process Yield* in the PSP). In fact, high defect densities in compile and test may be caused by a large number of defects injected (due to poor defect prevention) or a large number of defects escaped from previous defect filters (due to poor design and code reviews).

According to [2][17], the time spent in reviewing a work product in relation to its size is a leading indicator of the review yield (percentage of defects found) and consequently of the process yield. In a published study [10], the recommended review rate of 200 LOC/hour or less was found to be an effective rate, identifying nearly two-thirds of the defects in design reviews and more than half in code reviews. Hence, we indicate in Fig. 3 that the *Process Yield* is affected by the *Design Review Productivity* and the *Code Review Productivity*.

3.3 Productivity

In general, in the PSP, productivity may be measured in any size units per time units. Any size measure can be used (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) [2]. In this study, we use LOC/hour, in spite of its well-known limitations [18][19][20][21], because LOC is the size measure available in the data set we used.

Since in the PSP time is recorded per process phase, when a productivity problem is encountered one can analyze the productivity per phase, in order to determine the problematic phase(s). Hence, we indicate in Fig. 3 a set of PIs for the productivity per phase, which together affect the overall productivity (the formula for this dependency is shown in Table 3).

In turn, the time spent in the compile and test phases (which in the PSP include defect fixing) may be affected by the number of defects to fix, so we indicate in Fig. 3 that the *Compile Productivity* and the *Unit Test Productivity* may be affected by the *Defect Density in Compile* and *Defect Density in Unit Test,* respectively.

4. MODEL VALIDATION AND CALIBRATION

4.1 Data set

To validate and calibrate the 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.

4.2 Model validation

The PM of Fig. 3 indicates several 'affects according to literature' relationships between pairs of PIs, suggested from literature. In order to validate each relationship, using the PSP data set previously described, we computed the Pearson's linear correlation coefficient ($r_{pearson}$) and tested the null hypothesis "H_0: $r=0$" against the alternative hypothesis "H_1: $r>0$" or "H_1: $r<0$", depending on the sign of the expected correlation. Since the PIs under analysis may have non-linear relationships that are not adequately captured by the Pearson's linear correlation coefficient, we also computed the Spearman's [22] rank correlation coefficient ($r_{spearman}$). The Spearman's test checks if increasing values of X are monotonically associated with increasing ($r>0$) or decreasing ($r<0$) values of Y, independently of the form of the relationship. The results are presented in Table 1. In all cases, it was observed a statistically significant correlation between the PIs under analysis, regarding both the Person and Spearman correlation, so the null hypothesis was rejected. We tested other dependencies, but only present in this paper dependencies that exhibited a statistically significant correlation.

Table 1. Results of the correlation tests.

Affected Indicator (Y)	Affecting Indicator (X)	Correlation tests (H_0: r=0)					
		H_1	n (1)	$r_{pearson}$	$r_{spearman}$	p (2)	Reject H_0?(3)
Defect Density in Unit Test	Process Yield	r<0	9612	-0.27	-0.40	<2e-16	Yes
Defect Density in Unit Test	Defects Injected	r>0	27648	0.72	0.65	<2e-16	Yes
Defect Density in Compile	Process Yield	r<0	9612	-0.24	-0.39	<2e-16	Yes
Defect Density in Compile	Defects Injected	r>0	27648	0.74	0.69	<2e-16	Yes
Process Yield	Design Review Productivity	r<0	9371	-0.17	-0.23	<2e-16	Yes
Process Yield	Code Review Productivity	r<0	9548	-0.17	-0.25	<2e-16	Yes
Unit Test Productivity	Defect Density in Unit Test	r<0	27625	-0.20	-0.64	<2e-16	Yes
Compile Productivity	Defect Density in Compile	r<0	27625	-0.34	-0.72	<2e-16	Yes
Product. Estim. Accuracy	Productivity Stability	r>0	24574	0.46	0.65	<2e-16	Yes

(1) *n* denotes the number of data points with defined values for the variables under analysis.
(2) *p* is a probability that indicates the statistical significance of the correlation coefficient in the one-tailed test.
(3) We reject the null hypothesis if $p<0.05$, for a 5% significance level.

4.3 Model calibration

Regarding model calibration, we defined a set of thresholds and ranges (see Table 2) for classifying values of each PI into three categories: green - no performance problem; yellow - a possible performance problem; red - a clear performance problem. To define the ranges we took into account the definition of each PI, recommended values from literature, and the actual distribution in the PSP data set, so that there is an approximately even distribution of data points by the colors, in a way similar to benchmark-based product evaluation [23]. We didn't use thresholds based on mean and standard deviation because many PIs don't follow a normal distribution and some do even exhibit a hybrid continuous-discrete distribution in the PSP data set (see Appendix A or Fig. 6).

Table 2. Performance indicators and ranges (with optimal values underlined).

Indicator	Formula	Performance Ranges [(1)]		
		Green	**Yellow**	**Red**
Time Estimation Accuracy	$\frac{Actual\ Time}{Estimated\ Time}$	[0.8, 1.2] $\underline{1}$	[0.6, 0.8[∪]1.2, 1.4]	[0, 0.6[∪]1.4,∞[
Size Estimation Accuracy	$\frac{Actual\ Size}{Estimated\ Size}$	[0.8, 1.2] $\underline{1}$	[0.55, 0.8[∪]1.2, 1.45]	[0, 0.55[∪]1.45,∞[
Productivity Estimation Accuracy	$\frac{Actual\ Productivity}{Estimated\ Productivity}$	[0.8, 1.2] $\underline{1}$	[0.6, 0.8[∪]1.2 1.4]	[0, 0.6[∪]1.4,∞[
Productivity Stability	$\frac{Current\ Productivity}{Historical\ Productivity*}$ (*∑size/∑effort)	[0.8, 1.2] $\underline{1}$	[0.6, 0.8[∪]1.2 1.4]	[0, 0.6[∪]1.4,∞[
Process Quality Index	(see definition in Table 3)	[0.25, $\underline{1}$]	[0.06, 0.25[	[0, 0.06[
Defect Density in Unit Test	$\frac{\#Defects\ found\ and\ removed\ in\ Unit\ Test}{Actual\ Size\ (KLOC)}$	[$\underline{0}$, 10]	]10, 30]	]30, ∞[
Defect Density in Compile	$\frac{\#Defects\ found\ and\ removed\ in\ Compile}{Actual\ Size\ (KLOC)}$	[$\underline{0}$, 10]	]10, 40]	]40, ∞[
Defects Injected	$\frac{\#Defects\ found\ in\ all\ phases}{Actual\ Size\ (KLOC)}$	[$\underline{0}$, 50]	]50, 100]	]100, ∞[
Process Yield	$\frac{\#Defects\ removed\ before\ Compile\ \&\ Test}{\#Defects\ injected\ before\ Compile\ \&\ Test} \times 100$	[70,$\underline{100}$]	[50, 70[	[0, 50[
Design to Code Ratio	$\frac{Design\ Time}{Code\ Time}$	[0.5, 1.5] $\underline{1}$	[0.2, 0.5[∪]1.5, 2.0]	[0, 0.2 [∪]2.0, ∞[
Design Review to Design Ratio	$\frac{Design\ Review\ Time}{Design\ Time}$	[.35, .65[$\underline{0.5}$	[0.2, 0.35[∪]0.65, 0.9]	[0, 0.2 [∪]0.9, ∞[
Code Review to Code Ratio	$\frac{Code\ Review\ Time}{Code\ Time}$	[.35, .65[$\underline{0.5}$	[0.2, 0.35[∪]0.65, 0.9]	[0, 0.2 [∪]0.9, ∞[
Productivity	$\frac{Actual\ Size\ (LOC)}{Actual\ Time\ (hours)}$	[35, $\underline{\infty}$[	[20, 35[	]0, 20[
Plan Productivity	$\frac{Actual\ Size\ (LOC)}{Plan\ Time\ (hours)}$	[400, $\underline{\infty}$[	[200, 400[	]0, 200[
Design Productivity	$\frac{Actual\ Size\ (LOC)}{Design\ Time\ (hours)}$	[300, $\underline{\infty}$[	[120, 300[	]0, 120[
Design Review Productivity	$\frac{Actual\ Size\ (LOC)}{Design\ \ Review\ Time\ (hours)}$	[200,400] $\underline{300}$	[115,200[∪]400,700]	]0, 115[∪ [700,∞[
Code Productivity	$\frac{Actual\ Size\ (LOC)}{Code\ Time\ (hours)}$	[120, $\underline{\infty}$[	[60, 120[	]0, 60[
Code Review Productivity	$\frac{Actual\ Size\ (LOC)}{Code\ \ Review\ Time\ (hours)}$	[150,300] $\underline{200}$	[100,150[∪]300,500]	]0, 100[∪]500,∞[
Compile Productivity	$\frac{Actual\ Size\ (LOC)}{Compile\ \ Time\ (hours)}$	[1500, $\underline{\infty}$[	[500, 1500[	]0, 500[
Unit Test Productivity	$\frac{Actual\ Size\ (LOC)}{Unit\ Test\ \ Time\ (hours)}$	[300, $\underline{\infty}$[	[100, 300[	]0, 100[
Postmortem Productivity	$\frac{Actual\ Size\ (LOC)}{Postmortem\ \ Time\ (hours)}$	[400, $\underline{\infty}$[	[200, 400[	]0, 200[

(1) Open ranges are represented with the Bourbaki notation.

The first step is the definition of an *optimal value* for each PI. In most cases, the optimal value follows directly from the semantics of the PI, being it the maximum of the scale (e.g., 1 for the PQI), the minimum (e.g., 0 for the *Defect Density in Unit Test*), or a special value in between (e.g., 1 for *Estimation Accuracy* and *Productivity Stability*). In other cases, in order to balance conflicting aspects such as quality and productivity with an ultimate economic impact, we selected a recommended value from literature (we could not calibrate those values from the PSP data set, because some economic impacts occur later in the process). Regarding the *Code Review Productivity* (also called *Review Rate*), if reviews are performed too fast then the quality of the reviews may suffer; if reviews are performed too slowly, then the productivity is negatively affected. In this case, we selected the recommended value of 200 LOC/hour [2][17]. A similar reasoning was followed for the *Design Review Productivity*. As for the *Design to Code Ratio*, *Design Review to Design Ratio*, and *Code Review to Code Ratio* (components of the PQI) the optimal values selected correspond to the desired values indicated in the definition of the PQI [2][16].

After defining the optimal values, the thresholds for the 'green', ‘red’ and ‘yellow’ ranges were calibrated in a systematic way based on the PSP data set. In case the optimal value is located in one of the extremes of the scale, the 'green' range is also located in the same extreme of the scale, the 'red' range in the other extreme, and the 'yellow' range in the middle. In these cases, thresholds were computed from the data set based on terciles and subsequently rounded to a few precision digits. In case the optimal value is located somewhere in the middle of the scale, we determined a ‘green’ range around the optimal value containing approximately 1/3 of the data points, and considered 'red' values to exist in both ends of the scale.

5. RANKING APPROACH

5.1 Introduction

The PM presented so far allows the automated identification of performance problems and potential root causes for individual developers. However, when multiple potential root causes are identified for a performance problem, it does not provide enough information to prioritize or rank those root causes. For example, Fig. 4 suggests 5 potential causes for the poor productivity in project 7— poor productivity in Plan, Design, Design Review, Unit Test and Postmortem phases — but does not indicate their relative importance. We propose next an approach to rank those factors according to a cost-benefit estimate of improvement efforts.

Let $X_1, ..., X_n$ be a set of lower-level PIs that affect the value of a higher-level PI Y. We rank the factors X_i according to a cost-benefit estimate of improving each factor X_i whilst keeping the other factors unchanged. We assume that the above relationship can be described by a function $Y=f(X_1, ..., X_n)$, representing an exact formula for deriving Y or a regression formula derived from historical data. We also assume that an optimal value o_i (see Section 4.3) and an approximate cumulative distribution function F_i are defined for each X_i.

The *benefit* of a change in the value of a factor X_i can be expressed by the resulting variation in the value of Y, i.e., $\Delta Y/Y$. As for the *cost* of changing the value of a factor X_i, intuitively, the closest the value is to the optimal value, in terms of percentiles, the more difficult (and less relevant) it is to improve it. Let us denote by $P_i(x)=F_i(o_i)-F_i(x)$ the ‘percentile distance’ of a value x of X_i to the optimal value. Our base heuristic is that equal relative variations in the $P_i's$ have similar costs. So, we take as cost estimate the relative variation $\Delta P_i/P_i$.

We approximate the cost-benefit ratio using partial derivatives (for small variations) to derive a *ranking coefficient* (ρ_i):

$$\frac{\Delta Y/Y}{\Delta P_i/P_i}=\left[\frac{\Delta Y}{\Delta X_i}\left(\frac{X_i}{Y}\right)\right]\times\left[\frac{\Delta X_i}{\Delta P_i}\left(\frac{P_i}{X_i}\right)\right]\approx\left[\frac{\partial Y}{\partial X_i}\left(\frac{X_i}{Y}\right)\right]\times\left[\frac{dX_i}{dP_i}\left(\frac{P_i}{X_i}\right)\right]=\sigma_{X_i\to Y}\times\pi_{P_i\to X_i}=\rho_i$$

For example, a value $\rho_i = 0.5$ means that a 1% relative variation in the current percentile distance to the optimal value of X_i, will produce approximately a 0.5% relative variation in the value of Y. This way, if a factor X_i produces a bigger relative variation in the value of Y than a factor X_j, for the same relative variations in the percentile distance to the optimal value of X_i and X_j, then we will have $\rho_i > \rho_j$. The obtained ranking coefficient ρ_i is the product of two coefficients further analyzed next.

5.2 *Sensitivity coefficient*

The first coefficient, $\sigma_{X_i\to Y}=\frac{\partial Y}{\partial X_i}\left(\frac{X_i}{Y}\right)$, is a sensitivity coefficient [24][25] that computes the impact of small variations in the value of a factor X_i on the value of Y, whilst keeping all the other factors unchanged. For example, a value $\sigma_{X_i\to Y} = 2.0$ means that a 1% variation in the current value of X_i will produce approximately a 2% variation in the current value of Y. The multiplier X_i/Y makes the coefficient independent of the measurement scales used. Inherent to this coefficient are the following assumptions: (i) the higher ordered partial derivatives are negligible for small variations, so that $\sigma_{X_i\to Y}\approx\frac{\Delta Y/Y}{\Delta X_i/X_i}$; (ii) there is no correlation between the factors, so that one factor can be changed at a time [24].

Table 3 shows the computation of the sensitivity coefficient for the dependencies identified in the PM of Fig. 3. For example, the overall productivity (*Prod*) is related with the productivity per phase (*Prod$_k$*) according to the formula $Prod = 1/\sum_k \frac{1}{Prod_k}$, where k denotes a process phase. From this formula and the definition of productivity as a size by effort ratio we can derive the sensitivity coefficients $\sigma_{Prod_k\to Prod}=\frac{Prod}{Prod_k}=\frac{effort\ in\ phase\ k}{total\ project\ effort}$. This formula basically tells that *Prod$_k$* will be ranked higher for the phases that consume more effort. This implies that productivity improvement efforts should be directed towards the more time consuming phases (according to this coefficient alone).

5.3 *Percentile coefficient*

The second coefficient, $\pi_{P_i\to X_i}=\frac{dX_i}{dP_i}\left(\frac{P_i}{X_i}\right)=\frac{F_i(x)-F_i(o_i)}{xF_i'(x)}$, which we call a *percentile coefficient*, computes the impact of small variations in the current percentile distance *(P_i)* of X_i to the optimal value *(o_i)* on the value of X_i. We denote by $F_i'(x)$ the first derivative of $F_i(x)$, representing the probability density function. For small variations we'll have $\pi_{P_i\to X_i}\approx\frac{\Delta X_i/X_i}{\Delta P_i/P_i}$. For example, a value $\pi_{P_i\to X_i}=1.0$ means that a 1% relative reduction in the current percentile distance of X_i to the optimal value (say, from 0.50 to 0.495) will produce approximately a 1% variation in the current value of X.

The cumulative distribution function F_i needed for calculating the percentile coefficient $\pi_{P_i\to X_i}$ can be obtained by computing a theoretical distribution that best fits the historical data, or by linear interpolation between a few percentiles computed from the historical data. Since some of the PIs exhibit a hybrid continuous-discrete distribution, with non-zero probability at one or both ends of the scale (see *Process Yield*, *Defect Density in Compile*, and *Defect*

Density in Unit Test in Appendix A), we opted for the second method. The shapes of the cumulative distribution functions F_i constructed this way from the historical data are depicted in Appendix A. The calculation of $\pi_{P_i \to X_i}$ with this approach is illustrated in Fig. 8 in Appendix B. An example of ranking calculations is also illustrated in Appendix B.

Table 3. Dependencies and sensitivity coefficients between related performance indicators.

Affected Indicator (Y)	**Affecting Indicator (X_i)**	**Exact Formula or Regression Formula Y=f(X_1, ..., X_n)**	**Sensitivity Coefficient** $\sigma_{X_i \to Y} = \frac{\partial Y}{\partial X_i}\left(\frac{X_i}{Y}\right)$
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*)	$\mathrm{PEA} \sim 0.593 + 0.455 \times \mathrm{ProdS}$	$0.455 \times ProdS/PEA$
Productivity Stability (*ProdS*)	Productivity Stability in Phase *k* (*ProdS$_k$*)	$ProdS = 1/\sum_k \frac{HistF_k}{ProdS_k}$, 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*)	$\mathrm{PQI} = \min\left(\frac{10}{DDUT+5}, 1\right) \times \min\left(\frac{20}{DDC+10}, 1\right) \times \min\left(\frac{D2C}{1}, 1\right) \times \min\left(\frac{DR2D}{0.5}, 1\right) \times \min\left(\frac{CR2C}{0.5}, 1\right)$	*-DDUT/(DDUT+5),* if DDUT > 5; 0, otherwise
	Defect Density in Compile (*DDC*)		$-DDC/(DDC+10)$, 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*)	$DDUT \sim 28.5 - 0.20 \times PY$	$-0.20 \times PY/DDUT$
	Defects Injected (*DI*)	$\mathrm{DDUT} \sim -1.6 + 0.38 \times \mathrm{DI}$	$0.38 \times DI/DDUT$
Defect Density in Compile (*DDC*)	Process Yield (*PY*)	$\mathrm{DDC} \sim 28.33 - 0.22 \times PY$	$-0.22 \times PY/DDC$
	Defects Injected (*DI*)	$\mathrm{DDC} \sim -0.19 + 0.45 \times \mathrm{DI}$	$0.45 \times DI/DDC$
Process Yield (*PY*)	Design Review Productivity (*DRProd*)	$PY \sim 57.59 - 0.0030 \times DRProd - 0.0048 \times CRProd$	$-0.0030 \times DRProd/PY$
	Code Review Productivity (*CRProd*)		$-0.0048 \times CRProd/PY$
Productivity (*Prod*)	Productivity in Phase *k* (*Prod$_k$*)	$Prod = 1/\sum_k \frac{1}{Prod_k}$	Fraction of time in phase *k*
Unit Test Productivity (*UTProd*)	Defect Density in Unit Test (*DDUT*)	$\mathrm{UTProd} \sim 552 - 4.2 \times \mathrm{DDUT}$	$-4.2 \times DDUT/UTProd$
Compile Productivity (*CompProd*)	Defect Density in Compile (*DDC*)	$\mathrm{CompProd} \sim 2308 - 17 \times \mathrm{DDC}$	$-17 \times DDC/CompProd$

6. CASE STUDY

6.1 Case study design and planning

The objective of the case study described in this section, related with RQ1, is to show that it is possible to automatically analyze the performance data of an individual PSP developer in order to identify and rank performance problems and potential root causes, based on the PM and approach proposed in this paper with tool support, with a similar accuracy as in manual analysis but with less effort.

In this case, the performance data under analysis refers to 7 projects performed by a PSP developer during the PSP Fundamentals and Advanced training, using a programming language without an explicit Compile phase. The projects deal with numerical and text processing problems. The data was recorded using the SEI's PSP Student Workbook. 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 (PAR); such PAR is also available in this case and will be used for comparison purposes.

The automated analysis was conducted with version 2.2 of our novel ProcessPAIR tool available in http://blogs.fe.up.pt/processpair/. The tool is able to import and automatically analyze the performance data recorded by PSP developers in the SEI's PSP Student Workbook or the Process Dashboard tool (http://www.processdash.com/). The ProcessPAIR tool comprises a core framework, independent of the process under analysis, and an extension for the PSP, providing the PM described in this paper and loaders from the PSP tools.

6.2 *Case study data collection: model-based performance analysis outcomes*

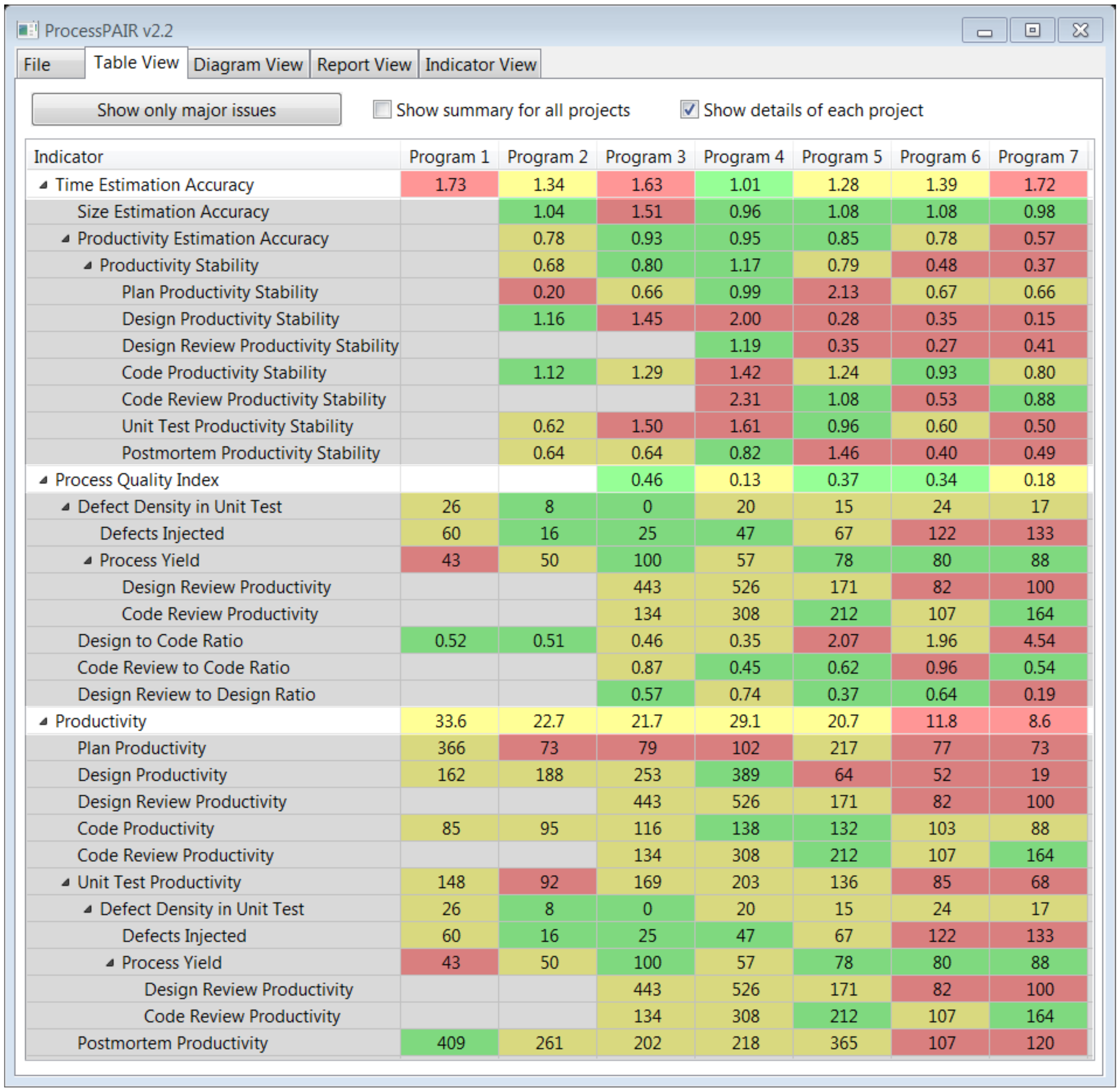
After selecting the PM, the type of input file, and the input file to analyze, the tool performs the analysis and makes the results of the analysis available in multiple views.

Table View. This view presents the detailed evaluation of all PIs for all projects under analysis, as depicted in Fig. 4. Initially, it shows only the 3 top-level PIs (non-shaded lines in Fig. 4). 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, according to the performance ranges described in Section 4. Cells with missing data are left blank. As indicated by the red cells in Fig. 4, the main top-level performance problems occur in time estimation (projects P1, P3 and P7) and in productivity (projects P6 and P7). By expanding the nodes in this view, one can drill down to lower level PIs, following the hierarchical structure of the PM, in order to identify potential causes of performance problems. For example, the red colored cells in Fig. 4 suggest that the time estimation problems in P3 and P7 have different causes: a size estimation problem in P3, and a productivity estimation problem in P7. As another example, the red colored cells in Fig. 4 suggest that the poor productivity in P7 may be caused by poor productivity in the Plan, Design, Design Review, Unit Test and Postmortem phases. Information about the relative importance of those potential causes can be found by skipping to the "Diagram View", as depicted in Fig. 5.

Diagram View. The goal of this view it to help identifying and prioritizing, project by project, the causes of performance problems, so that subsequent manual root cause analysis and improvement actions can be properly focused. The child indicators are sorted according to the value of the ranking coefficient described in Section 5. As explained there, the ranking coefficient represents a cost-benefit estimate that relates the cost of improving the value of the child indicator with the benefit on the value of the parent indicator. To facilitate the analysis, child PIs with a ranking coefficient below some configurable threshold can be hidden, as illustrated in Fig. 5 for a 0.1 threshold. Such a threshold excludes child PIs which improvement by say 10% (in terms of reduction of the percentile distance to the optimal value) is estimated to lead to ≤1% improvement in the value of the parent PI. For example, the diagram of Fig. 5 suggests that the major cause for the poor productivity in project 7 is the poor productivity in the Design phase (with a ranking coefficient of 15.9), followed (by a large distance) by the poor productivity in the Plan, Unit Test, Design Review and Postmortem phases. Similarly, the diagram suggests that the major cause for the poor time estimation in project 7 is the productivity instability in the design phase, followed by the productivity instability in the unit test phase (with a much smaller ranking coefficient).

Indicator View. This view provides useful information to visually inspect the behavior of each PI, as compared to the benchmarks embodied in the PM, as illustrated in Fig. 6. The chart on the right suggests a trend for improvement of the Process Yield in the data under analysis. The chart on the left reveals a hybrid continuous-discrete distribution in the training data set, with roughly 10% of the data points in each extreme of the scale.

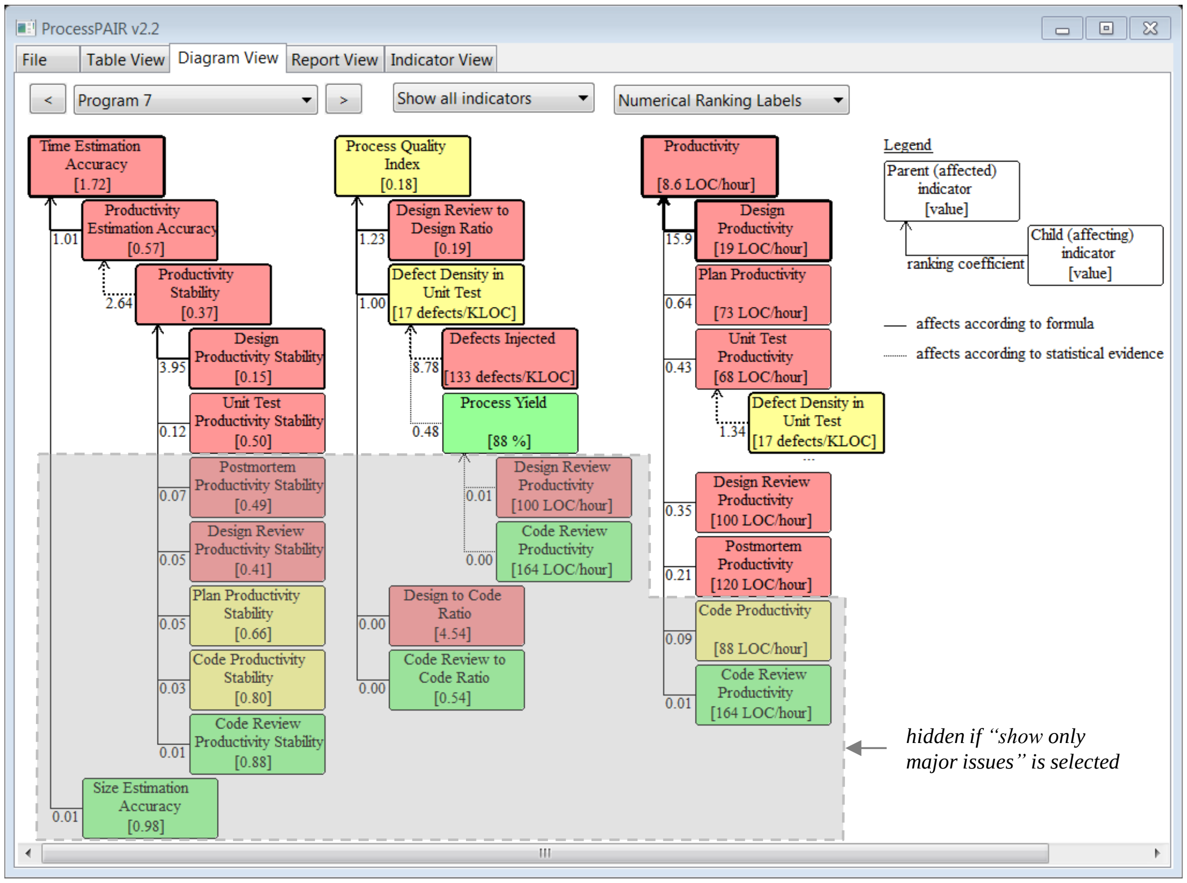
Report View. This view presents a textual description of the main performance problems encountered and potential causes (properly filtered and prioritized), aggregated or on a project by project basis. Due to space limitations we only present a synthesis in Table 4.



[ ] Show summary for all projects   [✓] Show details of each project

Indicator	Program 1	Program 2	Program 3	Program 4	Program 5	Program 6	Program 7
◢ Time Estimation Accuracy	1.73	1.34	1.63	1.01	1.28	1.39	1.72
Size Estimation Accuracy		1.04	1.51	0.96	1.08	1.08	0.98
◢ Productivity Estimation Accuracy		0.78	0.93	0.95	0.85	0.78	0.57
◢ Productivity Stability		0.68	0.80	1.17	0.79	0.48	0.37
Plan Productivity Stability		0.20	0.66	0.99	2.13	0.67	0.66
Design Productivity Stability		1.16	1.45	2.00	0.28	0.35	0.15
Design Review Productivity Stability				1.19	0.35	0.27	0.41
Code Productivity Stability		1.12	1.29	1.42	1.24	0.93	0.80
Code Review Productivity Stability				2.31	1.08	0.53	0.88
Unit Test Productivity Stability		0.62	1.50	1.61	0.96	0.60	0.50
Postmortem Productivity Stability		0.64	0.64	0.82	1.46	0.40	0.49
◢ Process Quality Index			0.46	0.13	0.37	0.34	0.18
◢ Defect Density in Unit Test	26	8	0	20	15	24	17
Defects Injected	60	16	25	47	67	122	133
◢ Process Yield	43	50	100	57	78	80	88
Design Review Productivity			443	526	171	82	100
Code Review Productivity			134	308	212	107	164
Design to Code Ratio	0.52	0.51	0.46	0.35	2.07	1.96	4.54
Code Review to Code Ratio			0.87	0.45	0.62	0.96	0.54
Design Review to Design Ratio			0.57	0.74	0.37	0.64	0.19
◢ Productivity	33.6	22.7	21.7	29.1	20.7	11.8	8.6
Plan Productivity	366	73	79	102	217	77	73
Design Productivity	162	188	253	389	64	52	19
Design Review Productivity			443	526	171	82	100
Code Productivity	85	95	116	138	132	103	88
Code Review Productivity			134	308	212	107	164
◢ Unit Test Productivity	148	92	169	203	136	85	68
◢ Defect Density in Unit Test	26	8	0	20	15	24	17
Defects Injected	60	16	25	47	67	122	133
◢ Process Yield	43	50	100	57	78	80	88
Design Review Productivity			443	526	171	82	100
Code Review Productivity			134	308	212	107	164
Postmortem Productivity	409	261	202	218	365	107	120

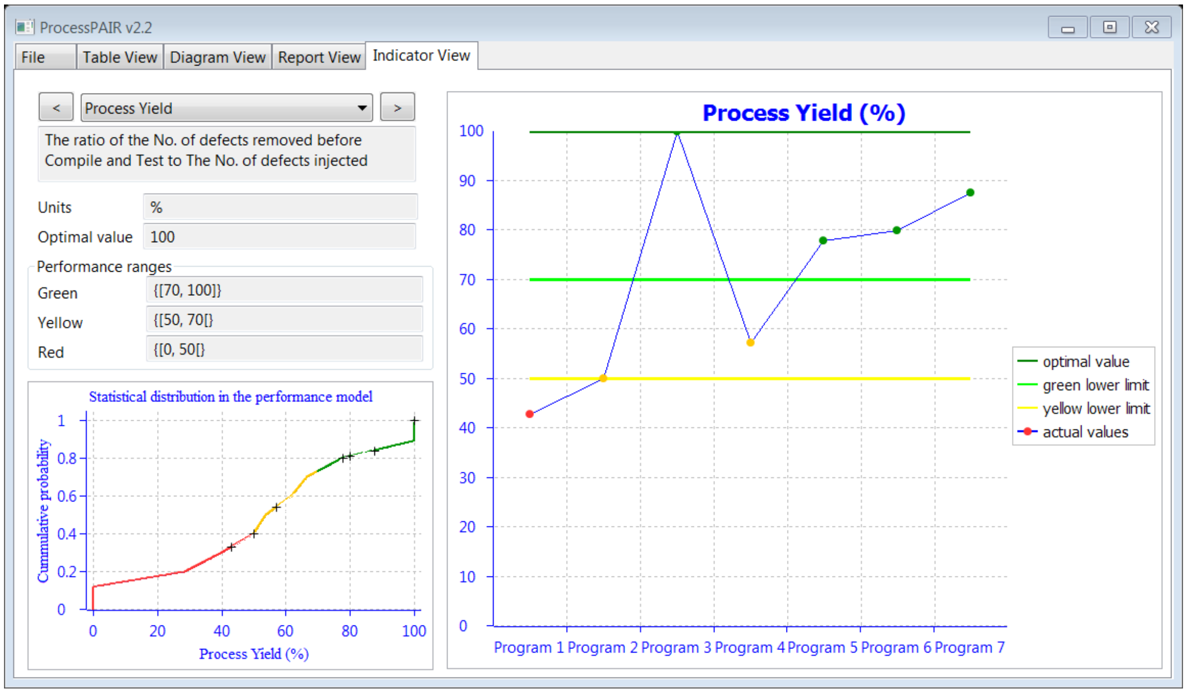
Figure 4 Evaluation of top-level and nested (shaded) PIs in the case study for projects 1 to 7.


ProcessPAIR v2.2

Program 7 | Show all indicators | Numerical Ranking Labels

File
Table View
Diagram View
Report View
Indicator View
Time Estimation Accuracy [1.72]
Productivity Estimation Accuracy [0.57]
1.01
Productivity Stability [0.37]
2.64
Design Productivity Stability [0.15]
3.95
Unit Test Productivity Stability [0.50]
0.12
Postmortem Productivity Stability [0.49]
0.07
Design Review Productivity Stability [0.41]
0.05
Plan Productivity Stability [0.66]
0.05
Code Productivity Stability [0.80]
0.03
Code Review Productivity Stability [0.88]
0.01
Size Estimation Accuracy [0.98]
0.01
Process Quality Index [0.18]
Design Review to Design Ratio [0.19]
1.23
Defect Density in Unit Test [17 defects/KLOC]
1.00
Defects Injected [133 defects/KLOC]
8.78
Process Yield [88 %]
0.48
Design Review Productivity [100 LOC/hour]
0.01
Code Review Productivity [164 LOC/hour]
0.00
Design to Code Ratio [4.54]
0.00
Code Review to Code Ratio [0.54]
0.00
Productivity [8.6 LOC/hour]
Design Productivity [19 LOC/hour]
15.9
Plan Productivity [73 LOC/hour]
0.64
Unit Test Productivity [68 LOC/hour]
0.43
Defect Density in Unit Test [17 defects/KLOC]
1.34
Design Review Productivity [100 LOC/hour]
0.35
Postmortem Productivity [120 LOC/hour]
0.21
Code Productivity [88 LOC/hour]
0.09
Code Review Productivity [164 LOC/hour]
0.01
Legend
Parent (affected) indicator [value]
ranking coefficient
Child (affecting) indicator [value]
affects according to formula
affects according to statistical evidence
hidden if "show only major issues" is selected

Figure 5 Ranked potential causes of performance problems in project 7 of the case study.


ProcessPAIR v2.2

Process Yield

The ratio of the No. of defects removed before Compile and Test to The No. of defects injected

Units	%
Optimal value	100

Performance ranges

Green	[[70, 100]]
Yellow	[[50, 70[]
Red	[[0, 50[]

Figure 6 Example of analysis of an individual PI in the case study.

6.3 *Case study data analysis: comparison of manual and model-based analysis*

Table 4 compares the results of manual analysis (extracted from the Performance Analysis Report produced by the developer) and the results of model-based analysis (consolidated from the Report View produced by the tool), showing that similar conclusions are drawn. In this case, the manual analysis took 8 hours as reported by the developer, whilst the model-based analysis took just a few seconds. This allows us to answer positively to RQ1. However, the information in Table 4 also shows that the manual analysis can go deeper in causal analysis—in this case by attributing the productivity problems in the design phase to long design documents. Hence the main advantage of the model-based analysis is that it can point out problematic areas to focus in subsequent manual analysis, making it more efficient and effective.

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 inefficient and unstable design process.	Significant time estimation problems in 3 projects (P1, P3, P7), possibly caused in P3 by a size estimation problem, and in P7 by productivity instability in several phases, notably in the design phase.
Product quality problems, with average defect density in unit test well above the recommended value of 5, caused by a high number of defects injected.	Potential process quality problems in 2 projects (P4 and P7), possibly caused in P7 by a high number of defects injected and a small design review to design ratio.
Productivity problems, namely at design phase, caused by an inefficient design 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 the design phase.

7. RELATED WORK

7.1 *Process performance models*

Our approach draws a strong inspiration from process performance models (PPMs). 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 [26].

In the case of continuous variables, a PPM often takes the form of a regression equation, relating controllable or uncontrollable factors (x) with outcomes (y), together with an indicator of the degree of variability in the model, such as the R^2 statistic. In the case of discrete variables, PPMs may be based on Bayesian networks [27].

PPMs are useful for project management and process management and improvement. In the latter case, PPMs help organizations identify and leverage important relationships among process factors and outcomes, and estimate the effects of alternative process changes.

The creation of a PPM usually involves the following steps, among others: (1) decide what outcomes to analyze; (2) hypothesize factors to investigate; (3) select the modeling techniques to use; (4) obtain relevant data; (5) fit the model to the data and evaluate the degree of fitness according to statistical and business criteria [27].

Examples of PPMs that can be constructed and applied in the context of the TSP/PSP, together with examples of outcomes and factors to consider for a few sub-processes, are described in [17]. An example of a PPM created by a TSP team for establishing a target code review rate (number of lines of code reviewed per hour), based on the predicted impact on the code review yield (percentage of defects found in code reviews), is as follows [17]:

- Regression equation: Code Review Yield = 146 – 0.364 * Code Review Rate
- $R^2 = 94.1\%$, p-value = 0.000

Possible factors to consider for analyzing the code review yield are factors currently collected by TSP (requirements inspections rate, high-level design inspections rate, detailed design review rate, detailed design inspection rate, code review rate, code review/coding time) and factors requiring further data collection (code complexity, encapsulation, program language & tools, code review checklist, coding skills and experiences with the languages and tools used, code review skills and experiences, quality of reused source code) [17].

Our work is strongly influenced by the concept of PPMs, with several similarities and differences. As for the similarities, we applied the steps indicated above for the derivation and validation of several components of our model: (1) selection of top-level PIs (outcomes); (2) selection of child PIs (factors); (3) selection of exact equations or regression equations relating parent and child PIs; (4) obtention of a large PSP data set with historical data; (5) computation of regression equations and correlation coefficients, in the case of PIs not related by exact equations. Our work is novel, because it was not done before for the PSP in a comprehensive way. We focused our attention on factors for which historical data is available for model validation and calibration. We privileged factors that are related by exact equations with the outcomes under analysis, to minimize context sensitivity. Regarding the differences, there is a small difference in terms of vocabulary, because we use the term "model" to refer to the network of related PIs, together with additional attributes for the nodes (approximate statistical distribution, recommended ranges) and edges (sensitivity coefficients, correlation coefficients), whilst in the PPM nomenclature a model refers to a single outcome and a set of factors. The main difference is that our model 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 the exact or regression equations).

7.2 Benchmark-based software evaluation

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 [23][28] to rate the maintainability of software products, with adaptations for process evaluation instead of product evaluation.

In [23][28], the authors claim that the effective use of software metrics is hindered by the lack of meaningful thresholds. They also note that thresholds have been proposed for a few metrics only, mostly based on expert opinion and a small number of observations, or systematically derived based on unjustified assumptions about the statistical properties of the metrics (such as normality). Hence, they propose a method to empirically derive in a systematic way metric thresholds from measurement data (benchmarks), in order to determine risk profiles and maintainability ratings for products under analysis. They propose a discrete rating schema (from 1 to 5 stars), based on thresholds that correspond to the 20%, 40%, 60%

and 80% quantiles. In our case, we use a similar schema, with 3 instead of 5 intervals, to assess process and product metrics.

7.3 Defect causal analysis

Many process improvement approaches (e.g., Six Sigma [29] or FMEA [30]) described in literature and practiced in industry include causal analysis activities for determining the causes of defects and other problems [31]. However, most of the techniques are essentially manual. Defect Causal Analysis [32] is one of the prominent methods for analyzing defects and identifying root causes for improvement in software engineering. Furthermore, the learning capability of DCA from defects enable improvement of processes and products, which is a significant benefit in the context of continuous improvement strategies [33].

The DCA process involves 6 steps to be performed in DCA workshops [32]: (1) select problem sample; (2) classify selected defects (e.g., using ODC); (3) identify systematic errors (e.g., with Pareto charts); (4) determine main causes (e.g., using Fishbone or cause-effect diagrams); (5) develop action proposals; (6) document meeting results.

The DCA approach is essentially complementary to our approach. The main advantage of our performance analysis 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. The DCA technique becomes particularly useful when problems associated with high defect density are identified by our approach.

8. LIMITATIONS AND THREATS TO VALIDITY

In the case study presented, using real world data, the conclusions obtained by the model-based analysis are very close to the ones obtained by the developer in his manual analysis. This suggests that our approach can be helpful in performance analysis and process improvement, by pointing out the areas to focus in manual analysis. However, further experiments need to be conducted to quantify the benefits that can be achieved with our approach.

Some choices that we made in the model construction and calibration process and in the definition of the performance analysis and ranking method may have to be adjusted when applying our approach in varying contexts, as discussed next.

Our choice of PIs was constrained by the data available, so relevant factors may have not been captured by those PIs, such as application domain, expertise, development technology, among others. However, in case data about those factors is available, our approach can be easily extended to take them into account.

The omission of a relevant factor Z is particularly misleading when Z acts as confounding factor, that is, when Z affects variables X and Y present in the model in such a way that they erroneously seem inter-dependent. We tried to avoid confounding factors by focusing whenever possible on dependencies that are determined by formulas, and by backing up by relevant references and studies from the literature other types of dependencies.

Although the data set used for model validation and calibration refers to a homogeneous set of projects and processes, it contains data from a broad set of contexts (regarding user expertise, programming language, development environment, etc.), which may diminish the precision of the conclusions that can be drawn from the model derived from that data set. However, that is not an inherent deficiency of our approach, but rather a limitation of the data available for modeling and analysis.

Our ranking method uses a novel cost-benefit heuristic to prioritize the factors to address in subsequent improvement actions. Although we successfully experimented the heuristic in several cases, since it uses very limited data (statistical distribution and sensitivity coefficients), it is possible that relevant cost or benefit drivers are not captured by the heuristic.

In the range calibration process, we tried to combine recommended values from literature with thresholds derived from the historical data, reason why the calibration process is not fully automatic. This complicates the adaptation of our approach for other data sets. In the future, we intend to investigate fully automated calibration processes.

We used LOC/hour for measuring productivity, in spite of its known limitations, because it is the metric available in the data set. Although the usage of LOC/hour for phase ranking purposes is not so much problematic (it reduces to effort comparisons), false positives or false negatives may occur in the identification of overall productivity performance problems.

Although our approach and tool are general and can be instantiated for any process, the model presented in this paper is applicable only to analyze PSP performance data. We intend to replicate our approach to other processes without such a well-defined measurement framework as the PSP, so we expect to encounter difficulties regarding data availability, data quality, and standardization.

9. CONCLUSIONS AND FUTURE WORK

We presented a performance model to enable the automated identification and ranking of performance problems and their root causes, and reduce the manual effort of performance analysis in the PSP. The performance analysis approach presented in this paper is currently automated in our ProcessPAIR (Process Performance Analysis and Improvement Recommendation) tool. The tool analyzes performance data produced by PSP developers in their projects, and pinpoints performance problems and a ranked list of possible root causes.

As future work we plan to build a comprehensive catalogue of improvement actions to recommend for the highest-ranked causes, build similar models for analyzing performance data produced in the context of other processes, and conduct further experiments to assess the benefits of our approach.

ACKNOWLEDGEMENTS

The authors would like to thank the SEI for giving access to the PSP data for performing this study and the anonymous reviewers for their helpful comments and suggestions. The works of J. Faria and M. Raza are partly financed by the FCT – *Fundação para a Ciência e a Tecnologia* (Portuguese Foundation for Science and Technology) within project UID/EEA/50014/2013 and research grant SFRH/BD/85174/2012, respectively.

REFERENCES

[1] Bohem, B. 2011. Some Future Software Engineering Opportunities and Challenges. In *The Future of Software Engineering*, Springer-Verlag, 1-32.

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

[3] Humphrey, W. and Over, J. 2011. Leadership, Teamwork, and Trust: Building a Competitive Software Capability. Addison-Wesley Professional

[4] Rombach, D., Münch, J., Ocampo, A., Humphrey, W., and Burton, B. 2008. Teaching disciplined software development. *Journal of Systems and Software* 81(5): 2008, 747-763.

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

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

[7] 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.

[8] 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.

[9] Nasir, M. and Yusof, A. 2005. Automating a Modified Personal Software Process. *Malaysian Journal of Computer Science*, vol. 18, 11–27.

[10] Kemerer, C., and Paulk, M. 2009. The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data. *IEEE Trans. on Soft. Eng.*, vol. 35, Issue 4, 534-550.

[11] Shen, W., Hsueh, N., Lee, W. 2011. Assessing PSP effect in training disciplined software development: A Plan–Track–Review model. *Inform. and Software Technology* 53, 137–148.

[12] Duarte, C., Faria, J., and Raza, M. 2012. PSP PAIR: Automated Personal Software Process Performance Analysis and Improvement Recommendation. In *Proc. of the 8th Int. Conf. on the Quality of Information and Communications Technology*, IEEE CPS, Lisbon, Portugal.

[13] Raza, M., Faria, J., Henriques, P., and Nichols, W. 2013. Factors Affecting Productivity Performance in PSP Training. In *TSP Symposium 2013*, CMU/SEI-2013-SR-022, 35-45.

[14] Raza, M., Faria, J. 2014. A Model for Analyzing Estimation, Productivity and Quality Performance in the Personal Software Process. In *Proc. of the 2014 Int. Conf. on Software and System Process (ICSSP 2014)*, ACM, 10-19.

[15] Jones, C. 2000. Software Assessments, Benchmarks, and Best Practices. Addison Wesley.

[16] Humphrey, W. 2009. *The Software Quality Profile*. White Paper, SEI.

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

[18] Wagner, S. and Ruhe, M. 2008. A Systematic Review of Productivity Factors in Software Development. In *Proc. of 2nd Int. Workshop on Software Productivity Analysis and Cost Estimation*.

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

[20] Card, D. 2006. The Challenge of Productivity Measurement. In *Proc. of the Pacific Northwest Software Quality Conference*, Portland, OR.

[21] Jones, C. 2010. Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies. McGraw-Hill.

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

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

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

[25] Hamby, D.M. 1994. A Review of Techniques for Parameter Sensitivity Analysis of Environmental Models. Environmental Monitoring and Assessment, Vol. 32, Issue 2, 135-154, Springer.

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

[27] Zubrow, D., Stoddard, B., Young, R., Schaaf, K. 2009. A Practical Approach for Building CMMI Process Performance Models. SEPG North America Conference, 2009.

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

[29] Kwak, Y.H., Anbari, F.T. 2006. Benefits, obstacles, and future of six sigma approach. Technovation 26(5), 708–715.

[30] Campos, J. 2012. Risk management and failure mode and effect analysis for product development. Rapid Innovation LLC.

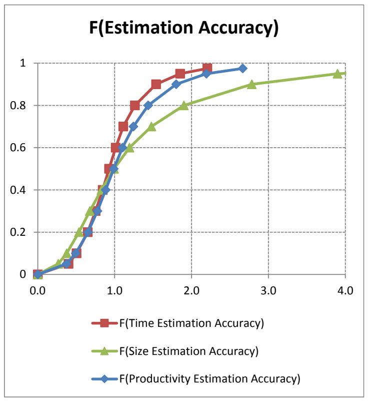
[31] Kalinowski, M., Card, D.N., Travassos, G.H. 2012. Evidence-based guidelines to defect causal analysis. IEEE Software 29(4), 16–18.

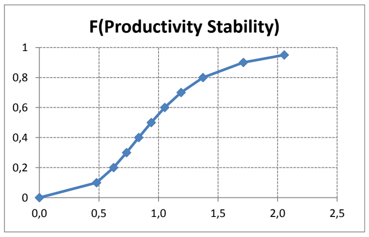
[32] Card, D.N. 1993. Defect-causal analysis drives down error rates. IEEE Software 10(4), 98–99

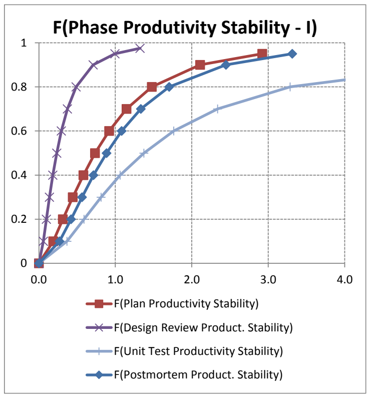
[33] Card, D.N. 2005. Defect Analysis: Basic Techniques for Management and Learning. *Advances in Computers*, *65*, 259-295

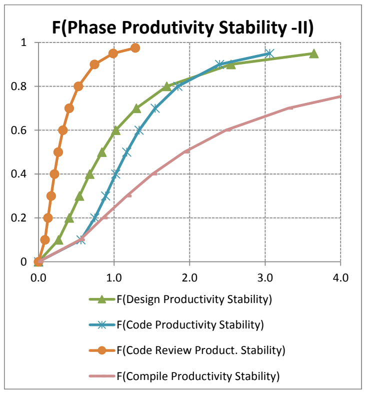
APPENDIX A – APPROXIMATE CUMULATIVE DISTRIBUTION FUNCTIONS

Predictability related PIs

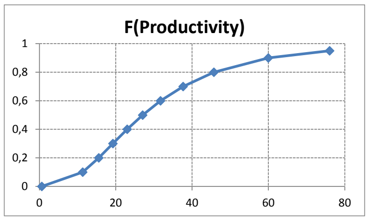


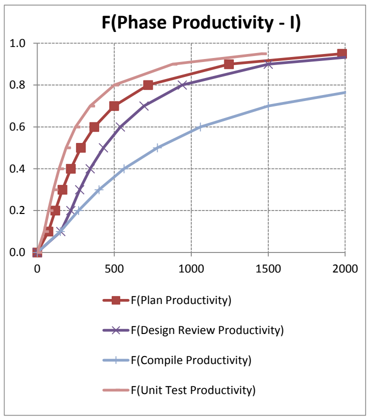


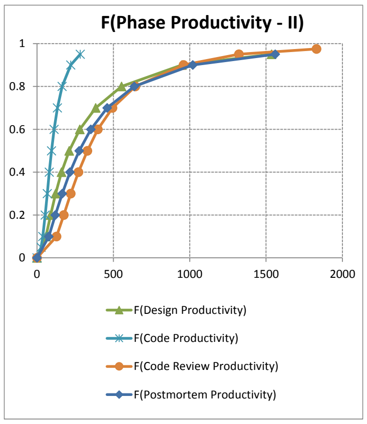




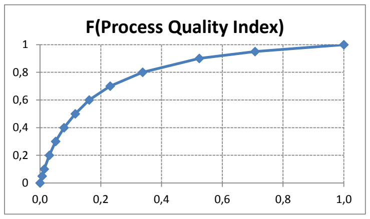
Productivity related PIs

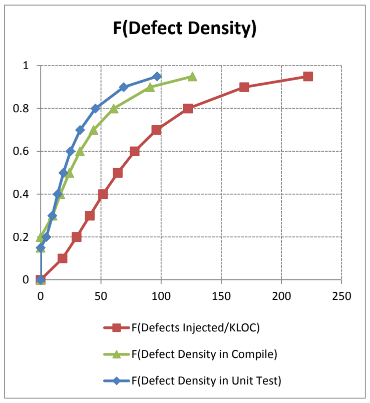


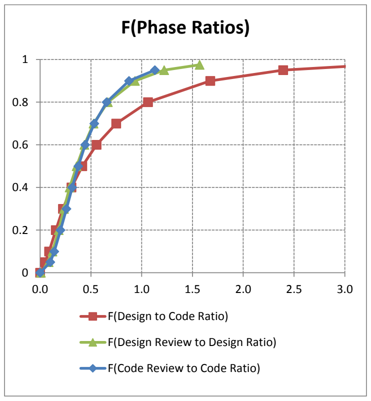




Quality related PIs







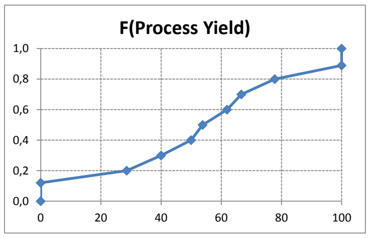


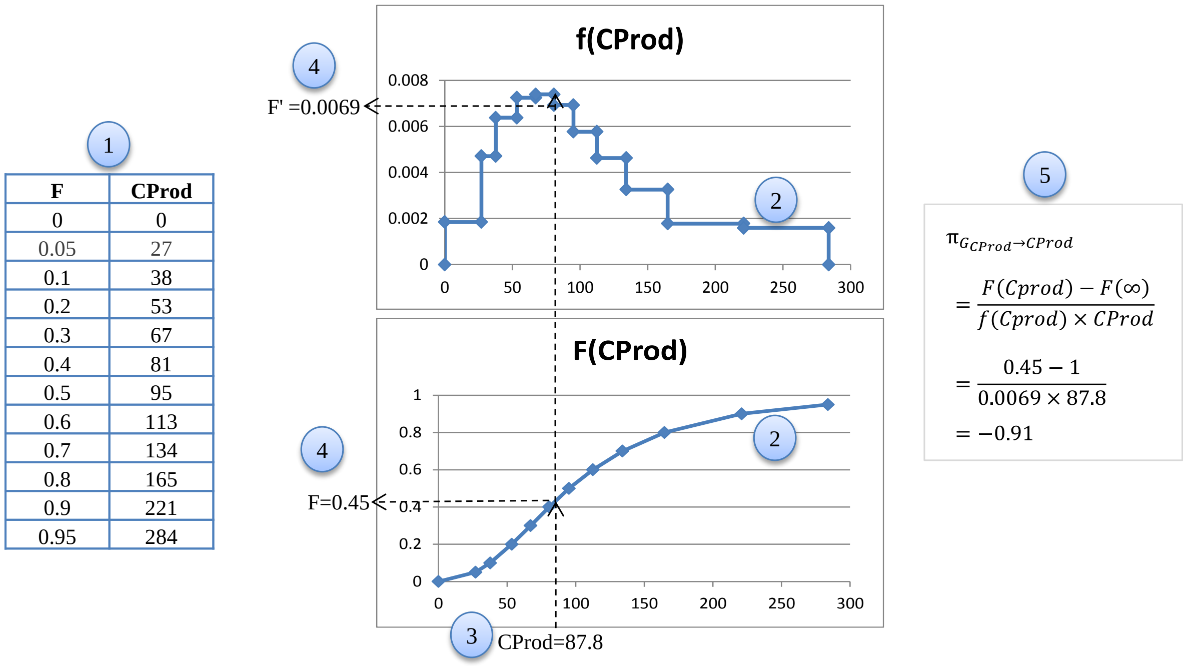
Figure 7 Approximate cumulative distribution functions for all PIs in our model derived from the PSP data set.

APPENDIX B – EXAMPLE OF RANKING CALCULATIONS

Table 5 presents the calculations performed to rank the factors that affect the overall productivity in "Program 7" of the case study. Figure 8 illustrates how a percentile coefficient is computed from the historical data. Regarding the sensitivity coefficient, the phases that consume more effort (i.e., with lower productivity) – Design, Unit Test, Plan and Code – are ranked at the top 4 positions. However, the productivity in Code is significantly closer to the optimal value (in terms of percentiles) than for the other phases; so, when computing the combined ranking coefficient, the productivity in Code goes down to the 6th position. In the final ranking, the top two phases which productivity should be improved (to improve the overall productivity with the best cost-benefit ratio) are the Design and Plan phases.

Table 5. Example ranking calculations for the factors that affect the overall productivity.

i	Variable	Value (LOC/ hour)	Percentile (F_i)	Probability Density (F'_i)	Percentile Coef. (π_i) $\frac{F_i(x)-F_i(o_i)}{xF'_i(x)}$	Sensitivity Coef. (σ_i) $\frac{Prod}{Prod_k}$	Ranking Coef. (ρ_i) $\pi_i \times \sigma_i$
0	Productivity	8.63	0.07	0.00936			
1	Plan Productivity	73.5	0.10	0.00223	5.48 (2nd)	0.118 (3rd)	0.64 (2nd)
2	Design Productivity	19.4	0.02	0.00142	35.65 (1st)	0.446 (1st)	15.9 (1st)
3	Design Review Prod.	100.0	0.07	0.00066	4.12 (3rd)	0.086 (5th)	0.35 (4th)
4	Code Productivity	87.8	0.45	0.00693	0.91 (6th)	0.098 (4th)	0.09 (6th)
5	Code Review Product.	163.6	0.18	0.00211	0.23 (7th)	0.053 (7th)	0.01 (7th)
7	Unit Test Prod.	67.9	0.18	0.00353	3.42 (4th)	0.127 (2nd)	0.43 (3rd)
8	Postmortem Product.	120.0	0.20	0.00220	2.86 (5th)	0.072 (6th)	0.21 (5th)



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

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

Figure 8 Computing the percentile coefficient for Code Productivity based on percentiles extracted from the historical data.