ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.6, June 2009, pp.1499−1510
doi: 10.3724/SP.J.1001.2009.00564

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于数据包络分析的软件任务性能基准评价*

阮 利[1,2,4+], 王永吉[1,3], 王 青[1], 曾海涛[1,2]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100190)

[2](中国科学院 研究生院,北京 100049)

[3](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100190)

[4](北京航空航天大学 计算机学院,北京 100191)

Benchmarking Software Task Performance Based on Data Envelopment Analysis

RUAN Li[1,2,4+], WANG Yong-Ji[1,3], WANG Qing[1], ZENG Hai-Tao[1,2]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

[3](State Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[4](School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

+ Corresponding author: E-mail: ruanli@itechs.iscas.ac.cn, http://itechs.iscas.ac.cn

Ruan L, Wang YJ, Wang Q, Zeng HT. Benchmarking software task performance based on data envelopment analysis. *Journal of Software*, 2009,20(6):1499−1510. http://www.jos.org.cn/1000-9825/564.htm

Abstract: In this paper, a software task performance benchmarking method based on Data Envelopment Analysis (DEA)-TaskBeD is proposed. TaskBeD's fundamental benchmarking model and algorithms (transforming undesirable outputs, identifying high performance tasks, establishing reference sets and performing sensitivity analysis) are introduced. Experimental results show that the proposed TaskBeD method achieves a good result of dealing with multivariate and VRS.

Key words: software project; software process; task performance benchmarking; data envelopment analysis

摘 要: 提出了一种基于数据包络分析的软件任务性能基准评价新方法——TaskBeD.介绍了 TaskBeD 的任务基准评价模型和核心算法(挖掘高性能的软件任务,建立参考任务集和结果的敏感度分析).实验结果显示,TaskBeD 能够高效处理多变元和可变规模收益任务数据.

关键词: 软件项目;软件过程;任务性能基准评价;数据包络分析

中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant Nos.60573082, 60473060, 60673121 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2006AA01Z185, 2005BA113A01, 2006AA01Z182, 2007AA010303, 2007AA01A127, 2007AA01Z186 (国家高技术研究发展计划(863))

Received 2007-12-28; Accepted 2008-06-11

1 Introduction

Software (development) task is an atomic unit of project and a key performance indicator for developers. With the wide application of CMMI/TSP/PSP[1,2], fine-grained software task benchmarking, which is a critical step in software task management, gains increasing research focus in software process improvement field[2]. Benchmarking in this context is the process of determining which task is the very best, which task sets the standard, and what that standard is from software task repositories.

For quantitative software task management, it is really a difficult issue to correctly benchmark software tasks. Because tasks tend to have multivariate inputs (effort, etc.) and outputs (program size, defects, document, etc.), and the relationship between those inputs and outputs is usually nonlinear[3,4], the major challenge for benchmarking is how to effectively cope with multivariate and variable return to scale (VRS)[3].

In this paper, we propose a novel software task benchmarking method based on Data Envelopment Analysis (DEA)-TaskBeD, which can deal with the multivariate, VRS and undesirable outputs problems. TaskBeD can be regarded as a further extension of the work in Refs.[3,5] and our previous work in Ref.[6] by scaling the DEA-based software project benchmarking down to fine-grained software tasks and introducing mechanisms of transforming undesirable outputs and sensitivity analysis.

This paper is organized as follows. Section 2 presents the related work. Section 3 introduces TaskBeD's CCR and BCC task benchmarking models. Section 4 presents TaskBeD's fundamental benchmarking algorithms of transforming undesirable outputs, identifying high performance tasks, establishing reference sets and sensitivity analysis. To verify task benchmarking models and algorithms, an experiment on real task application datasets is demonstrated and its results are analyzed in Section 5. Section 6 introduces a sensitivity analysis process to guarantee the effectiveness of experiments. Section 7 summarizes our conclusions and points out the future work.

2 Related Work

Software process standards CMMI/TSP/PSP[1,2] classify software process management into three levels: organization level, team level and individual level. Moreover, according to CMMI that focuses on the most macro view of software process, software project is the key performance benchmarking components to measure the organizational performance. According to PSP that focuses on the most micro view of software process, software task is the key performance benchmarking components to measure personal performance. To date, in software process improvement field, most current benchmarking research mainly focuses on projects[3,4,7–9]. Using COTS projects as an example, it is analyzed by Ref.[7] that "there are several groups interested in intraorganizational productivity benchmarking of software projects. Customers of COTS software projects within the Enterprise Resource Planning (ERP) market (e.g. products like SAP and Oracle) demand that productivity benchmarks are included in proposals. Therefore, COTS project bidders must provide benchmarks to stay competitive. Organizations use benchmarks internally to evaluate projects. Project managers and methodologists need benchmarks to identify best practice processes and technologies. The need is clear".

On the other hand, due to the research of PSP, the recent trend of software process improvement is "scaled down" to the level of individual developers. Compared with the benefit of benchmarking software projects on macro process management when applying CMMI, task performance benchmarking is vital for any developer seeking to continuously improve his task management practices and identify competitive strengths and weaknesses when applying PSP. However, although there have been deep research [3,4,7–9] on benchmarking macro software projects, the benchmarking of fine-grained software tasks, which is one key step to achieving quantitative micro personal

software process management and realizing total (including both macro level and micro level) process management, is ignored. Moreover, existing task management literature has proposed few task benchmarking methods that explicitly consider their multivariate and VRS constraints. Commonly applied performance evaluation methods, such as earned value management (EVM)[10], provide organizations with a method of systematically comparing actual performance to task goals. It does not take into account of the task's uniqueness when performing cross-task evaluation[8]. Statistical methods[11] are proposed to compare the task performance with some theoretical optimal ones (e.g. theoretical baselines)[4]. However, as Ref.[3] recently reports, in software engineering, it seems more sensible to compare the performance with the best practice rather than with some theoretical optimal (and probably non-attainable) performance. Furthermore, multivariate regression[11] is unsuitable for identifying the best tasks because it tends to evaluate tasks relative to the average rather than to the best[3]. Moreover, software tasks data are heteroscedastic. We could therefore wrongly tend to identify mainly the large tasks as the most productive without taking into account of the VRS constraints of tasks[3]. In a word, to the best of our knowledge, few research results on benchmarking software tasks under multivariate and VRS constraints have been publicly reported.

Data Envelopment Analysis (DEA) developed by Charnes and Cooper in 1978 is a linear non-parametric programming-based performance evaluation technology[4,12] and has gained successful benchmarking applications in financial, industrial process, etc. Recently, DEA is gaining increasing benchmarking research interests in software process field[3,5,8] after Stensrud, *et al*. first introduced it into software project benchmarking in 1999[3,4]. DEA gains interests mainly because it provides a powerful unique advantage of handling task uniqueness[8], multivariate and VRS[3]. Ref.[3] especially points out that "DEA is the only method complying with the two requirements (multivariate inputs/outputs and VRS) that we consider crucial to perform correct performance assessment in software engineering". DEA is appealing to software practitioners because it uses the best practice frontier as a benchmark rather than some theoretical baseline (and probably non-attainable)[3,5]. Although Refs.[3–5] introduced DEA into project and personal software process evaluation, they do not take task's fine-grained uniqueness into account. Moreover, existing DEA-based benchmarking methods in software process field[3–5] usually assume that the output of every task is an ideal output, i.e., there are no undesirable output in the output set. Thus, they[3,5] do not provide effective mechanisms to transform undesirable outputs that should not be ignored in practical applications. This paper extends Refs.[3–6] by scaling the DEA-based software project benchmarking down to micro software tasks and introducing mechanisms of transforming undesirable outputs and sensitivity analysis.

3 TaskBeD's Task Benchmarking Model Description

In this section, we first introduce TaskBeD's benchmarking model. Let n denote the number of software tasks to be benchmarked. Each task has m inputs (reflect the "resource" consumption in the software process, e.g. time) and s outputs (reflect the performance after consuming "resources", e.g. software products).

Definition 1 (A Task Set (T)). A task set is defined as $T=\{t_1,\ldots,t_j,\ldots,t_n\}$. The basic requirement is that the n tasks are homogeneous which can be efficiently evaluated on their relative performance.

Definition 2 (A Task (t_j)). Each task t_j is defined as a tuple $t_j=(X_j,Y_j)$ where X_j denotes the m inputs of t_j and Y_j denotes the s outputs of t_j.

Definition 3 (A Task Input (X_j)). Each task t_j's input is defined as $X_j=(x_{1j},x_{2j},\ldots,x_{ij},\ldots,x_{mj})$ where x_{ij} denotes the amount of input i utilized by t_j and $x_{ij}\geq 0$.

Definition 4 (A Task Output (Y_j)). The task t_j's output is defined as $Y_j=(y_{1j},y_{2j},\ldots,y_{rj},\ldots,y_{sj})$ where y_{rj} denotes the amount of output r produced by t_j and $y_{rj}\geq 0$.

The basic understanding of inputs and outputs of software tasks is that the less inputs the better and the larger

outputs the better in the view of performance. Based on the two classical DEA models (the CCR model by Charnes, Cooper and Rhodes[13] and the BCC model by Banker, Charnes and Cooper[14]), we establish TaskBeD's task performance benchmarking models (see Table 1) for any task $t_u(t_u \in T)$. Let the optimal value of task benchmarking models be θ^u, $\{\lambda_j^u\}(j\in[1,n])$, $\{s_i^{+u}\}(i\in[1,m])$, $\{s_k^{-u}\}(k\in[1,s])$ for each task $t_u(t_u \in T)$. θ^u represents the nonnegative performance score of the task t_u. $\theta^u \in [0,1]$. s_i^{+u} represents nonnegative slack (i.e., excess resources) associated with the inputs X_i^u, and s_k^{-u} represents nonnegative slack (i.e., additional output) associated with the outputs Y_k^u. s_k^{-u} is the additional amount of the kth output that is expected, and s_i^{+u} is the amount by which the ith input has to be reduced, if the t_u was to become efficient. $\varepsilon(\varepsilon>0)$ represents a non-Archimedean constant and is smaller than any positive valued real number.

Table 1 TaskBeD—Task benchmarking models

(1) CCR model	(2) BCC model
$\min\theta-\varepsilon(\sum S_i^+ + \sum S_k^-)$	$\min\theta-\varepsilon(\sum S_i^+ + \sum S_k^-)$
s.t. $\theta X_u - \sum x_{ij}\lambda_j - s_i^+ = 0$, $i=1,...,m$	s.t. $\theta X_u - \sum x_{ij}\lambda_j - s_i^+ = 0$, $i=1,...,m$
$\sum x_{ij}\lambda_j - Y_u - s_k^- = 0$, $k=1,...,s$	$\sum x_{ij}\lambda_j - Y_u - s_k^- = 0$, $k=1,...,s$
$\theta\in[0,1]$	$\sum \lambda_j = 1$
$\lambda_j\geq 0$, $j=1,\ldots,n$	$\theta\in[0,1]$
$s_i^+\geq 0$	$\lambda_j\geq 0$, $j=1,\ldots,n$
$s_k^-\geq 0$	$s_i^+\geq 0$
$u\in[1,n]$	$s_k^-\geq 0$
	$u\in[1,n]$

The CCR task benchmarking model's assumption is Constant Return to Scale (CRS) and the BCC model's assumption is Variable Return to Scale (VRS) (see Fig.1).

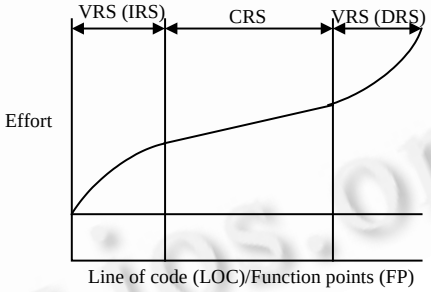


Fig.1 CRS and VRS models

CRS assumes a linear relationship between inputs X_u and outputs Y_u[3] that is consistent with the productivity model $p=y/x$(p=*productivity*, x=*effort*, y=*FP* or *SLOC*). VRS assumes a nonlinear relationship between X_u and Y_u that is consistent with cost estimation models (like COCOMO) $x=(1/p)y^B$ (p=*productivity*, x=*effort*, y=*FP* or SLOC, B>1). DRS (decreasing returns to scale) and IRS (increasing returns to scale) are two special cases of VRS. As CRS indicates the linear relationship between X_u and Y_u, IRS (DRS) indicates that an increase in one unit's inputs X_u will yield a greater (or less) proportionate increase of its outputs Y_u.

4 TaskBeD's Task Benchmarking Algorithms

Commonly applied benchmarking process[4,7] at least has the following key steps: Handling the undesirable outputs to pre-handle the task data, identifying relatively high performance tasks to set up the task benchmark,

establishing reference sets to determine the task performance improvement gaps and performing sensitivity analysis to guarantee the effectiveness. In this section, we introduce four algorithms in TaskBeD for the four steps.

4.1 Undesirable outputs transforming mechanism

First, we give the following definitions.

Definition 5 (The Task's Undesirable Input). For the i_{th} input $x_{ij}\in X_j$, if an increase in x_{ij} does not contribute to the increase of any output $y_{ij}\in Y_j$, x_{ij} is defined as an undesirable input to Y_{ij}.

Definition 6 (The Task's Undesirable Output). For the r_{th} output $y_{rj}\in Y_j$, if an increase in any input x_{rj} does not contribute to the increase of the output y_{rj}, y_{rj} is defined as an undesirable output to x_{rj}. e.g., *defects* can be regarded as the task's undesirable outputs when *size* is the input because developers usually do not desire an increase in *size* contributes to an increase in *defects*.

In TaskBeD, we employ a $[TR\beta]$[15] transformation method (see Algorithm 1) to transform the undesirable outputs. In $[TR\beta]$ transformation method, the undesirable output (y_{rj}) is subtracted from a significantly large scalar (β_r)(r=1,...,s), such that all resulting (transformed) values ($y_{rj}=\beta_r-y_{rj}$) are positive and increasing values are desirable. TaskBeD's undesirable outputs transformation algorithm is presented in Algorithm 1. The rule to choose β_r is that β_r is generally a value just slightly larger than the maximum value of the undesirable output in the data set, since choosing a β_r value that is much greater than this maximum value can distort model results.

Algorithm 1. Transforming undesirable outputs.

Input: The Task Output (Y_j)(j=1,...,n);

Output: The transformed Output.

1 **for** r=1 to s **do**
2 **if** $(y_{r1},...,y_{rn})\in Y_j$ and each $y_{rj}\in(y_{r1},...,y_{rn})$ is an undesirable output **then**
3 /*select the maximum value $y_r^{\max}$ */
4 $y_r^{\max}=\max(y_{r1},...,y_{rn})$;
5 /*get a random value $\xi_r\in(0, y_r^{\max}/6)$*/
6 $\xi_r=randomize(0, y_r^{\max}/6)$;
7 /*get a β_r which is slightly larger than $y_r^{\max}$ by ξ_r*/
8 $\beta_r= y_r^{\max}+\xi_r$;
9 **for** each $y_{rj}\in\{y_{r1},...,y_{rn}\}$ **do**
10 /*subtract each undesirable output from β_r*/
11 $y_{rj}=\beta_r-y_{rj}$;
12 **end for**
13 **end if**
14 **end for**

4.2 Identifying the relatively high performance tasks

After the undesirable inputs/outputs have been pre-handled, the second step is to identify relatively high performance tasks. Only after the tasks of relatively high performance have been identified, can developers/project managers know which tasks perform well and should be learned best practices from. We now introduce TaskBeD's algorithm of identifying high performance tasks (see Algorithm 2). First, we give the following definitions.

Algorithm 2. Identifying relatively high performance tasks.

Input: The task set T;

Output: The task set with tasks of relatively high performance Ω.

1 /*Initialize Ω*/

2 Ω={};

3 /*Initialize ε*/

4 ε=0.00001;

5 **for all** $t_u \in T(u \in [1,n])$ **do**

6 /*calculate the performance score of t_u under T and TaskBeD's task benchmarking models*/

7 θ^{μ}=*caculatePerformanceScore*(t_u,T);

8 **if** (θ^{μ}=1) **then**

9 /*If t_u is of high performance then add t_u to Ω*/

10 $\Omega=\Omega\cup\{t_u\}$;

11 **end if**

12 **end for**

13 return Ω

Definition 7. If the optimal value of task benchmarking models θ^{μ}=1, we define t_u to be of relatively high performance, or relatively efficient in short.

Here, the word relatively means that the performance of t_u is a comparative measure based on the task set T[16].

Definition 8. If the optimal value θ^{μ}<1, we define t_u to be of relatively low performance.

That is, the task's outputs (see Definition 4) could be increased without increasing inputs (see Definition 3) or conversely inputs could be decreased without decreasing outputs[16].

Definition 9. The task set with tasks of relative high performance $\Omega=\{\Phi_1,...,\Phi_i,...,\Phi_m\}$. $\Phi_i \in T$ and $\Omega \subset T$. The optimal value θ^{i} of each Φ_i satisfies that θ^{i}=1.

Definition 10. The task set with tasks of relatively low performance is defined as $\Psi=T-\Omega$.

In Algorithm 2, we use the performance score θ^{μ} to distinguish between relatively high performance tasks and relatively low performance tasks, and establish a task set of relatively high performance tasks. After the relatively high performance tasks (Ω) have been identified using Algorithm 2, T will thus be clearly classified into relatively high performance (see Definition 7) and low performance ones (see Definition 8).

4.3 Establishing the reference sets

In practical applications, each identified relatively efficient task $\Phi_i(\Phi_i \in \Omega)$ is typical and usually of different improvement reference value for the inefficient one. Therefore, we must establish different reference sets for each relatively inefficient task. In this section, we present an algorithm of establishing reference sets for tasks of relatively low performance in TaskBeD (see Algorithm 3). First, we give the following definitions:

Definition 11. A task $t_u(t_u \in T)$'s reference set is defined as $RS_u=\{t_j: \theta^{\mu}=1$ and $\lambda_j^u \neq 0, j=1,...,n\}$.

Algorithm 3. Establishing a task's reference set.

Input: A task t_u and $T=\{t_1,t_2,...,t_n\}$;

Output: The t_u's reference set RS_u.

1 RS_u={};

2 ε=0.00001;

3 **for** (j=1 to n) **do**

4 /*calculate the λ_j^u under T and task benchmarking models*/

5 λ_j^u =*calculateLamda*(t_u,T);

6 **if** ($\lambda_j^u \neq 0$) **then**

7 $RS_u=RS_u\cup\{t_j\}$;

8 **end if**

9 **end for**

10 return RS_u

It should be noted that each task $t_j(t_j\in RS_u)$ is efficient, i.e., $\theta^u=1$ and $t_j\in\Omega$ (see Definition 7). For convenience, we call each efficient task t_j a ***peer*** of the task t_u. The corresponding λ_j^u (calculated from Table 1) of the peer t_j is thus called ***peer weight***. The *peer weight* λ_j^u indicates the improvement reference importance/value of the peer t_j to t_u. Via the reference set (RS_u), peers $\{t_j:t_j\in RS_u\}$ and the peer weights $\{\lambda_j^u\}$ of t_j derived from Algorithm 3, developers can determine which peer $\{t_j\}$ is of the biggest improvement reference value to the task t_u.

4.4 Sensitivity analysis mechanism

In this section, we present the sensitivity analysis mechanism (see Algorithm 4). As the kernel of TaskBeD is DEA which identifies best practice rather than the average or say the best 10 percent, it makes TaskBeD sensitive to extreme observations. It is therefore necessary to design a sensitivity analysis mechanism of outliers. There are several techniques (e.g., superefficiency and analysis of reference units) each of which have its strengths and limitations depending on the purpose of the DEA analysis[3]. In task performance benchmarking field, the most important purpose is twofold: first to identify best practice tasks as well as the reference tasks for individual tasks, and second, to determine the average efficiency of the software tasks to quantify the overall potential for performance improvement. For this, the simplest and probably most reasonable sensitivity analysis mechanism is to remove all the frontier tasks one by one and study the effect on the mean efficiency[3].

Algorithm 4. Sensitivity analysis.

Input: The task set $T=\{t_1,t_2,...,t_n\}$;

Output: Sensitivity.

1 **for all** t_u such that $t_u\in T$ **do**

2 $\theta^u=caculatePerformanceScore(t_u,T)$;

3 **end for**

4 /*calculate T's average performance scores*/

5 $avg^\theta=average(\theta^1,...,\theta^n)$;

6 /*initialize the number of outliers to be 0*/

7 $coutForOutlier$=0;

8 **for all** Φ such that $\Phi_i\in\Omega$ **do**

9 /*remove a high performance task from T*/

10 $T_i^{rev}=remove(\Phi_i,T)$;

11 **for all** t_k such that $t_k\in T_i^{rev}$ **do**

12 /*calculate performance score of t_k in T_i^{rev} */

13 $\theta_{ik}^{rev}=caculatePerformanceScore(t_k,T_i^{rev})$;

14 **end for**

15 /*calculate T_i^{rev}'s average performance score*/

16 $avg_i^\theta=average(\theta_{i1}^{rev},...,\theta_{i(n-1)}^{rev})$;

17 **if** ($|avg^\theta-avg_i^\theta|>0.001$) **then**

18 *coutForOutlier*++;
19 **end if**
20 **end for**
21 **if** *coutForOutlier*==0 **then**
22 return "*the sensitivity is reasonable*".
23 **else**
24 return "*the sensitivity is unreasonable*".
25 **end if**

5 Experiments

We have implemented our ideas about TaskBeD in our tool TaskBench which is a key task management component of software process management toolkit and was previously reported in Ref.[5,6] for the benchmarking of software task. TaskBench has implemented key benchmarking functions about how to identify which task is the very best, which task sets the standard, and what that standard is from software task repositories. TaskBench supports a full DEA-based undesirable outputs transformation, the relatively high performance tasks identification, the reference sets establishment and sensitivity analysis. Techniques for the design of software task data structure, the establishment of performance benchmarking metrics, the mining, evaluation and selection of the task set T from task repositories have been previously reported in Ref.[6]. In this experiment, based on our previous work in Ref.[6], we further present experimental results of TaskBeD's mechanisms of transforming undesirable outputs, identifying high performance tasks, establishing reference sets and sensitivity analysis on a task set mined from the software process management tool (SoftPM) in Institute of Software, Chinese Academy of Sciences (ISCAS). Based on our previous work in Refs.[5,6], three typical task outputs ($Y=\{Y_{1i},Y_{2i},Y_{3i}\}$) and one typical input ($X=\{X_{1i}\}$) (see Table 2) are mined out from softPM as the software task performance benchmarking metrics. Based on the established metrics, 30 completed software tasks are mined from the task repository, i.e., $T=\{t_1,t_2,...,t_{30}\}$ (see Table 3). Each $t_j(j=1,...,30)$ is of the engineering type, implements the same software process management package and is based on J2EE Web Applications, i.e. T is homogenous. The further detailed task data mining rules and process of T have been previously defined and reported in our previous work[5,6]. As this experiment emphasizes more on the TaskBeD's models and algorithms, therefore we directly use T reported in Ref.[6]. Based on the metrics and task data set mining rules and process in Ref.[5,6], TaskBeD also can further be applied to more metrics and the task set of larger size. Experiments of more metrics and task set of larger size can be found in Ref.[17]. Therefore, we here present our representative experiment on T as a demonstration of TaskBeD's models and algorithms due to page limits.

After T has been mined out, we next transform the undesirable outputs. By investigating on the metrics in Table 2, Y_{2j} (*Program Defects*) is identified as one undesirable output (see Definition 6) and requires to be transformed. We thus apply TaskBeD's undesirable outputs transforming algorithm to Y_{2j}. The maximum *Program Defects* is 12 defects. Therefore, 14, which is slightly larger than the maximum *Program Defects* (12) by 2, is chosen as β_j. Next, the *Program Defects* of each t_j is subtracted from β_j. After undesirable outputs are transformed, T is prepared well for further benchmarking processing.

Table 2 Input and output evaluation metrics of tasks

Type	Metric	Meaning	Unit	Type	Metric	Meaning	Unit
X_{1j}	Effort	Actual effort of the task	Person hour	Y_{2j}	Program defects	Program defects found in test phase	Defects
Y_{1j}	Program size	Program size of work products	LOC	Y_{3j}	Documents	Documents for the task	Pages

Table 3 Task data set

t_u	Size	Def	Doc	Effort	t_u	Size	Def	Doc	Effort	t_u	Size	Def	Doc	Effort
t_1	1579	12	6	7.5	t_{11}	725	5	5.5	5.5	t_{21}	345	5	4	5.3
t_2	1320	10	5	7	t_{12}	718	6	6	6	t_{22}	263	3	5	5.1
t_3	1202	8	7	7.5	t_{13}	700	4	3	5.4	t_{23}	236	5	3	3
t_4	1000	5	5	7	t_{14}	685	9	5	5	t_{24}	233	4	3	3.5
t_5	980	9	2	6.5	t_{15}	678	7	6.5	5.5	t_{25}	220	2	4	3
t_6	940	7	4	6	t_{16}	620	3	7.5	6.5	t_{26}	200	4	2.5	4
t_7	824	6	5	6.5	t_{17}	598	5	3	6.4	t_{27}	178	3	1	3
t_8	763	7	5	5	t_{18}	568	3	5	5.5	t_{28}	155	2	1.5	2
t_9	744	5	5.5	6	t_{19}	460	5	3	6.5	t_{29}	144	2	1	1.5
t_{10}	735	6	4	7	t_{20}	458	4	2.5	6	t_{30}	124	3	3	2

(size: Program size; def: Program defects; doc: Documents)

5.1 Identifying the relatively efficient tasks

We next apply Algorithm 2 to T to identify the relatively efficient tasks. The resulted relative performance score θ^u for each t_u are shown in Table 4. In Table 2, we can see each task has multivariate inputs and outputs (one input like *Effort*; three outputs like *Program Size*, *Program Defects* and *Documents*). Table 4 shows that θ^u for each t_u are obtained under the multivariate inputs/outputs (see Table 2). These results verify that TaskBeD can effectively identify the relatively efficient tasks under multivariate inputs/outputs. Moreover, TaskBeD's CCR task benchmarking model only puts t_1, t_{29} and t_{30} on the task performance frontier (see Definition 7). The BCC puts t_1, t_3, t_4, t_{15}, t_{16}, t_{18}, t_{25}, t_{29} and t_{30} on the task performance frontier. The notable difference between the results of CCR and BCC lies in that t_3, t_4, t_{15}, t_{16}, t_{18} and t_{25} are positioned on the efficiency frontier in BCC while not recognized as the relatively efficient tasks in CCR. This result reveals that TaskBeD's BCC model has a better capability to establish different performance benchmarks for tasks of different sizes. For example, in CCR model, only t_1, t_{29} and t_{30} can be identified as the task performance benchmarks. In BCC model, more fine-grained efficiency scores (t_1, t_3, t_4, t_{15}, t_{16}, t_{18}, t_{25}, t_{29}, t_{30}) can be identified and such fine-grained efficiencies enable developers to establish much fine-grained performance benchmarks for tasks of different sizes. To make it clearer, let us use t_{14} as an example. In BCC, we can benchmark t_{14} on t_{15}. In CCR, we can only benchmark t_{14} on t_1, t_{29} or t_{30}. The size difference between t_{14} and t_{15} is obviously much smaller than that of t_{14} between t_1, t_{29} or t_{30}. This result surely shows that TaskBeD's BCC model is more appropriate to evaluate software tasks with similar scale and ensures that relatively larger tasks are compared with other relatively larger tasks, and relatively smaller tasks with relatively smaller tasks than CCR. For example, one task benchmarking solution of T may be that $\{t_1,t_3,t_4\}$ is used as the performance benchmark for relatively large tasks, $\{t_{15},t_{16},t_{18}\}$ for relatively middle-scale tasks and $\{t_{29},t_{30}\}$ for relatively small tasks under Algorithm 2.

Table 4 Performance scores of tasks (ES: efficiency score)

		CCR	BCC			CCR	BCC			CCR	BCC
t_u	Size	ES	ES	t_u	Size	ES	ES	t_u	Size	ES	ES
t_1	1579	**1.000 0**	**1.000 0**	t_{11}	725	0.904 6	0.971 7	t_{21}	345	0.592 3	0.596 2
t_2	1320	0.918 6	0.934 8	t_{12}	718	0.872 7	0.917 2	t_{22}	263	0.682 6	0.784 3
t_3	1202	0.942 2	**1.000 0**	t_{13}	700	0.732 3	0.933 2	t_{23}	236	0.764 8	0.786 1
t_4	1000	0.795 3	**1.000 0**	t_{14}	685	0.918 2	0.924 7	t_{24}	233	0.663 7	0.671 0
t_5	980	0.756 1	0.768 5	t_{15}	678	0.983 6	**1.000 0**	t_{25}	220	0.936 8	**1.000 0**
t_6	940	0.821 7	0.865 9	t_{16}	620	0.894 6	**1.000 0**	t_{26}	200	0.512 8	0.525 5
t_7	824	0.762 6	0.812 5	t_{17}	598	0.552 9	0.566 1	t_{27}	178	0.529 8	0.547 4
t_8	763	0.959 3	0.973 5	t_{18}	568	0.778 8	**1.000 0**	t_{28}	155	0.847 5	0.875 0
t_9	744	0.837 5	0.912 0	t_{19}	460	0.470 4	0.473 0	t_{29}	144	**1.000 0**	**1.000 0**
t_{10}	735	0.610 4	0.623 5	t_{20}	458	0.491 8	0.507 8	t_{30}	124	**1.000 0**	**1.000 0**

To sum up, the above analysis results show that although software tasks are much fine-grained compared with projects, TaskBeD can effectively identifies the relatively efficient tasks to set up the performance benchmark using

Algorithm 2. Moreover, with the merits of dealing with multivariate and VRS and enabling developers to establish different task performance benchmarks for tasks of different scale, TaskBeD's VRS model is more appropriate for benchmarking software tasks than CCR model.

5.2 Establishing different reference sets

To verify the effectiveness of Algorithm 3, we apply the Algorithm 3 to *T*. The resulted reference relationships among tasks in *T* are shown in Table 5.

Table 5 Reference relationships of *T*

	CCR			BCC				CCR			BCC		
t_u	ES	P	PW	ES	P	PW	t_u	ES	P	PW	ES	P	PW
t_1	**1.000 0**	t_1	**1.000 0**	**1.000 0**	t_1	**1.000 0**	t_{14}	0.918 2	t_1	0.359 4	0.924 7	t_1	0.249 4
t_2	0.918 6	t_1	0.818 0	0.934 8	t_1	0.780 3			t_{30}	0.947 9		t_{15}	0.357 7
		t_{29}	0.197 0		t_4	0.065 8						t_{30}	0.392 9
					t_{29}	0.153 9	t_{15}	0.983 6	t_1	0.307 5	**1.000 0**	t_{15}	**1.000 0**
t_3	0.942 2	t_1	0.685 7	**1.000 0**	t_1	**1.000 0**			t_{30}	1.552 6			
		t_{30}	0.961 9				t_{16}	0.894 6	t_1	0.232 9	**1.000 0**	t_{16}	**1.000 0**
		t_1	0.575 8						t_{30}	2.034 2			
t_4	0.795 3	t_{29}	0.262 0	**1.000 0**	t_4	**1.000 0**	t_{17}	0.552 9	t_1	0.316 1	0.566 1	t_1	0.291 0
		t_{30}	0.427 7						t_{29}	0.518 7		t_{25}	0.141 7
t_5	0.756 1	t_1	0.591 6	0.768 5	t_1	0.582 6			t_{30}	0.194 9		t_{29}	0.537 3
		t_{29}	0.318 1		t_{29}	0.417 4	t_{18}	0.778 8	t_1	0.271 5	**1.000 0**	t_{18}	**1.000 0**
		t_1	0.551 0		t_1	0.582 6			t_{30}	1.123 7			
t_6	0.821 7	t_{29}	0.402 4	0.768 5	t_{29}	0.417 4	t_{19}	0.470 4	t_1	0.228 6	0.473 0	t_1	0.226 3
		t_{30}	0.097 2						t_{29}	0.308 7		t_{29}	0.339 4
		t_1	0.463 8		t_1	0.312 5			t_{30}	0.439 8		t_{30}	0.434 3
t_7	0.762 6	t_{30}	0.739 0	0.812 5	t_3	0.083 3	t_{20}	0.491 8	t_1	0.218 3	0.507 8	t_1	0.188 1
					t_4	0.124 9			t_{29}	0.624 0		t_4	0.039 6
					t_{25}	0.479 2			t_{30}	0.188 7		t_{25}	0.133 7
		t_1	0.418 0		t_1	0.346 9						t_{29}	0.638 6
t_8	0.959 3	t_{30}	0.830 7	0.973 5	t_{15}	0.193 4	t_{21}	0.592 3	t_1	0.135 0	0.596 2	t_1	0.064 0
					t_{25}	0.282 5			t_{30}	1.062 4		t_{15}	0.230 9
					t_{30}	0.177 3						t_{30}	0.705 1
		t_1	0.388 2		t_3	0.447 0	t_{22}	0.682 6	t_1	0.042 3	0.784 3	t_{16}	0.285 7
t_9	0.837 5	t_{30}	1.057 0	0.912 0	t_4	0.100 5			t_{30}	1.582 0		t_{25}	0.714 3
					t_{16}	0.016 7	t_{23}	0.764 8	t_1	0.084 1	0.786 1	t_1	0.075 4
					t_{25}	0.435 8			t_{30}	0.831 7		t_{29}	0.113 1
		t_1	0.412 3		t_1	0.395 9						t_{30}	0.811 4
t_{10}	0.610 4	t_{29}	0.191 7	0.623 5	t_{25}	0.312 7	t_{24}	0.663 7	t_1	0.076 9	0.671 0	t_1	0.073 4
		t_{30}	0.443 0		t_{29}	0.250 1			t_{29}	0.064 6		t_{29}	0.110 1
					t_{30}	0.041 4			t_{30}	0.824 6		t_{30}	0.816 5
		t_1	0.373 9	0.971 7	t_3	0.477 5	t_{25}	0.936 8	t_1	0.041 1	**1.000 0**	t_{25}	**1.000 0**
t_{11}	0.904 6	t_{30}	1.085 5		t_4	0.042 6			t_{30}	1.251 2			
					t_{16}	0.007 1	t_{26}	0.512 8	t_1	0.054 8	0.525 5	t_1	0.047 8
					t_{25}	0.472 8			t_{29}	0.231 5		t_{29}	0.321 7
		t_1	0.353 1		t_1	0.201 2			t_{30}	0.646 6		t_{30}	0.630 5
t_{12}	0.872 7	t_{30}	1.293 8	0.917 2	t_{15}	0.243 6	t_{27}	0.529 8	t_1	0.029 6	0.547 4	t_1	0.023 7
					t_{16}	0.282 4			t_{29}	0.911 7		t_{29}	0.976 3
					t_{25}	0.272 7						t_{25}	0.166 7
		t_1	0.373 0		t_1	0.010 3	t_{28}	0.847 5	t_1	0.008 2	0.875 0	t_{29}	0.083 3
t_{13}	0.732 3	t_{29}	0.771 2	0.933 2	t_4	0.632 2			t_{29}	0.799 7			
					t_{29}	0.357 5			t_{30}	0.217 1			
							t_{29}	**1.000 0**	t_{29}	**1.000 0**	**1.000 0**	t_{29}	**1.000 0**
							t_{30}	**1.000 0**	t_{30}	**1.000 0**	**1.000 0**	t_{30}	**1.000 0**

(P: peer; PW: peer weight)

Both peers and peer weights (see Definition 11) of each task derived from Algorithm 3 are shown in Table 5. For example, t_7 derives a reference set of tasks $\{t_1,t_{30}\}$ using CCR model under the results of Algorithm 3. By further investigating the peer weight {0.4638,0.7390} of each peer in the reference set $\{t_1,t_{30}\}$, t_{30} is identified as the most suitable task to learn best practices from because t_{30} has the biggest peer weight 0.7390 in the peer set

$\{t_1,t_{30}\}$. Similarly, using BCC models, results indicate that developers can determine to emulate best practices from t_{25} by comparing the peer weight $\{0.3125,0.0833,0.1249,0.4792\}$ among its reference set $\{t_1,t_3,t_4,t_{25}\}$. The reference set and the most valuable task to emulate for the other tasks $t_u(t_u \in T)$ in Table 5 can be derived in a similar way.

To sum up, by investigating the reference set using Algorithm 3, TaskBeD can establish different reference sets for each relatively inefficient task. Moreover, at the aid of peer weights of the peers, TaskBeD can further find which task is of the biggest improvement reference value to the developer's own personal software process.

6 Sensitivity Analysis

In this experiment, we verify TaskBeD's capability of sensitivity analysis by applying Algorithm 4. Using Algorithm 4, TaskBeD first calculates the average performance score of (θ^u) for T. The average VRS efficiency (E_{mean}), standard deviation (SD), minimum VRS efficiency ($E_{\min}$) and the number of efficient tasks (N_{eff}) for T are shown in Table 6. E_{mean}=0.8323 shows that there is a potential for performance improvement of the task set T between 10 to 20 percent compared with the best practices tasks $\{t_1,t_3,t_4,t_{15},t_{16},t_{18},t_{25},t_{29},t_{30}\}$. Those nine tasks $\{t_1,t_3,\ t_4,t_{15},t_{16},t_{18},t_{25},t_{29},t_{30}\}$ are identified by Algorithm 2 as of relatively high performance (see Definition 7). Secondly, each of the nine tasks is removed one at a time and the E_{mean} is calculated using Algorithm 4 in TaskBeD. The results are shown in Table 7. In Table 7, t_u^{remv} indicates the uth task is removed from T. E_{mean} indicates that the average performance scores of the task set ($T-\{t_u^{remv}\}$) and thus N equals to the size of the set ($T-\{t_u^{remv}\}$), i.e., N=29. Finally, the E_{mean} in Table 6 and Table 7 is compared.

Table 6 Average performance score of T

N	E_{mean}	SD	E_{min}	N_{eff}
30	0.832 3	0.181 3	0.473 0	9

Table 7 Average performance score of T with a task removed

N	t_u^{remv}	E_{mean}	N	t_u^{remv}	E_{mean}	N	t_u^{remv}	E_{mean}
29	t_1	0.840 8	29	t_{15}	0.828 9	29	t_{25}	0.832 7
29	t_3	0.826 6	29	t_{16}	0.837 0	29	t_{29}	0.844 4
29	t_4	0.830 3	29	t_{18}	0.826 6	29	t_{30}	0.836 4

(t_u^{remv}: A removed task; E_{mean}: Mean of E_{VRS})

By comparing E_{mean} in Table 6 and Table 7, we can know that none of the identified frontier tasks $\{t_1,t_3,t_4,t_{15},t_{16},t_{18},t_{25},t_{29},t_{30}\}$ are extreme in the sense that their removal hardly influence the average efficiency E_{mean}. i.e., there is still a potential improvement of around 20 percent (see Table 7). This result verifies that TaskBeD's task benchmarking models and algorithms on software tasks (see Table 3) are reasonable.

7 Conclusions and Future Work

In this paper, we propose a novel software task performance benchmarking method based on DEA-TaskBeD to support quantitative software process improvement. TaskBeD can be regarded as a further extension of Refs.[3−6] by scaling the DEA-based software projects benchmarking method down to fine-grained software tasks and introducing mechanisms of transforming undesirable outputs and sensitivity analysis. Experimental results show that the proposed TaskBeD method achieves a good result of dealing with multivariate, VRS and undesirable outputs.

Our future work will concentrate on the following topics. Firstly, we plan to further evaluate the software tasks to analyze the developer's personal software process change effects using TaskBeD. Secondly, we are extending our

interactive and visual tool TaskBench to provide more and better supports for benchmarking software tasks using TaskBeD.

References:

[1] Humphrey WS. Introduction to the Team Software Process. Addison Wesley Professional, 1999. 10−343.

[2] Stark JA, Crocker R. Trends in software process: The PSP and agile methods. IEEE Software, 2003,20(3):89−91.

[3] Stensrud E, Myrtveit I. Identifying high performance ERP projects. IEEE Trans. on Software Engineering, 2003,29(5):398−416.

[4] Myrtveit I, Stensrud E. Benchmarking COTS projects using data envelopment analysis. In: William A, James B, *et al*., eds. Proc. of the 6th Int'l Software Metrics Symp. Washington: IEEE Computer Society, 1999. 269−278.

[5] Ding LP, Yang QS, Sun L, Tong J, Wang YJ. Evaluation of the capability of personal software process based on data envelopment analysis. In: Li MS, Barry W, Leon JO, eds. Proc. of the Software Process Workshop. Beijing: Springer-Verlag, 2005. 235−248.

[6] Ruan L, Wang YJ, Wang Q, Li MS, Yang Y, Xie LZ, Liu DP, Zeng HT, Zhang SH, Xiao JC, Zhang L, Nisar MW, Dai J. Empirical study on benchmarking software development tasks. In: Wang Q, Dietmar P, Diet MR, eds. Proc. of the Int'l Conf. on Software Process. Minneapolis: Springer-Verlag, 2007. 221−232.

[7] Katrina D, Maxwell PF. Benchmarking software development productivity. IEEE Software, 2000,17(1):80−88.

[8] Farris JA, Groesbeck RL, Aken EMV, Letens G. Evaluating the relative performance of engineering design projects: A case study using data envelopment analysis. IEEE Trans. on Engineering Management, 2006,53(3):471−482.

[9] Rollo AL, Morris P, Wasylkowski E, Dekkers C, Forselius P. Benchmarking Standards, Vol.1.0. US: Int'l Software Benchmarking Standards Group, 2008. 7−43.

[10] Fleming QW, Koppelman JM. Earned Value Project Management. 3rd ed., Project Management Institute, 2005. 8−230.

[11] Florac WA, Carleton AD. Measuring the Software Process: Statistical Process Control for Software Process Improvement. Boston: Addison-Wesley Professional, 1999. 10−272.

[12] Banker RD, Charnes A, Cooper W. Some models for estimating technical and scale inefficiencies in data envelopment analysis. Management Science, 1984,30(9):1078−1092.

[13] Charnes A, Cooper WW, Rhodes E. Measuring the efficiency of decision making units. European Journal of Operation Research, 1978,2(6):429−444.

[14] Cooper WW, Seiford LM, Tone K. Data Envelopment Analysis: A Comprehensive Text with Models, Applications, References and DEA-Solver Software. 2nd ed., Springer-Verlag, 2006. 10−490.

[15] Holger S. Undesirable outputs in efficiency valuations. European Journal of Operation Research, 2001,132(2):400−410.

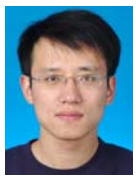
[16] Boussofiane A, Dyson RG, Thanassoulis E. Applied data envelopment analysis. European Journal of Operational Research, 1991, 52(1):1−15.

[17] Ruan L. Empirical study on benchmarking software development tasks. Beijing: Institute of Software, the Chinese Academy of Sciences, 2006. 1−14.

RUAN Li is a Ph.D. candidate at the Institute of Software, the Chinese Academy of Sciences. Her current research areas are software process technologies, quality management and data mining.

WANG Qing is a professor at the Institute of Software, the Chinese Academy of Sciences. Her major research fields are software process technologies and quality assurance, software estimation and requirement engineering.

WANG Yong-Ji is a professor at the Institute of Software, the Chinese Academy of Sciences. His researches areas are computer-controlled real-time systems, artificial intelligence, data mining, software engineering.



ZENG Hai-Tao is a Ph.D. candidate at the Institute of Software, the Chinese Academy of Sciences. His current research areas are operating system security, database security and real-time system.