EXHIBIT D

26 March 2010

Expert Report of Donald J. Reifer

Oracle USA, Inc., et al. v. SAP AG, et al.

Designated Highly Confidential
Pursuant to Protective Order

Donald J. Reifer, President
Reifer Consultants, Inc.
14820 N. Dragons Breath Lane
Prescott, AZ 86305-5644

---

exceptionally high when compared with current salary benchmarks for Information Technology (IT) workers from this nation.

- **Soundness** – I investigated whether or not Mr. Pinto's estimating approach was sound. It was not. For example, Mr. Pinto incorrectly assumes that user documentation is outside the scope of the normal software development activities. As will be discussed later in this report, most of it is not.

As part of my evaluation of the Pinto Report, I identified a number of major concerns related to my assessment criteria. For convenience, major issues identified are discussed in more detail in the next section of this report, Section VI.

b. Specific Analysis Performed

I reviewed the Pinto Report from a COCOMO II point-of-view. My findings are summarized in the following paragraphs.

- **Mr. Pinto's Ten-Step Estimating Approach**

Because of its potential impact on the factors used in the COCOMO II model, I reviewed Mr. Pinto's ten-step estimating approach. My comments are as follows:

- **Mr. Pinto's Step 1: Identify and Group Source Code Components**

Most organizations do not develop source code from scratch. They try to reuse legacy and Commercial Off-The-Shelf (COTS) software to reduce the volume of work involved. They use applications generators to develop the code whenever possible because they accomplish this task automatically without human intervention. They convert some of the applications using commercial tools developed for that purpose again of reducing the development effort. Some of the most advanced software groups develop their own libraries of reusable software when the payback associated with the extra costs involved is warranted. To account for these practices, users of models like COCOMO II.2000 group the software so that they can reduce the size of the software generated to account for the reduced workload. The reuse model in COCOMO II.2000 [BOE01] addresses this existing code and converts them into equivalent new source lines of code using user inputs for modified code in the model or the rules of thumb summarized in Table 2.

Code Category	% Design Modified	% Code Modified	% Integration Modified	Relative Percent Effort
New – all original	N/A	N/A	N/A	100%
Adapted – existing software that is changed or modified	0 to 100%	0 to 100%	0 to 100%	0 to 100%
NOMINAL	40%	40%	60%	46%
Reused – existing software that is used as-is/calls counted	0%	0%	0 to 100%	30% (at most)
COTS – requires glue code wrappers that are counted	Count the wrapper code as new and add effort needed to test wrapped COTS package			

Table 2: Counting Conventions for Equivalent Source Lines of Code

In other words, the size for a software package is most often never all new source lines of code. It is smaller because of these reuse considerations.

Mr. Pinto assumed that all of the suites of products under consideration would have to be redeveloped as new code. This assumption is just not true for most of the applications that I have been associated with. Instead, Mr. Pinto should have grouped the software into new, modified, reused, generated and COTS categories and used the Software Engineering Institute (SEI) counting standards that he referenced to address these different types of software [ORCLX-PIN-000017] as these different groupings of software were used to develop the suites of products in a manner similar to that shown in Figure 1 to calculate source lines of code.

The net results of Mr. Pinto's failure to use such practices as he grouped the software are that his estimates of source lines of code are highly suspect and his size estimates seem biased high. This in turn biases Mr. Pinto's COCOMO II estimates high.

- **Mr. Pinto's Step 2: Count the Number of Source Lines of Code**

I next tried to acquire copies of the specialized counting utilities that Mr. Pinto developed to tally source lines of code. My goal was to replicate his analysis as I tried to understand how he counted source lines of code assuming that all of the code was considered new code. While Mr. Pinto infers that calculating source lines of code is simple [Pinto Report, p. 15], the SEI manual that he relied on to provide counting conventions refutes his claim. Counting lines of code is difficult and requires more powerful tools than Mr. Pinto developed to deal with the many nuances that he acknowledges may be present in the code that the counters must handle.

Why Mr. Pinto developed his own source lines of code counters puzzled me. Powerful tools, frequently used by industry, that perform the task exist and can be acquired for free from

sites like those at the University of Southern California (see the tools section of http://sunset.usc.edu). When investigating Mr. Pinto's counters more closely, one sees that while they count the code, they do not do so in a manner that fully complies with the standards and conventions defined by the SEI. Many of the nuances that Mr. Pinto acknowledges that are present like embedded constants in the C programming language were just overlooked by his utilities. [Pinto Report, pp. 15 – 16]

To understand the impact of these counts, my assistant and I developed a set of utilities that replicated the code for Mr. Pinto's counters as described in ORCLX-PIN-000067 for the C programming language (including headers) running on a PC running Windows Vista. I then had my assistant download the C source code for a piece of public domain software for a flight simulator called FlightGear (http://www.flightgear.org). I next had him count the code for the main routine using the Pinto utilities and the freely available USC developed language code counters called Unified CodeCount (UCC) (see the download section of http://sunset.usc.edu). The results of this counting experiment are provided in Table 3 [see SAP-DJR-000003 for summary]. These differences lead me to question both the accuracy and correctness of Mr. Pinto's counts and his customized counting utilities.

SLOC Counting Tool	Total Number Lines	Total Blank Lines	Total Comment Lines	Total Physical SLOC	Total Logical SLOC	Total Number Files
Pinto Code Counter[1]	58,739	9,687	11,941	37,111	30,215	199
USC Code Counter	58,752	9,687	12,086	36,979	27,585	199
DIFFERENCE	- 13	0	- 145[2]	132	- 2,630	0

Table 3: Results of Code Counting Experiment using FlightGear

Notes

[1] This is a counter that follows Mr. Pinto's parsing rules as described in ORCLX-PIN-000066 and replicated his code as described in ORCLX-PIN-000067.

[2] The difference in comment lines is primarily the number of embedded constants in the count.

The main difference in Logical Source Lines of Code ("SLOC") calculation occurred due to how embedded comments were counted by Mr. Pinto's utility software. There was also some confusion over how Mr. Pinto counted compiler directives and data declarations.

To verify whether this error consistently existed in the JD Edwards code, my assistant and I developed a second set of counters for the Java J2EE programming language following the

parsing rules described in ORCLX-PIN-000076 and replicating the code described in ORCLX-PIN-000077 to run on my Windows/Vista PC platform. We then extracted three C and two Java J2EE routines from the JD Edwards EnterpriseOne code library and ran them through our versions of the Pinto utilities and USC UCC counter. The results, which are summarized in Table 4, verify that an error of nine and one half percent exists for all of the code inspected [see SAP-DJR-000004 for summary including file list].

Definition Checklist for Source Statements Counts

Definition name: __Logical Source Statements___ Date:_____

_____(basic definition)_____Originator:_COCOMO II____

Measurement unit:	Physical source lines						
	Logical source statements	√					
Statement type	Definition	√	Data Array			Includes	Excludes
When a line or statement contains more than one type, classify it as the type with the highest precedence.							
1 Executable			Order of precedence		1	√	
2 Non-executable							
3 Declarations					2	√	
4 Compiler directives					3	√	
5 Comments							
6 On their own lines					4		√
7 On lines with source code					5		√
8 Banners and non-blank spacers					6		√
9 Blank (empty) comments					7		√
10 Blank lines					8		√
11							
12							
How produced	Definition	√	Data array			Includes	Excludes
1 Programmed						√	
2 Generated with source code generators							√
3 Converted with automated translators						√	
4 Copied or reused without change						√	
5 Modified						√	
6 Removed							√
7							
8							
Origin	Definition	√	Data array			Includes	Excludes
1 New work: no prior existence						√	
2 Prior work: taken or adapted from							
3 A previous version, build, or release						√	
4 Commercial, off-the-shelf software (COTS), other than libraries							√
5 Government furnished software (GFS), other than reuse libraries							√
6 Another product							√
7 A vendor-supplied language support library (unmodified)							√
8 A vendor-supplied operating system or utility (unmodified)							√
9 A local or modified language support library or operating system							√
10 Other commercial library							√
11 A reuse library (software designed for reuse)						√	
12 Other software component or library						√	
13							
14							

Figure 1: SLOC Definition Checklist

SLOC Counting Tool	Language	Total Number Lines	Total Blank Lines	Total Comment Lines	Total Physical SLOC	Total Logical SLOC	Total Number Files
Pinto	C[1]	779	10	230	539	528	3
	Java[1]	156	8	43	105	95	2
USC	C	779	10	245[2]	524	478	3
	Java	156	8	55[2]	104	86	2

Table 4: Results of Code Counting Experiment using Five Routines from the JD Edwards EnterpriseOne Software Applications Package

Notes

[1] These are utilities that count C and Java code following the rules and replicating the code as Mr. Pinto's describes in ORCLX-PIN-000066, PIN-000067, PIN-000076 and PIN-000077.

[2] The difference in comment lines is the number of embedded constants in the count.

While seemingly small, a nine and one half percent error in counts is significant when working with numbers of this magnitude. For the C and Java programming language code in the JD Edwards EnterpriseOne suite, this error means that the code count in Mr. Pinto's Table 5 should be reduced by 738,605 source lines of code (using 7,774,791 SLOC as the base count). I will address this error in Section VII of this report.

Because of the impact, I went a step further. As summarized in Table 5, I counted a larger sample of the C code in the JD Edwards EnterpriseOne suite to assess whether this error propagated throughout it. As noted in the summary, the error for C code including the headers was 14.5% when I compared the USC versus Pinto counts [see SAP-DJR-000005 for summary]. I use these results to correct the C and Java sizing source lines of code counts later in this report when I develop an independent cost estimate for this suite, which I develop in order to point out the various, substantial errors in Mr. Pinto's analysis and conclusions.

Language	No. Programs	USC Count	Pinto Count[1]	% Difference
Header	836	153,172	153,205	0.02
C	728	779,139	937,620	16.9
	TOTAL	932,311	1,090,825	14.5

Table 5: Results of Code Counting for JD Edwards EnterpriseOne Package

Notes

[1] These are C language counting utilities that replicate Mr. Pinto's code and follow the rules provided in ORCLX-PIN-000066 and ORCLX-PIN-000067.

- **Mr. Pinto's Step 3: Determine the Amount of Functionality**

Because the results of this step are not germane to my COCOMO II analysis, I will not comment on them other than to say that in all of my experience I have not seen SLOC backfired to determine the number of function points.

- **Mr. Pinto's Step 4: Determine the Number of Pages of Documentation**

Mr. Pinto next assumes that user and support documentation must be estimated in addition to the documentation that is normally produced as a by-product of the software development process. This is a controversial assumption and an issue that I will discuss in the next Section. Estimating this documentation separately leads to double counting because the user and programmer reference manuals are normally already accounted for by tasks performed during standard software development processes.

Mr. Pinto then uses a partial Table that he took from [JON02], which he did not cite, to identify the types of documentation that need to be generated using a conversion factor of so many pages of documentation per function point to develop his page counts. The complete Table [ORCLX-PIN-000065, Table 8, Pages per FP] identifies a range of factors that is much broader than appears in the Pinto Report.

As I previously stated, when Mr. Pinto uses the conversion factors in his report, an unreasonable estimate of the amount of documentation results. As I will discuss in the next Section of this report, one reason for this is that substantial double counting is involved. Another is that Mr. Pinto did not step back to assess what the numbers really meant from a user perspective. Using Mr. Pinto's numbers, millions of pages of documentation would have to be produced to satisfy user needs. Having over five thousand volumes of user and support documentation each over four hundred pages in length is just more than the normal user would want to deal with in my opinion. This leads to me to question the reasonableness of his approach.

- **Mr. Pinto's Step 5: Derive the Productive Hours of Effort**

For his next step, Mr. Pinto makes the following, questionable assumptions when he expands the number of function points into productive hours of effort.

1. He used the value 144 productive staff hours per staff month in his calculations, but this value comes from a reference that was developed in Europe [ORCLX-PIN-

000013]. Instead, he should have increased the number of hours to 152 staff hours per staff-month because this is the expansion ratio used by the COCOMO II and other cost models for labor in the United States. [BOE01] The result leads to an error in cost because the COCOMO II.2000 model assumes that it will take more effort in staff hours than he uses to get the job done per the predicted duration.

2. Mr. Pinto estimates the effort involved in labor hours to generate the volumes of documentation that he determined were needed in addition to that which is normally generated as a by-product of the software development process. Because there is substantial double counting involved, these additional costs in staff hours inflate his already high estimates even higher.

Because of the inconsistencies noted, I again have to question the accuracy, correctness, currency, reasonableness, and soundness of the approach that Mr. Pinto takes to build his software development cost estimates. These inconsistencies make it difficult to believe that his results are credible.

- **Mr. Pinto's Step 6: Distribute the Effort across the Product Development Life-Cycle**

Mr. Pinto next begins building an effort distribution model so that he can assign labor rates to the productive hours of effort that he estimated by role to activities performed during the software development process. This is a normal procedure that estimators perform to determine the labor rates to use to price the staff hours predicted using a cost model like COCOMO II.

For this step, Mr. Pinto selects a software development life cycle and distributes a percentage of the software effort involved to each of its phases using a draft of one of my publications as his source [ORCLX-PIN-000014]. He next takes the hours that he estimated in his Step 5 and distributes them to life cycle phases/activities using these percentages. However, he did not use the numbers correctly [page 24 of the Pinto Report]. Instead, he builds an effort distribution model which spends a seemingly disproportionate amount of time doing front-end tasks. This loading focuses the effort inappropriately on specification rather than production tasks. The results are unreasonable in my opinion.

I will correct these distributions in the next Section of this report when I develop a corrected labor rate model for use with the COCOMO II.2000 cost model. Needless to say, I

question the correctness, reasonableness, and soundness of the effort distribution model that Mr. Pinto built in this step and the accuracy of number of hours he derived in Table 14 of his report.

- **Mr. Pinto's Step 7: Allocate Productive Hours of Effort to Team Roles**

Mr. Pinto next distributes effort by role to activities within each of the phases of the software development cycle he selected as illustrated in Table 15 of his report. Mr. Pinto did this so that he could distribute estimated staff hours by role to specific tasks as shown in Table 16 of his report. These spreads summarize how many hours each of the team members will spend on the development effort by role.

When the final product of this distribution model is examined, it suggests that about sixty percent of the labor force will need to perform management and support tasks, while the remaining forty percent of the labor force does the software development work. As I will discuss in the next Section of this report, these percentages seem reversed.

I will correct these distributions in the next Section of this report when I develop a corrected labor rate model for use with the COCOMO II.2000 cost model. Mr. Pinto's emphasis on management overhead rather than technical effort again makes me question the accuracy, correctness, reasonableness, and soundness of his effort distribution model.

- **Mr. Pinto's Step 8: Derive the Cost of Localization and Documentation Translation**

In this next step, Mr. Pinto estimates the cost of localization of documentation. He calls for translation of user documentation into twenty-one different languages assuming that support documentation can remain in English. This translation expands his original million or so page estimate to over thirty-two million pages of user documentation. [Pinto Report, pp. 28 – 29]

Whether or not the user documentation would need to be translated into twenty-one languages is debatable. Mr. Pinto cites no evidence to suggest – as he does – that there are marketplace demands for such documentation. However, Mr. Pinto's numbers are skewed on the high side, since it appears to me that the costs quoted by Mr. Pinto are primarily for manual translation of court and patent documents where errors in language use cannot be tolerated [ORCLX-PIN-000020]. He does not consider using the many powerful automated translation tools like SYSTRAN that can reduce the time to generate a page of user documentation from days to seconds and cost from $15 per page, as quoted in the Pinto Report, to about 30¢ a page (see for example http://www.translation.net/net_faq.html).

The use of professionals to translate 32,010,559 pages of user text seems unreasonable to me when capable tools are available to do the job at about two percent of Mr. Pinto's estimated cost. The costs would stay low even if professionals were hired to check the results of these automated translators for accuracy in those areas of the manuals where it mattered to the users. Also, producing and keeping such a vast amount of documentation current seems overwhelming.

- **Mr. Pinto's Step 9: Apply Hourly Rates to Determine the Development Costs**

Mr. Pinto next develops what he calls a hybrid staffing model to develop labor rates to be used to price the effort as distributed by roles to phases. The following four staffing scenarios are considered by Mr. Pinto and then combined primarily in order to try to meet one of the conditions set by Mr. Pinto, namely, that the development be completed in two years:

1. **Offshore** – Product development tasks entirely outsourced to an offshore company, typically in India.
2. **On-staff** – Product development tasks staffed using full-time TN personnel located in Bryan, Texas.
3. **Outsourced to U.S. Integrator** – Product development tasks would be outsourced to a U.S. organization with the skills, knowledge and experience to need to develop and integrate Oracle replacement products.
4. **Outsourced to Oracle Consulting** – Product development would be outsourced to Oracle Consultants with expertise allowing them to charge Oracle's consulting rates.

[Pinto Report, p. 30]

Mr. Pinto next develops the estimated cost of development by applying a set of standard hourly rates to each team member's role using a staffing model that calls for Oracle consultants to accomplish the Planning Phase activities, the U.S. based Integrator to handle the specify and deploy phases, TN on-staff resources at Bryan, Texas, to conduct the design and document phases, and offshore resources in India to complete the build and test phases.

Mr. Pinto's development of cost estimates for Oracle suites of products using this approach raises many issues. First and foremost, recommending that the primary Program, Project and Quality Management roles and responsibilities for such a large software development project be given to consultants is something most companies that I have worked with would never consider. Senior managers want their people in charge of such efforts because their people understand the organization and its practices and have loyalty to it. Second, because of the rates,

companies tend to use high priced consultants only in specialist roles. If there are specific design or implementation problems, companies pay the price because those consultants have the skills and knowledge to resolve issues quickly. Third, rates quoted for India seem high even when consulting Mr. Pinto's references [ORCLX-PIN-000011]. For example, he quotes an hourly rate for offshore project managers of $95, while the article he cites suggests that when performed offshore in India the rate should be $34 per hour [ORCLX-PIN-000011; see Pinto Report p. 30, n. 24]. Why he used the higher rate is not explained.

I will address these and other issues in the next Section of this report when I develop a corrected labor rate model for use with the COCOMO II.2000 cost model. Based on the assumptions behind Mr. Pinto's hybrid staffing model, I have no alternative but to question the accuracy, correctness, currency, reasonableness, and soundness of his approach.

- **Mr. Pinto's Step 10: Analyze the Estimated Development Costs**

In his final step, Mr. Pinto multiplies the labor hours he developed using his labor distribution model by the rates he derived for all five scenarios (the four listed in the previous step plus his hybrid model which uses elements of each of them) to develop cost estimates for the JD Edwards EnterpriseOne 8.12 and PeopleSoft 8.X suites of products. Estimated costs for these two suites of products are expressed in cost/program, cost/source line of code, cost/function point, and total cost in Tables 22 and 23 of the Pinto Report.

I looked at the reasonableness of these costs by comparing them to proprietary benchmarks that I recently developed for that purpose [SAP-DJR-000001] and which are included as Reference materials in Appendix B. A summary of my findings is provided in Table 6. Mr. Pinto's estimated costs for the both JD Edwards EnterpriseOne and PeopleSoft suites of products are both higher than the norm. Such high values of predicted cost lead me to believe that Mr. Pinto did not properly use the COCOMO II model when developing his estimates.

b. Summary of Estimates

I am highly confident in my numbers based on the estimates that I independently developed for them. To create confidence in my numbers, I developed three independent COCOMO II.2000 estimates for each of the four suites of products discussed in this report.

The first estimate was that developed by Mr. Pinto using the COCOMO II.1997 model with only changes made needed to correct mathematical errors. This estimate creates a baseline cost for numbers taken from the Pinto Report.

The second estimate was developed using the COCOMO II.2000 with the number of hours assumed per staff-month of effort increased from the 144 hours assumed by Mr. Pinto to the 152 hours used by the COCOMO team when calibrating the model.

The third estimate was also developed using the COCOMO II.2000 model. However, I updated the ratings for the scale and cost drivers used by the model employing my knowledge of the estimation package to calibrate it more closely with the realities of the situation. I continued to use Mr. Pinto's labor rates to develop these results.

The fourth estimate used the new labor rates I derived based primarily on variations to the salaries paid in India to update the numbers generated by the third estimate. As part of this prediction, I also developed the changes needed to take into account my new size estimates for the code written in the C language for the JD Edwards EnterpriseOne applications software.

The final estimate was developed using new size estimates that make corrections to Mr. Pinto's SLOC counts for the JD Edwards suites of products. The first correction fixes a problem with Mr. Pinto's results, brought about when he used his faulty counting utilities. To date, I have not made corrections to the SLOC counts for the PeopleSoft suite of products due to the difficulty in extracting the source code from the materials provided by Plaintiffs. In addition, because Plaintiffs did not provide source code for the Siebel suite of products, to date it has not been possible to make any SLOC corrections for that suite of products.

The results of all of the COCOMO II.2000 model runs are summarized and compared in the Table 53 that follows. This comparison shows a wide range of variation in the estimates due primarily to the following factors: use of COCOMO II.1997 versus COCOMO II.2000, use of 144 hours/staff-month versus 152 hours/staff-month, variations in setting cost and scale drivers,

updated size estimates, and corrections to the labor rate model used by Mr. Pinto and his by role percentage allocation model to life cycle phases.

The project files that I created in the process of developing these estimated for both the USC COCOMO II.2000 and the COSTAR models are provided in SAP-DJR-000002 along with an inventory identifying them by name.

Estimate	Applications Package	Cost ($)			Duration (Months)		
		OPT	LIKELY	PESS	OPT	LIKELY	PESS
Pinto	JDE Enterprise	?[1]	$325.0M	?[1]	?[2]	?[2]	?[2]
	PeopleSoft	?[1]	$647.0M	?[1]	?[2]	?[2]	?[2]
	JDE World	?[1]	$248.0M	?[1]	?[2]	?[2]	?[2]
	Siebel	?[1]	$257.3M	?[1]	?[2]	?[2]	?[2]
TOTALS		**?**	**$1,477.3M**	**?**			
Reifer (COCOMO II.2000 + 152 hrs/SM)	JDE Enterprise	$207.1M	$258.9M	$323.6M	77.8	83.3	89.1
	PeopleSoft	$402.3M	$502.8M	$628.5M	91.9	98.4	105.3
	JDE World	$168.4M	$210.5M	$263.1M	73.0	78.2	83.7
	Siebel	$185.2M	$231.5M	$289.4M	62.4	67.0	71.9
TOTALS		**$963.0M**	**$1,203.7M**	**$1,504.6M**			
Reifer (above plus recalibrate parameters)	JDE Enterprise	$148.4M	$185.6M	$231.9M	34.1	36.5	39.0
	PeopleSoft	$200.2M	$250.3M	$310.4M	35.4	37.8	40.4
	JDE World	$129.1M	$161.4M	$201.8M	32.7	35.0	37.4
	Siebel	$96.1M	$120.2M	$150.2M	30.7	32.9	35.2
TOTALS		**$573.8M**	**$717.5M**	**$894.3M**			
Reifer (above + new labor rate model)	JDE Enterprise	$93.4M	$116.8M	$145.9M	34.1	36.5	39.0
	PeopleSoft	$125.9M	$157.4M	$195.1M	35.4	37.8	40.4
	JDE World	$81.3M	$101.6M	$127.0M	32.7	35.0	37.4
	Siebel	$60.4M	$75.5M	$94.4M	30.7	32.9	35.2
TOTALS		**$361.0M**	**$451.3M**	**$562.4M**			
Reifer (above + size update)	JDE Enterprise	$80.4M	$100.5M	$125.6M	32.6	34.9	37.3
	PeopleSoft	$125.9M	$157.4M	$195.1M	35.4	37.8	40.4
	JDE World	$36.1M	$45.2M	$56.5M	25.7	27.5	29.4
	Siebel	$60.4M	$75.5M	$94.4M	30.7	32.9	35.2
TOTALS		**$302.8M**	**$378.6M**	**$471.6M**			
Reifer (above + optimal schedule)	JDE Enterprise	$56.2M	$70.3M	$87.8M	43.5	46.5	49.7
	PeopleSoft	$88.0M	$110.0M	$137.5M	47.2	50.4	53.8
	JDE World	$25.3M	$31.6M	$39.5M	34.3	36.6	39.2
	Siebel	$42.3M	$52.8M	$66.0M	40.9	43.8	46.9
TOTALS		**$211.8M**	**$264.7M**	**$330.8M**			

Table 53: Summary and Comparison of COCOMO Model Runs

Legend

OPT – optimistic LIKELY – most likely PESS – pessimistic