

EXHIBIT 3

2009

Expert Report of Paul C. Pinto

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

Designated Highly Confidential

Pursuant to Protective Order

Paul C. Pinto
Managing Partner, Sylvan VI. Inc.
November 16, 2009



TEXT REMOVED - NOT RELEVANT TO MOTION

C. Selection of Function Point Analysis

While the benefits to Defendants from infringement rather than development are extensive, this report specifically quantifies a sub-set of those benefits associated with the dollar value of avoided R&D expenses. As described in Section V, I created an estimated cost of development for JD Edwards EnterpriseOne and PeopleSoft applications, using Function Point

TEXT REMOVED - NOT RELEVANT TO MOTION

Analysis. This method of analysis is focused on assessing the size of a software product, in normalized terms that are directly related to the amount of business functionality provided to the end-user of the application. As such, this approach can be applied across a wide range of application development environments and throughout the full life-cycle of the software development effort. When coupled with a series of business metrics, such as productivity and the hourly rates for assigned personnel, the total cost of application development can be readily derived.

The method of Function Point Analysis was introduced in 1979 (by IBM), and is actively maintained by the International Function Point Users Group (“IFPUG”) as part of its Functional Size Measurement Method. Function Point Analysis provides an objective, comparative measure that assists in the evaluation, planning, management, and control of software production. Among other things, it is used, as applied here, to develop an estimated cost of development of a software product.³

I chose to use Function Point Analysis for this assessment because it is recognized by the International Standards Organization (“ISO”) as a valid method for assessing the size of a software product and for deriving the associated cost of product development.⁴ It is also recognized by a number of the world’s largest I.T. consulting companies and has been used by IBM, TCS, and Infosys since its inception. Also, I have considerable experience applying the required techniques in real business scenarios, where it is regularly used to estimate software development efforts and associated costs that are based on a set of defined requirements, which is known as “forward-engineering.” I have also applied this method in situations where legacy software products needed to be redeveloped onto a modern computing platform, while maintaining the existing functionality.

³ International Function Point Users Group, About IFPUG, <http://www.ifpug.org/about>. [ORCLX-PIN-000008]

⁴ International Standard ISO/IEC, 20926, Manual, October 2003, Software engineering - IFPUG 4.1 Unadjusted functional size measurement method - Counting practices manual, http://webstore.iec.ch/preview/info_isoiec20926%7Bed1.0%7Den.pdf. [ORCLX-PIN-000009]

D. Selection of COCOMO Analysis

To confirm the estimates reached through Function Point Analysis for the JD Edwards EnterpriseOne and PeopleSoft products, and to assess the cost of development for the JD Edwards World and Siebel products, I applied an alternate estimating method known as Constructive Cost Model (COCOMO) analysis. COCOMO is an industry-accepted method that provides a reliable approach to performing high-level “top-down” estimating, as a valid alternate method to performing a low-level “bottom-up” analysis as is required for Function Point Analysis.

COCOMO is an algorithm-based software cost estimation model that employs the use of regression formulas, coupled with parameters that were derived from historical project characteristics. The model was originally published in 1981 as a method for estimating the level of effort, project duration, and costs associated with developing software. This original model was referred to as COCOMO 81.⁵

In 2001, the second version of the model, COCOMO II, was published. This recent iteration is better suited for estimating modern software development projects, by providing an updated set of project characteristics that are more aligned with today’s software development tools, iterative approaches, and relational databases. The need for this new model was prompted by the evolution of software development technologies, which moved away from mainframe and overnight batch processing, and moved toward desktop development and code reusability.⁶

COCOMO II estimates the software development effort as a function of a limited set of “scaling drivers” that describe the development process, and a set of “cost drivers” that include subjective assessments about the product, platform, personnel, and project attributes. The end result of a COCOMO II analysis is the estimated total cost of development.

⁵ COCOMO Model II, Center for Systems and Software Engineering, http://csse.usc.edu/csse/research/COCOMOII/cocomo_main.html. [ORCLX-PIN-000003]

⁶ *Id.*

I chose to apply COCOMO II analysis here (which I also refer to generally as “COCOMO”), because it provides a reliable method for confirming the development costs for JD Edwards EnterpriseOne and PeopleSoft that were estimated through Function Point Analysis. COCOMO analysis also allows the JD Edwards EnterpriseOne and PeopleSoft estimates to be reasonably extrapolated to the JD Edwards World and Siebel products, respectively.

TEXT REMOVED - NOT RELEVANT TO MOTION

TEXT REMOVED - NOT RELEVANT TO MOTION

B. Step Two: Count the Number of Source Lines of Code

The next step involved counting Source Lines of Code (“SLOC”) using specially-designed counting utilities. Counting SLOC is a simple procedure that provides an accurate predictor of development effort.¹⁰ When development effort is appropriately attributed to the roles that participate in the Product Development Life-Cycle, and then combined with hourly rates, enough information is available to develop a reliable estimate of the cost of product development.¹¹

Counting SLOCs still requires a certain amount of nuance, however. Imbedded within Source Code are various statements such as: physical lines of code, logical source lines of code, blank lines, and commented (unused or educational) lines of code. Each software development

text files produced at ORCLX-PIN-000024 to ORCLX-PIN-000062.

¹⁰ Software Size Measurement: A Framework for Counting Source Statements, Technical Report CMU/SEI-92-TR-020, ESC-TR-92-020, September 1992, Robert E. Parker, Software Engineering Institute at Carnegie Mellon University, pgs. 13-15. [ORCLX-PIN-000017]

¹¹ *Id.* at 1-15.

language has rules for constructing its Source Code, in the same way that the English language has rules for constructing statements and sentences. These software coding rules, or standards, enable software utilities to be built that can distinguish the different rules and, therefore, count the different types of statements. The end product is the total number of logical Source Lines of Code.

Since 1984, the Software Engineering Institute (SEI), at Carnegie Mellon University, has established standards for defining a Logical Source Code Statement. SEI is a federally-funded research and development center that conducts software engineering research in acquisition, architecture and product lines, process improvement and performance measurement, security, and system interoperability and dependability.¹² I relied on these standards for this portion of my analysis.

In order to use the logical Source Lines of Code count as the foundation for estimating software size and ultimately deriving the total cost of development, I constructed a number of software utilities that counted the logical Source Lines of Code, which are produced as ORCLX-PIN-000066 to ORCLX-PIN-000085. Each line counting utility was specifically designed and tailored to address the specific needs of each type of source code that was analyzed (e.g., COBOL, C, SQL, SQR, etc). Below, Table 4 (ORCLX-PIN-000065 Table 4) is a sample of the output from the automated code counting utility for a series of “C” program files.

Sample SLOC Counting Utility Output (for JDE EnterpriseOne example)		
File Name	Total Lines of Source Code	Logical Source Lines of Code
n4002340.c	701	379 SLOC
n4002350.c	984	519 SLOC
n4002380.c	882	315 SLOC
n4002400.c	192	81 SLOC
n4002440.c	801	410 SLOC

Table 4 - Sample SLOC Counting

In sum, Step Two involved counting the number of logical SLOC within each grouping, which then served as the basis for establishing the size of the code base in subsequent steps. The

¹² *Id.* at 13-21.

Source Code components, as identified in Step One, were used as the input for determining the number of logical SLOC. Below, Table 5 (ORCLX-PIN-000065 Table 5) displays the size of code base, for the identified groupings, expressed as the number of logical SLOC.

Number of Source Lines of Code			
Software Product Version	Programming Language (stratum)	Number of logical Source Lines of Code	Totals
JDE EnterpriseOne Version 8.12	C	6,906,168	7,774,791 SLOC
	Java J2EE	868,623	
PeopleSoft Version 8.X	COBOL/400	2,057,468	7,650,493 SLOC
	SQC, SQR, DMS and SQL	2,282,005	
	RPT and MDL	244,760	
	PeopleCode	3,066,260	
Totals:		15,425,284	15,425,284 SLOC

Table 5 - Source Lines of Code

TEXT REMOVED - NOT RELEVANT TO MOTION

F. Step Six: Distribute the Effort across the Product Development Life-Cycle

After determining the amount of PHE required to perform full life-cycle product development, it is necessary to distribute that effort across the Product Development Life-Cycle (PDLC). This is an interim step to ultimately assigning particular hours to specific roles that perform the activities within the PDLC. The PDLC refers to the activities associated with constructing a software application from inception to deployment, and underpins many types of software development methodologies, which form the framework for estimating the software development effort.

The International Software Benchmarking Standards Group (ISBSG) defines the standard phases for the PDLC as Plan, Specify, Design, Build, Test, and Implement.¹⁹

TEXT REMOVED - NOT RELEVANT TO MOTION

¹⁹ Industry Software Cost, Quality and Productivity Benchmarks, whitepaper, April 2004, by Donald J Reifer, Reifer Consultants, Inc., <http://www.compaid.com/caiinternet/ezine/Reifer-Benchmarks.pdf>. [ORCLX-PIN-000014]

TEXT REMOVED - NOT RELEVANT TO MOTION

In support of developing these estimates, I chose to use the Constructive Cost Model (COCOMO), which is also accepted as a valid approach to estimating, but from a “top-down” perspective, as opposed to performing a detailed-level Function Point Analysis.

A. Constructive Cost Model (COCOMO)

COCOMO is an algorithm-based software cost estimation model that employs the use of regression formulas, coupled with parameters that were derived from historical project characteristics. The model was originally published in 1981, by Barry Boehm, as a method for

TEXT REMOVED - NOT RELEVANT TO MOTION

D. COCOMO II Estimate for JD Edwards World

In performing this top-down analysis for JD Edwards World, I assumed that the product had similar functionality to that of JD Edwards EnterpriseOne. This assumption is based on the fact that JD Edwards World was the predecessor to JD Edwards EnterpriseOne, and that it was predominantly developed in the RPG programming language as opposed to COBOL.³¹ As a result of this base assumption, I assumed that JD Edwards World contains the same number of SLOC as JD Edwards EnterpriseOne (specifically, 7,774,791 SLOC), as well as similar application characteristics to those found in the JD Edwards EnterpriseOne application, with two modifications. The modifications are associated with Reusability and Platform Volatility stemming from its underlying technology for the product (namely, that JD Edwards World was written in RPG programming language and is run on the IBM I-Series platform), with my assessments annotated in Table 27 (ORCLX-PIN-000065 Table 27), below.

³¹ Oracle Indefinitely Extends the life of JDE World, IT Jungle Newsletter, April 24, 2008, by Timothy Prickett Morgan, <http://www.itjungle.com/tfh/tfh042406-story02.html>. [ORCLX-PIN-000010]

JD Edwards World Lines of Code	
Number of Source Lines of Code	7,774,791
Number of Source Lines of Code (in 1,000s)	7,775

Table 27a - COCOMO Analysis for JDE World: SLOC

Scaling Characteristic		
Categories	Assessment	Weighting
Precedentedness	High	1.62
Development Flexibility	High	2.43
Architecture / Risk Resolution	High	1.69
Team Cohesion	High	1.98
Process Maturity	High	1.82
Total:		9.54
Process Scale Factor:		1.1054

Table 27b - COCOMO Analysis for JDE World: Scaling

Effort Characteristics			
Category	Effort Drivers	Rating	Weighting
Product	Required Software Reliability	High	1.15
	Database Size	High	1.09
	Product Complexity	High	1.15
	Required Reusability	Nominal	1
	Documentation to match lifecycle needs	High	1.06
Platform	Execution Time Constraint	Nominal	1
	Main Storage Constraint	Nominal	1
	Platform Volatility	Low	0.87
Personnel	Analyst Capability	Very High	0.67
	Programmer Capability	Very High	0.74
	Personnel continuity	Very High	0.84
	Applications Experience	Very High	0.81
	Platform Experience	Very High	0.81
	Language and Tool Experience	Very High	0.84
Project	Use of Software Tools	High	0.86
	Multi-site operation	High	0.92
	Required Development Schedule	High	1
Overall Weighting Factor:			0.241417477

Table 27c - COCOMO Analysis for JDE World: Effort

JDE World Estimated Effort	
Person Months	11,822
Person Hours	1,702,412
Average Blended Rate	\$145.72
Total Cost	\$248,073,123

Table 27d - COCOMO Analysis for JDE World: Cost

As a result of the performing COCOMO II analysis, the model indicated that the development effort would require 11,822 person-month of effort, or 1,702,412 person-hours of effort. When the number of person hours is multiplied by the average blended rate of

\$145.72/hour, for the “Hybrid” staffing scenario (identified in the Function Point Analysis discussions, above), the estimated cost of development is calculated to be \$248,073,123. In adopting similar proportions to the cost ranges estimated for JD Edwards EnterpriseOne, the JD Edwards World development costs would have ranged between \$172M and \$581M, depending on the selected staffing model.

E. COCOMO II Estimate for Siebel

In performing this top-down analysis for Siebel, I based my analysis on the assumption that the Siebel product contained 79.4% more functionality than the PeopleSoft CRM module, including its use of PeopleTools. This analysis was based on the fact that Siebel contained 7,593 tables (4,435 for SIA and 3,158 for HOR³²), while PeopleSoft CRM contained 4,233 tables. This method of sizing provides a reasonable, while simplistic, approach to estimating the relative amount of functionality between software products that are built in similar technologies. The reasonableness of this approach is supported by the fact that PeopleSoft CRM was acknowledged as a competitor to Siebel, and that Siebel was acknowledged as the industry leader in the CRM space and offered significantly greater functionality than PeopleSoft CRM.³³ As a result of this analysis, it is estimated that Siebel contains 1,195,091 Source Lines of Code (SLOC), and similar application characteristics to those found in the PeopleSoft, with modifications associated with the Personnel characteristics stemming from the use of a non-integrated development environment (not PeopleCode with PeopleTools), which are annotated in Table 28 (ORCLX-PIN-000065 Table 28), below.

Siebel Source Lines of Code	
Number of Source Lines of Code	1,195,091
Number of Source Lines of Code (in 1,000s)	1,195

Table 28a - COCOMO Analysis for Siebel: SLOC

³² Siebel SIA refers to Siebel Industry Application, while Siebel HOR refers to Siebel’s Horizontal Application. Both are components of Siebel available to customers as part of Siebel’s CRM product. Table numbers are identified in ORCLX-PIN-000004 and ORCLX-PIN-000015.

³³ The Forrester Wave: Enterprise CRM Suites, Q3 2008, by William Band, August 28, 2008, updated September 2, 2008. [ORCLX-PIN-000006]

Scaling Characteristic		
Categories	Assessment	Weighting
Precedentedness	Nominal	2.43
Development Flexibility	Nominal	3.64
Architecture / Risk Resolution	Nominal	2.53
Team Cohesion	Nominal	2.97
Process Maturity	Nominal	2.73
Total:		14.3
Process Scale Factor:		1.153

Table 28b - COCOMO Analysis for Siebel: Scaling

Effort Characteristics			
Category	Effort Drivers	Rating	Weighting
Product	Required Software Reliability	High	1.15
	Database Size	High	1.09
	Product Complexity	High	1.15
	Required Reusability	High	1.14
	Documentation to match lifecycle needs	High	1.06
Platform	Execution Time Constraint	Nominal	1
	Main Storage Constraint	Nominal	1
	Platform Volatility	Nominal	1
Personnel	Analyst Capability	High	0.83
	Programmer Capability	High	0.87
	Personnel continuity	Nominal	1
	Applications Experience	Nominal	1
	Platform Experience	Nominal	1
	Language and Tool Experience	Nominal	1
Project	Use of Software Tools	Nominal	1
	Multi-site operation	Nominal	1
	Required Development Schedule	Nominal	1
Overall Weighting Factor:			1.257854015

Table 28c - COCOMO Analysis for Siebel: Effort

JDE Siebel Estimated Effort	
Person Months	10,890
Person Hours	1,568,203
Average Blended Rate	\$164.08
Total Cost	\$257,306,140

Table 28d - COCOMO Analysis for Siebel: Cost

As a result of the performing COCOMO II analysis, the model indicated that the development effort would require 10,890 person-month of effort, or 1,568,203 person-hours of effort. When the number of person hours is multiplied by the average blended rate of \$164.08/hour, for the Hybrid scenario, the estimated cost of development is calculated to be \$257,306,140. In adopting similar proportions to the cost ranges estimated for PeopleSoft, the

Siebel development costs would have ranged between \$198M and \$573M, depending on the selected staffing model.

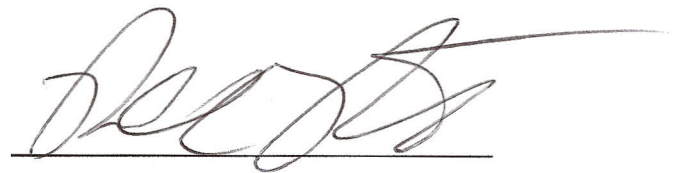
TEXT REMOVED - NOT RELEVANT TO MOTION

X. OPTION TO REVISE

I reserve the right to modify and/or supplement this report and/or the opinions set forth herein if additional damages rulings are made by the Court and/or additional evidence becomes available.

I, Paul C. Pinto, having conducted the aforementioned analysis and having authored this report, confirm that the opinions contained herein represent a fair and unbiased analysis of the facts presented to me.

Date: 11/16/09

A handwritten signature in black ink, appearing to read "Paul C. Pinto", written over a horizontal line.

Paul C. Pinto